

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

آرماندو فاکس و دیوید پترسون

ترجمه توسط آرش پورحبیبی و صبا جمالیان

ویرایش ۰/۱

۱۴۰۲ آذر ۸

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

ویرایش ترجمه: ۰/۰
ویرایش متن اصلی: ۸/۰ ب

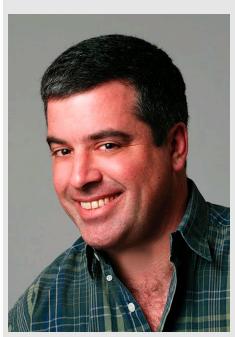
نمای پس‌زمینه روی جلد عکسی از آب‌گذر سگوبیا در کشور اسپانیا است. ما آن را بهعنوان نمونه‌ای از طراحی زیبا، بادوام و ماندگار انتخاب کردیم. طول کامل این آب‌گذر حدود ۲۰ مایل (۳۲ کیلومتر) است و توسط رومیان در قرن اول یا دوم پس از میلاد ساخته شده است. این عکس مربوط به قطعه‌ای به طول ۸/۰ کیلومتری و ارتفاع ۲۸ 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 3.0 CC-BY-SA: https://commons.wikimedia.org/wiki/Aqueduct#/media/File:Aqueduct_of_Segovia_02.jpg

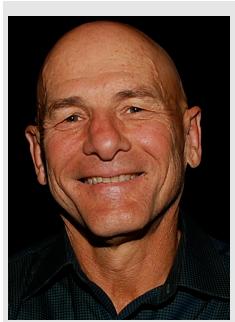
این کتاب با استفاده از LATEX و مجموعه‌ای از بسته‌های رویی تهیه شده است که همگی بهصورت آزاد و رایگان در <http://github.com/armandofox/latex2ebook> در دسترس هستند.

آرتور کلیچوکوف 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 بنا پی دریافت کرد. در سال ۲۰۱۶، او و دیگر نویسنده‌این کتاب، دیوید پترسون جایزه نویبدخش 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 پروژه‌های 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 آمده‌ن در آکادمی ملی مهندسی، آکادمی ملی علوم و تالار مشاهیر مهندسی سیلیکون ولی؛ برگزیده شدن به عنوان قلوب موزه تاریخ رایانه، IEEE، ACM و هر دو سازمان AAAS. وی اخیراً به همراه جان هنیسی، استاد دانشگاه استنفورد، به دلیل کارشان بر روی RISC و همچنین رویکرد کمی آن‌ها به معماری و طراحی رایانه، مشترکاً موفق به کسب جایزه تورینگ شدند. وی همچنین جوازی متعددی در زمینه آموزشی دریافت کرده است، از جمله: جایزه مدرس برجسته دانشگاه کالیفرنیا، برکلی، جایزه مدرس برجسته Karl V. Karlstrom از ACM، مدال آموزشی IEEE و همچنین جایزه تدریس در مقطع کارشناسی از IEEE Mulligan. پیش از برندۀ شدن جایزه 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 فوتبال و مسابقات دوچرخه‌سواری خیریه.

درباره مترجمان



آرش پورحبيبي محقق و توسعه‌دهنده حوزه سامانه‌های رایانه‌ای است. وی در حال حاضر در شرکت اوراکل به عنوان مهندس ارشد بر روی سرویس مای‌اس‌کیوال هیت‌ویو (MySQL HeatWave) کار می‌کند که سرویس یکپارچه‌ای بری پایگاه داده مای‌اس‌کیوال برای پردازش تراکنش‌ها، تجزیه و تحلیل بلادرنگ داده در انبارهای داده و دریاچه‌های داده و همچنین یادگیری ماشین است. او دکتراخود را در رشته علوم رایانه و ارتباطات از مؤسسه فناوری فدرال لوزان سوئیس (EPFL) در سال ۱۴۰۰ دریافت کرد. پایان‌نامه دکتراخوی او بر گریز از پرداخت هزینه و سربار فراخوانی‌های رویه‌ای دور دست (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 خوره 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 اطمنان سایت در شرکت بِریز (Braze) است و به توسعه زیرساخت‌های بری آن‌ها کمک می‌کند. قبل از بِریز، او برای ارائه نرم‌افزار رلیتیویتی‌وان (RelativityOne) به صورت سرویس یکپارچه بری 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 چاپک و رایانش ابری ..	۲
بخش اول: چارچوب‌ها و زبان‌های برنامه‌نویسی SaaS		
۲	چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم	۵۴
۳	ریزسرویس‌ها، API‌ها و REST	۵۶
۴	ریلز، یک چارچوب مدل-نما-کنترل‌گر	۵۸
۵	تجربه‌های پیشرفت‌هه برنامه‌نویسی برای SaaS	۶۰
۶	مقدمه‌ای بر جاوا‌اسکریپت	۶۲
بخش دوم: توسعه نرم‌افزار با رویکردی چاپک		
۷	توسعه رفتارمحور و روایت‌های کاربری	۶۴
۸	توسعه آزمون محور	۶۶
۹	میراث‌کد، بازسازی، و روش‌های چاپک	۶۸
۱۰	تیم‌های چاپک	۷۰
۱۱	الگوهای طراحی برای SaaS	۷۲
۱۲	توسعه/عملیات: به کاراندازی، کارابی، قابلیت اطمینان و امنیت کاربردی ...	۷۴
	پس‌گفتار	۷۶

فهرست مطالب

ح	پیش‌گفتار مترجمان
۲	۱ مقدمه‌ای بر نرم‌افزار به صورت یک سرویس، توسعه چابک و رایانش ابری
۵	۱-۱ مقدمه
۶	۲-۱ فرآیندهای توسعه نرم‌افزار: طرح-و-ثبت
۱۳	۳-۱ فرآیندهای توسعه نرم‌افزار: منشور چابک
۱۸	۴-۱ تصمیم‌گیری کیفیت نرم‌افزار: آزمون
۲۱	۵-۱ بهره‌وری: اختصار، سنتز، بازکاربرد و ابزارها
۲۵	۶-۱ نرم‌افزار به صورت یک سرویس و معماری سرویس‌گرا
۲۸	۷-۱ به کاراندازی SaaS: رایانش ابری
۳۱	۸-۱ به کاراندازی SaaS: مروگرها و تلفن همراه
۳۷	۹-۱ زیبائگ در مقابله میراث کد
۳۹	۱۰-۱ گذرنی بر این کتاب و نحوه استفاده از آن
۴۳	۱۱-۱ باورهای نادرست و خطوهای پنهان
۴۵	۱۲-۱ نکات پایانی: مهندسی نرم‌افزار چیزی فراتر از برنامه‌نویسی است
۵۳	اول نرم‌افزار به صورت یک سرویس: چارچوب‌ها و زبان‌ها
۵۴	۲ چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم
۵۶	۳ ریزسرویس‌ها، API‌ها و REST
۵۸	۴ ریلز، یک چارچوب مدل-نمای-کنترل‌گر
۶۰	۵ تجریدهای پیشرفته برنامه‌نویسی برای SaaS
۶۲	۶ مقدمه‌ای بر جاوا‌اسکریپت
۶۳	دوم توسعه نرم‌افزار با رویکردی چابک
۶۴	۷ توسعه رفتارمحور و روایت‌های کاربری

۶۶	۸ توسعه آزمون محور
۶۸	۹ میراث‌کد، بازسازی، و روش‌های چابک
۷۰	۱۰ تیم‌های چابک
۷۲	۱۱ الگوهای طراحی برای SaaS
۷۴	۱۲ توسعه/عملیات: به کاراندازی، کارایی، قابلیت اطمینان و امنیت کاربردی
۷۶	۱۳ پسنگفتار
۷۷	واژه‌نامه انگلیسی به فارسی
۸۸	واژه‌نامه فارسی به انگلیسی
۱۰۱	فهرست اختصارات

پیش‌گفتار مترجمان

زبان، نقشهٔ راه یک فرهنگ است. به شما می‌گوید مردم آن از کجا آمده‌اند و به کجا می‌روند.
—^۱ریتا مای براون

زبان یکی از ارکان تعریف یک فرهنگ است و دایرهٔ کوچک لغات تخصصی دنیای رایانه^۲ در زبان فارسی باعث ایجاد ضعف فرهنگ تخصصی حوزهٔ رایانه^۳ بین فارسی‌زبانان شده است. باور ابتدایی ما این بود که زبان انگلیسی پیش‌زمینه‌ای مهم برای یادگیری اصولی علوم رایانه^۴ و یا مهندسی رایانه^۵ 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 ناعالتی است، بلکه از بزرگترین چالش‌های صنعت فناوری، نرم‌افزار^۶ و رایانه است.

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 ترجمهٔ 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^۷ بپیونددند و مدخل‌های جدیدی را با استناد به این کتاب در ویکی‌پدیای فارسی ایجاد کنند.

در این ترجمهٔ ما تلاش کردیم که به خواننده کمک کنیم تا در حین یادگیری مطالب، با کلمات کلیدی و تخصصی دنیای رایانه، با زبان اصلی نیز آشنایی پیدا کند. این آشنایی می‌تواند به ارتباط ساده‌تر بین فارسی‌زبانان و متون و منابع انگلیسی منجر شود. از این رو، 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 واژهنامه شامل تمامی این کلمات کلیدی و ترجمهٔ فارسی آن‌ها به‌طور دوطرفه قرار داده‌ایم. تلاش کردیم که با وسوس و دقت در ترجمه چنین کلماتی، نه تنها به انسجام متن فارسی کمک کنیم، بلکه به خواننده نیز کمک کنیم تا با معادل زبان اصلی لغات نیز آشنا شود.

علوم رایانه یا مهندسی رایانه؟
در بسیاری از کشورها، رشته علوم رایانه رشته اصلی در زمینه رایانه است و رشته مهندسی رایانه کمی مرتبط با مهندسی برق است. در ایران اما، علوم رایانه بیشتر به قسمت نظری این حوزه اشاره دارد و رشته مهندسی رایانه در واقع همان علوم رایانه متدائل در سایر کشورهاست.

^۱Rita Mae Brown

^۲Computer

^۳ با وجود اینکه واژهٔ کامپیوتر در زبان فارسی واژه‌ای جاافتاده است، ما تصمیم گرفتیم که از «رایانه» به عنوان معادل فارسی Computer استفاده کنیم. یکی از دلایل اصلی این انتخاب، هماهنگی و انسجام بهتر با سایر اصطلاحات وابسته مثل «رایانش» (Computing) بوده است.

^۴Computer Science

^۵Computer Engineering

^۶Software

^۷Wikipedia

بارها مشاهده شده که متخصصان فارسی‌زبان علوم رایانه در صحبت‌ها و نوشتارهای فارسی خود از لغات عمومی یا تخصصی زیادی به زبان انگلیسی استفاده می‌کنند. این لغات یا اصطلاحات عموماً با تلفظ نادرست و گاهی با معنای اشتباه، خارج از موضوع و یا بهطور کلی نادرست استفاده می‌شوند. تکرار این عمل منجر به تولید یک زبان میانی شده است که اصول تعریف شده درستی ندارد، نه انگلیسی است و نه فارسی اصولی. ترجمه کتب مرجع، مخصوصاً اگر از اصول و استاندارد مشترک استفاده کنند بهترین ابزار برای مبارزه با تناقض‌های زبان‌های میانی است. در این کتاب سعی کردیم که تا حد امکان از اصطلاحات و کلمات جایگزین مصوب استفاده کنیم و حتی تعداد قابل توجهی کلمه و اصطلاح جدید نیز ابداع کردیم که امیدواریم مورد استقبال خوانندگان و متخصصین این حوزه قرار گیرد.

کتاب پیش رو، روش‌های اصولی برای طراحی و مهندسی نرم‌افزار مدرن را ارائه می‌دهد. برای انتقال بهتر مفاهیم، به یک زبان برنامه‌نویسی^۸ به عنوان ابزار اصلی نیاز است که کتاب زبان برنامه‌نویسی رویی^۹ و به همراه آن چارچوب^{۱۰} رویی آن ریلز^{۱۱} 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 و دقیق از پروژه‌ها، تمرین‌ها و ابزارهایی همراه است که روی یک مخزن^{۱۲} عمومی در دسترس هستند. هدف ما از ترجمه، همراهی با همین اکوسیستم برای معرفی این کتاب به عنوان یک مرجع تخصصی و حتی دانشگاهی برای علم مهندسی نرم‌افزار^{۱۳} است. از این روسوی بر این داریم که کلیه مجموعه ارائه شده را در آینده نزدیک به فارسی تهیه کنیم.

سعی کردیم در ترجمه این کتاب از آخرین اصول نگارشی زبان فارسی پیروی کنیم. در این راستا تلاش کردیم تا از نکته‌های نگارش ترجمه متون تخصصی رایانه که توسط دکتر محمد قدسی و دکتر حمید ضرابیزاده تدوین شده است^{۱۴} نیز پیروی کنیم. از لطف و همکاری خانم مونا اصفهانی برای ویراستاری این اثر نیز سپاسگزاریم. همچنین لازم می‌دانیم از پدیدآورندگان بسته زی پرشین^{۱۵} (XEPersian) و جامعه توسعه‌دهندگان حوزه *TeX* فارسی، مخصوصاً آقای وفا خلیقی، که حروف‌چینی به زبان فارسی و با کیفیت بالا را ممکن کرده‌اند قدردانی کنیم. سپاسگزار زحمات آقای صابر راستی‌کردار نیز برای تهیه مجموعه قلم^{۱۶} وزیر (شامل وزیرمن^{۱۷} و وزیرک^{۱۸}) و در اختیار گذاشتن آن به صورت رایگان هستیم؛ بادشان گرامی باد.

در آخر، بابت اشتباهات احتمالی و بی‌دقیقتی‌های ما در ترجمة این کتاب، به خصوص در این نسخه‌های ابتدایی، پیش‌پیش عذرخواهی می‌کنیم و از شما دعوت می‌کنیم پیشنهادها و موارد اصلاحی خود را حتماً برای ما ارسال کنید. نهایتاً از آرماندو فاکس^{۱۹} و دیوید پترسون^{۲۰} که سخاوتمندانه به ما اجازه دادند تا این کتاب را به فارسی ترجمه کنیم و ترجمه فارسی آن را به رایگان در اختیار فارسی‌زبانان بگذاریم صمیمانه سپاسگزاریم. این ترجمه بر اساس ویرایش ۲/۰ ب ۸ نسخه اصلی کتاب است. امیدواریم که محتوای این کتاب به خوانندگان کمک کند تا درک و فهم بهتری نسبت به طراحی نرم‌افزار پیدا کنند.

آرش و صبا
پاییز ۱۴۰۲

⁸ Programming Language

⁹ Ruby (Programming Language)

¹⁰ Framework

¹¹ Ruby On Rails

¹² Repository (Version Control)

¹³ Software Engineering

¹⁴ Font (Computer)

¹⁵ Armando Fox

¹⁶ David Patterson

پیوندها

<https://sharif.edu/~ghodsi/typing-rules.pdf>^۱

<https://ctan.org/pkg/xepersian>^۲

<https://rastikerdar.github.io/vazirmatn/>^۳

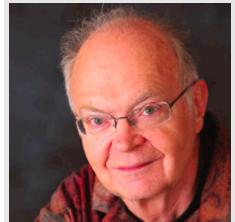
<https://rastikerdar.github.io/vazir-code-font/>^۴

مقدمه‌ای بر نرم‌افزار به صورت یک سرویس، توسعه 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 (۱۹۸۴-)، یکی از برجسته‌ترین دانشمندان علم رایانه است. او جایزهٔ تورینگ را در سال ۱۹۷۴ به خاطر مشارکت‌های عمدۀ 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 TELEX را ابداع کرد که این کتاب با استفاده از آن تهیه شده است.



۵	۱-۱	مقدمه
۶	۲-۱	فرآیندهای توسعه نرم‌افزار: طرح-و-ثبت
۱۳	۳-۱	فرآیندهای توسعه نرم‌افزار: منشور چاپک
۱۸	۴-۱	تضمين کيفيت نرم‌افزار: آزمون
۲۱	۵-۱	بهره‌وری: اختصار، سنتز، بازکاربرد و ابزارها
۲۵	۶-۱	نرم‌افزار به صورت یک سرویس و معماری سرویس‌گرا
۲۸	۷-۱	به‌کاراندازی SaaS: رایانش ابری
۳۱	۸-۱	به‌کاراندازی SaaS: مرورگرها و تلفن همراه
۳۷	۹-۱	زیباگد در مقابل میراث کد
۳۹	۱۰-۱	گذری بر این کتاب و نحوه استفاده از آن
۴۳	۱۱-۱	باورهای نادرست و خطرهای پنهان
۴۵	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...	اما نه اگر می‌خواهید...
به اصول طراحی نرم‌افزار اجازه دهید تا ارزیابی و ایجاد فناوری‌های جدید را به شما اطلاع دهد	فقط چارچوب فلان را بیاموزید، بدون اینکه اصول طراحی آن را بدانید
چگونگی یادگیری چارچوب‌ها و زبان‌های برنامه‌نویسی جدید را یاد بگیرید و به سرعت از آنها استفاده کنید	یک آموزش گام به گام مشابه یک «دستور پخت» را برای یک چارچوب یا زبان برنامه‌نویسی خاص دنبال کنید
با انجام دادن و تمرین کردن یاد بگیرید	فقط با خواندن و تماسای فیلم یاد بگیرید

پیش‌نیاز فنی این فصل داشتن آگاهی ابتدایی در مورد **HTML**، زبان نشانه‌گذاری ابرمتنی است که زبان اصلی یا میانجی وب است. ما مطالعه 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 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.

- فرآیندهای طرح-و-ثبت و یا چرخه‌های حیات محصول متنکی به طرح و برنامه‌ریزی دقیق و اولیه است، در حالی که 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** (SaaP) 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 توسعه 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 نرم‌افزارهای کاربردی که به‌طور خاص برای این دستگاه‌ها ساخته می‌شوند، می‌توان از همان ابزارهایی بهره برد که برای تحت SaaS مروگرهای دسکتاب استفاده می‌شود. در این راه می‌توان از **طراحی وب واکنش‌گرا** برای انتساب خودکار با اندازه‌های مختلف صفحه نمایش و در عین سازگاری با کاربران دارای معلومات 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 (نوری ۱۳۰۱). 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 شده، در صورتی که در مورد و بگاه آمازون این زمان‌ها محاسبه شده‌اند (راپتیشن ۲۰۱۳). نرخ خطأ برای خطاهای اساسی در فرم‌های که به شرکت‌های پیمه فروخته شده محاسبه شده (هورسلی ۲۰۱۳). بسیاری از کارشناسان حوزه امنیت، سایت را نامن خوانند، احتمالاً به این دلیل که توسعه‌دهندگان تحت فشار زیادی برای ایجاد قابلیت‌های مورد نظر بودند و توجه کمی به امنیت داشتند (هینگکتون ۲۰۱۳).

۱-۱ مقدمه

خیلی ساده است. این وبگاهی است که می‌توانید در آن بیمه‌های درمانی مقرن به صرفه تهیه کنید، همانطوری که از (وبگاه) کایاک^۱ بلیط هوایی‌ها تهیه می‌کنید یا از (وبگاه) آمازون^۲ 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 تشويقي).

— رئیس جمهور باراک اوباما، سخنرانی در مورد لایحه مراقبت مقرن به صرفه^۳، دانشگاه دولتی پرنス جرج، مریلند، ۲۶ سپتامبر ۲۰۱۳

...اکنون شش ماه از آغاز به کار بازارچه بیمه مربوط به لایحه مراقبت مقرن به صرفه می‌گذرد. می‌توانم بگویم این حرکت تا اینجای کار با پستی و بلندی‌های همراه بوده و فکر می‌کنم همه متوجه یاشند که من هم از اینکه این حرکت، خوب می‌دونید، با مشکلات مختلف خریداری می‌کنم. خوشحال نیستم و این مسائل فکر من را مشغول کرده.

— رئیس جمهور باراک اوباما، سخنرانی در مورد لایحه مراقبت مقرن به صرفه، ملاقات با خبرنگاران در کاخ سفید، ۱۴ نوامبر ۲۰۱۳

وقتی لایحه مراقبت مقرن به صرفه (ACA) در سال ۲۰۱۰ تصویب شد، به عنوان بلندپروازانه‌ترین برنامه اجتماعی آمریکا طی چند دهه شناخته می‌شد، و می‌توان گفت گل سرسبد فعالیت‌های دولت اوباما بود. همانطور که میلیون‌ها نفر از طریق و بگاه^۴ آمازون اجناس مختلف خریداری می‌کنند، —که به همان وبگاه لایحه مراقبت مقرن 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 نرم‌افزارهای بسازند که می‌توانند به مشتریان بسیار بیشتری خدمات بسیار بهتری ارائه کنند؟ در حالی که رسانه‌ها بعده‌ها از تصمیم‌های سوال‌برانگیز زیادی پرده برداشتند، تعداد شگفت‌آوری از موارد اتهام به روش‌های به کار گرفته شده در تولید و

¹Kayak (Travel Search Engine)

²Amazon.com

³Affordable Care Act

⁴Website

فصل ۱. مقدمه‌ای بر توسعه چابک و SAAS

توسعه نرم‌افزار^۵ مربوط بودند (جانسون و رید ۲۰۱۳). با توجه به رویکرد آن‌ها، همانطور که یک تحلیل‌گر گفت، «اگر به نتیجه می‌رسیدند جای تعجب بود.» (جانسون ۲۰۱۳^۶) 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 شکست‌خورده 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 توسعه 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^۷ 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^۸ نرم‌افزار بود تا به نوعی شورش کنند. در حالی که منشور چابک^۹ هنگام معرفی اش بسیار بحث‌برانگیز بود، اما با گذشت زمان توسعه نرم‌افزاری چابک^{۱۰} بر منتظران خود غلبه کرد. روش چابک 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^{۱۱} این امکان را فراهم می‌کند تا با ترکیب تعداد زیادی سرویس نرم‌افزاری کوچک، که هر یک توسط گروه‌های کوچک چابک توسعه یافته و اداره می‌شوند، خدمات نرم‌افزاری بزرگ و موفقی همانند آمازون ارائه داد.

نکته 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 چابک 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 چابک، 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 فرآیندهای توسعه نرم‌افزار: طرح-و-ثبت

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

⁵Software Development

⁶Affordable Care Act

⁷Plan-and-Document

⁸Developer (Software)

⁹Agile Manifesto

¹⁰Agile

¹¹Service-oriented Architecture

قانون دوم واينبرگ، ۱۹۸۷، منسوب به جرالد واينبرگ^{۱۲}، دانشمند رايانيه در دانشگاه نيراسكا^{۱۳}

غیرقابل پيش‌بياني بودن توسعه نرم‌افزار در اواخر دهه ۱۹۶۰ به همراه فحايي مانند ACA تحقيقاتي در مورد توسعه نرم‌افزارهای باکيقيت در زمان‌بندی و بودجه قابل پيش‌بياني انجاميد. همانند ساير رشته‌های مهندسي، عبارت **مهندسي نرم‌افزار** خلق شد (ناور و زندل ۱۹۶۹). هدف يافتن روش‌هایي برای ساخت نرم‌افزارهایي بود که همانند ساختن پل‌ها در مهندسي عمران، از نظر كيفيت، هزينه و زمان قابل پيش‌بياني باشدند.

يکي از تلاش‌های مهندسي نرم‌افزار اعمال يک روش مهندسي منظم بر روند توسعه^{۱۴} معمولاً بي برنامه نرم‌افزار بود. بر همين اساس، تاكيد بر آن بود که برای انجام پروژه قبل از شروع به برنامه‌نويسی^{۱۵}، طرحی شامل مستندات كامل و با جزئيات وجود داشته باشد که پيشرفت پروژه را بتوان با آن بررسی و اندازه‌گيري کرد. در ادامه، هر تغيير در پروژه باید در مستندات و طرح نيز اعمال شود.

هدف همه اين فرآيندها توسعه نرم‌افزار «طرح-و-ثبت» اين است که با استفاده از مستندسازی پيش‌بياني‌پذيری را افزایش دهند. بدبهی است هر بار اهداف پروژه تغيير کند، مستندات نيز باید متعاقباً عوض شوند. برخی نويسندگان كتاب‌های دانشگاهی اين مطلب را اين‌طور بيان کردن (ليتبريج و لاگانير ۲۰۰۲؛ براد ۲۰۰۱) به:^{۱۶}

مي‌بايست از تمامي مراحل توسعه نرم‌افزار مستندات تهييه کرد، که اين شامل نيازمندي‌ها، طراحی‌ها، راهنمای کاربران، دستورالعمل‌های آزمون‌گران^{۱۷} و طرح‌های پروژه می‌شود.

تيموتى ليتبريج^{۱۸} و رابت لاگانير^{۱۹}، ۲۰۰۲

مستندسازی خونی است که در رگ‌های مهندسي نرم‌افزار جريان دارد.

اريك براد^{۲۰}، ۲۰۰۱

(جاشيه‌های اين چنینی در اين کتاب با هدف ارائه توضیحات اضافی در مورد زمینه تاریخی و یا چشم‌اندازی در مورد مطلب مربوطه هستند. طالعه اين قسمت‌ها اختیاري است اما همانطور که جورج سانتيانا گفت، «کسانی که تاريخ نمی‌دانند، محکوم به تکرار آن هستند.») شركت فناوري گروه سی‌جي‌آی مناقصه ساخت قسمت پسین ویگا ACA را بروزه شد. تخمين اوليه ۹۶ ميليون دلاري آن‌ها در ادامه به ۲۹۲ ميليون دلار افزایش پيدا کرد (يکل ۲۰۱۳). همين شركت در ساخت يك سامانه ثبت اسلحه در کانادا هم درگير بود که در آن نيز هزينه از تخمين اوليه ۲۰ ميليون دلار به دو مليارد دلار جهش داشت. وقتی سازمان MIT تحقيقاتي نرم‌افزار تأخير بيشتری داشته باشد، در نهايتم درآمد بيشتری هم خواهد داشت. بنابرين هنر واقعي در چانه‌زنی در مفاد قرارداد و بندهای جريمه است. همانطور که يکي از مفسران در مورد ACA گفت (هاوارد ۲۰۱۳): «شركت‌هایي که موفق می‌شوند اين‌گونه پروژه‌ها را بگیرند، شركت‌هایي هستند که در گرفتن پروژه مهارت دارند و معمولاً نه آن‌هایي که در اجرای آن مهارت دارند.» مفسر دیگری معتقد

¹²Gerald Weinberg

¹³University of Nebraska

¹⁴Develop (Software)

¹⁵Programming

¹⁶Tester

¹⁷Timothy Lethbridge

¹⁸Robert Laganiere

¹⁹Eric Braude

²⁰Process

²¹Life Cycle

فصل ۱. مقدمه‌ای بر توسعه چاپک و SAAS

بود روش طرح-و-ثبت برای نیازها و شیوه‌های امروزی مناسب نیست، به خصوص در شرایطی که پیمان‌کاران دولتی تمرکز خود را روی بیشینه کردن سود می‌گذارند (چونگ ۲۰۱۳).^{۲۰} نسخه اولیه این روش توسعه 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^{۲۲}

۴- درستی‌سننجی

۵- بهره‌برداری و نگهداری^{۲۳}

با این فرض که هرچه اشکالات زودتر پیدا شوند، درست کردن شان ارزان‌تر خواهد بود، فلسفه این فرآیند تکمیل یک فاز قبل از وارد شدن 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 فرآیند سال‌ها به طول بینجامد، مستندسازی^{۲۴} گسترده باعث این اطمینان خاطر می‌شود که اگر یک نفر از پروژه جدا شود اطلاعات مهم از بین نخواهد رفت و افراد جدید می‌توانند بعد از ملحق شدن به پروژه به سرعت به روز شوند.

از آنجایی که این فرآیند از بالا به پایین کامل می‌شود، فرآیند توسعه نرم‌افزار **مدل آبشاری**^{۲۵} یا

چرخه حیات توسعه نرم‌افزار مدل آبشاری نام گرفته است. واضح و قابل درک است که با توجه به پیچیدگی هر مرحله از چرخه حیات مدل آبشاری، هر انتشار محصول یک واقعه بزرگ است که مهندسان برایش عرق ریخته‌اند و با هیاهوی زیاد همراه است. در چرخه 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 آتی 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 درباره آن چیزی که می‌خواهد ثابت نیست، به مشکل برمی‌خورد. این نقل قول از یک برنده جایزه تورینگ^{۲۶} به خوبی بیانگر این نگرش است:

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

— فرد بروکس جونیور^{۲۷}

به عبارت دیگر برای مشتری درک اینکه واقعاً چه می‌خواهد، زمانی که نسخه 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 چرخه حیات توسعه نرم‌افزار جدید در دهه ۱۹۸۰ انجامید که تهیه پیش‌نمونه‌ها^{۲۸} را با مدل آبشاری ترکیب می‌کرد (بوهم ۱۹۸۶). ایده این روش تکرار متناوب چهار فاز است که هر تکرار منجر به پیش‌نمونه‌ای خواهد شد که نسخه اصلاح‌شده تکرار پیشین است.

ویندوز ۹۵ با یک جشن ۳۰۰ میلیون دلاری به صورت عمومی معرفی شد. برای این برنامه مایکروسافت کمدين معروف جی لنو را استخدام و ساختمان امپایر استیت نیویورک را با رنگ‌های لوگوی ویندوز نورپردازی کرد. مجوز استفاده از آهنگ «من راه بیندار» گوه موسیقی رولینگ استونز نیز برای استفاده به عنوان موسیقی اصلی جشن خریداری شد.

²²Integration

²³Maintenance

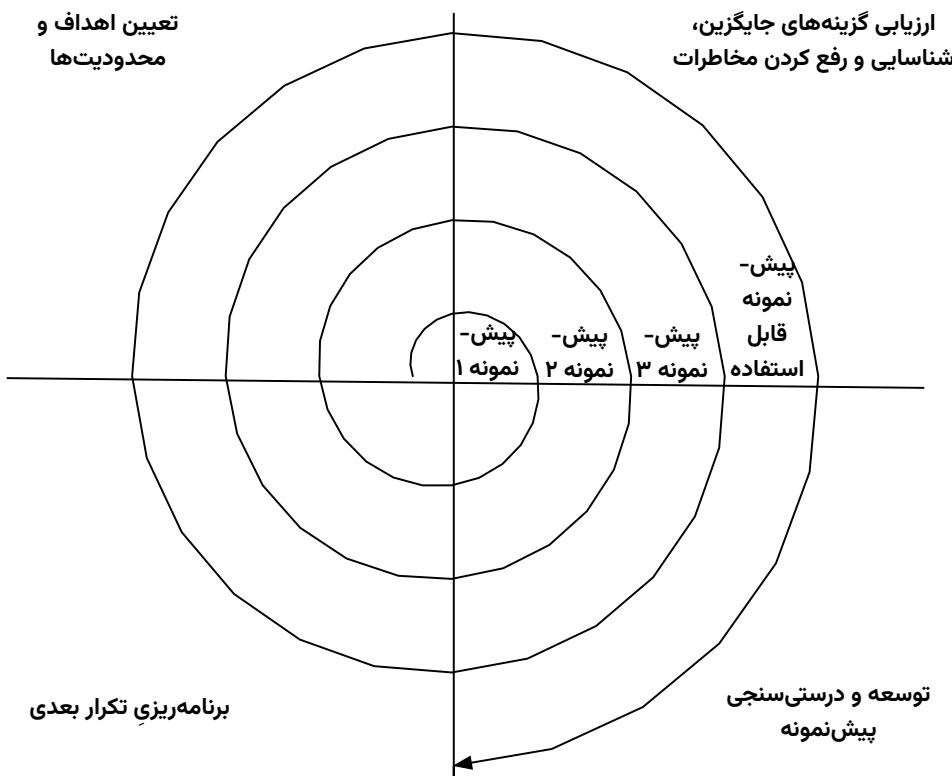
²⁴Documentation

²⁵Waterfall Model

²⁶Turing Award

²⁷Fred Brooks, Jr.

²⁸Prototype



شکل ۱-۲: چرخه حیات مدل مارپیچی، چرخه حیات مدل آبشاری را با تهیی 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.

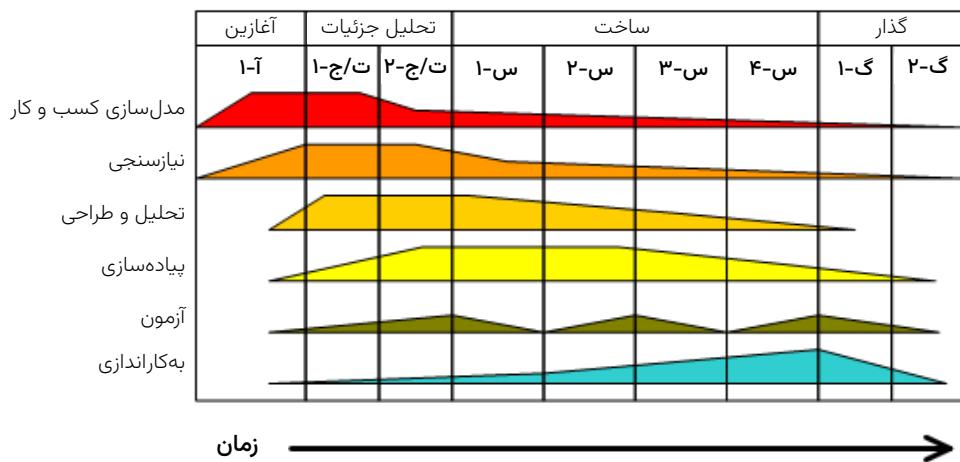
شکل ۱-۲ چهار فاز این مدل از توسعه نرم‌افزار را نمایش می‌دهد که در آن علت نام‌گذاری این روش به **مدل مارپیچی^{۲۹}** نیز مشخص می‌شود. این فازها عبارت‌اند از:

- ۱- تعیین اهداف و محدودیت‌های این تکرار^{۳۰}
- 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!

²⁹Spiral Model

³⁰Iteration



شکل ۱-۳: چرخه حیات فرایند یکپارچه 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 «ضابطه مکمل» هم دارد که در این شکل نشان داده نشده‌اند: مدیریت پیکربندی و تغییرات، مدیریت پروژه و محیط. (عکس از وب‌بانار توسط Dutchgilder).³¹

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 توسعه نرم‌افزار، علاوه بر دو روش ذکر شده فوق، انواع متعدد دیگری از متداول‌ترین طرح-و-ثبت پیشنهاد شده‌اند. یک مورد جدید، **فرایند یکپارچه رشنال^{۳۲}** (RUP^{۳۳}) (کراچن^{۳۴} ۲۰۰۳) است که در دهه ۱۹۹۰ توسعه یافته و ویژگی‌های چرخه‌های حیات مدل آبشاری و مدل مارپیچی را در هم می‌آمیزد و همچنین برای تهییه مستندسازی و نمودارها استانداردهایی را در بر می‌گیرد. ما RUP را به عنوان نماینده 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^{۳۴}: 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^{۳۵}: همکاری با ذی‌نفعان^{۳۶} به منظور تعیین موارد استفاده^{۳۷} طراحی معماري نرم‌افزار، تعیین برنامه توسعه و در نهایت ساخت پیش‌نمونه اولیه.

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

³¹Rational Unified Process

³²Inception (RUP phase)

³³Elaboration (RUP phase)

³⁴Stakeholder

³⁵Use case

- ساخت^{۳۶}: نوشتن و پیاده‌سازی محصول و همچنین مورد آزمون^{۳۷} قرار دادن آن که در نهایت به انتشار اولین نسخه خارجی می‌انجامد.

- گذار^{۳۸}: محصول را از محیط توسعه به محیط واقعی منتقل می‌کند، که شامل آزمون پذیرش^{۳۹} توسط مشتری و همچنین آموزش کاربران می‌شود.

با خلاف 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^{۴۰} شش «ضابطهٔ مهندسی» (که گردش کار^{۴۱} هم نامیده می‌شوند) را تعریف می‌کند که افرادی که روی پروژه کار می‌کنند می‌بایست آنها را به طور جمعی بتوشند:

- ۱- مدل‌سازی کسب و کار^{۴۲}
- ۲- نیازسنجی
- ۳- تحلیل و طراحی
- ۴- پیاده‌سازی
- ۵- آزمون
- ۶- به کاراندازی^{۴۳}

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 کسب و کار)، در حالی که برخی ضوابط متأوابا در طول پروژه (مانند آزمون) و برخی دیگر نزدیک به پایان پروژه (به کاراندازی) 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. متسافانه 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.

³⁶Construction (RUP phase)

³⁷Testing

³⁸Transition (RUP phase)

³⁹Acceptance Testing

⁴⁰Rational Unified Process

⁴¹Workflow

⁴²Business Modeling

⁴³Deployment (Software)

چکیده: در همهٔ فرآیندها یا چرخه‌های حیات توسعهٔ نرم‌افزار، **فعالیت‌های اساسی** مهندسی نرم‌افزار مشابه هستند، اما تعامل آن‌ها در طول زمان نسبت به انتشار محصول در بین مدل‌های مختلف متفاوت است. مشخصهٔ چرخهٔ حیات مدل آبشاری این است که بسیاری از طرح‌ریزی‌ها قبل از شروع برنامه‌نویسی انجام می‌شود، و اینکه هر فاز تتها پس از اتمام فاز قبل شروع می‌شود. چرخهٔ حیات مدل مارپیچی برای ساخت پیش‌نمونه‌ها مرتب‌آمدهٔ فازهای توسعه را تکرار می‌کند، اما مانند مدل آبشاری مشتریان تنها هر ۶ تا ۲۴ ماه یکبار با پروژه دستگیر می‌شوند. چرخهٔ حیات فرایند یکپارچه 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 کامل تکیه دارند و همگی پیشرفت پروژه را بر اساس طرح و برنامه 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 کنند!)

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

- ۱- ابتدایی یا پر هرج و مرچ: توسعهٔ نرم‌افزار بدون مستندسازی، بی‌ثبات و بدون اصول و موردي.
- ۲- تکراری‌ذیر: بدون دنبال کردن یک انضباط سفت و سخت اما بعضی فرآیندها قابل تکرار با نتایج ثابت هستند.
- ۳- تعریف‌شده: فرآیندهای 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 فرآیند کنترل کند، و فرآیند مورد استفاده را با پروژه‌های مختلف با موفقیت تطبیق دهد.
- ۵- بهینه‌سازی: اعمال بهینه‌سازی‌های کمی و سنجیده بر فرآیندها به عنوان بخشی از فرآیند مدیریت.

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

خودآزمایی ۱-۲-۱. یک وجه تشابه و یک تفاوت اصلی فرآیندهایی مانند مدل مارپیچی و RUP

نسبت به مدل آبشاری چیست؟

◆ همگی بر برنامه‌ریزی، طراحی و مستندسازی تکیه دارند اما فرآیندهای 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. ■

خودآزمایی ۱-۲-۲. تفاوت‌های میان فازهای سه فرآیند طرح-و-ثبت بحث شده کدام‌اند؟

◆ فازهای مدل آبشاری برنامه‌ریزی (نیازمندی‌ها و طراحی معماری) را از پیاده‌سازی جدا 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 مارپیچی، یک تکرار را مورد هدف قرار می‌دهند: تعیین اهداف تکرار، بررسی همهٔ راحلهای، توسعه و درستی‌سنجی 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

در بیان هر قسمت، یک یا چند سوال 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 سوالات بدھید.

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 به کارآمدانزی می‌کند. ■

۱-۳ فرآیندهای توسعه نرم افزار: منشور چابک

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 پرس^{۴۴}

پروژه شماره ۵۱۰ موشك

آریان^۵. در چهارم ژوئن ۱۹۹۶^{۳۷} تاریخ پس از برخاستن، زمانی که یک عدد ممیز شناور به یک عدد صحیح کوتاه‌تر تبدیل می‌شود، در سامانه هدایت موشك سریزی اتفاق افتاد که منجر به انفجار در حین برخاستگی شد. این استثنای خطای غیرمنتظره در موشك آریان^۴ که سرعت کمتری داشت، اتفاق نمی‌افتد. همانطور که این حادثه نشان می‌دهد، استفاده مجدد از اجزای نرم‌افزار بدون آزمون کامل و باقتضای سامانه می‌تواند گران تمام شود: ماهواره‌های به ارزش ۳۷۰ میلیون دلار از دست رفته‌اند.

هنوز هم پروژه‌های نرم‌افزاری وجود داشتند که به شکل افتضاحی شکست می‌خوردند و در بدنامی به سر بردن. برنامه‌نویسان^{۴۵} آنقدر داستان‌های تاسف‌بار انفجار **موشك آریان ۵**^{۴۶}، بیش‌مصرفی کشنده ناشی از تشبع‌شاعت ماشین پرتو درمانی تراک-۲۵^{۴۷}، فروپاشی مدارگرد اقلیمی مریخ^{۴۸}، و نیمه‌کاره رها شدن پروژه **نرم‌افزار پرونده مجازی افبی‌آی**^{۴۹} 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، در حالی که اکثر آن‌ها نیمه‌کاره رها شده بودند (چارت ۲۰۰۵).

شکل ۱-۱ چهار مطالعه بر روی پروژه‌های نرم‌افزاری را خلاصه می‌کند. تنها ۱۶٪ از پروژه‌ها طبق بودجه و زمان‌بندی تعیین‌شده پیش‌رفتند و مابقی که در اکثریت قرار دارند، متوقف یا نیمه‌کاره رها شده‌اند. با نگاهی دقیق‌تر به ۱۳٪ از پروژه‌های موقوفیت‌آمیز در مطالعه (ب) همه چیز حتی نگران‌کننده‌تر به نظر می‌رسد، زیرا کمتر از ۱٪ از پروژه‌های جدید طبق زمان‌بندی و بودجه تعیین‌شده 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 نیمه‌کاره رها شدند، و ۵۰٪ از آن‌ها با تأخیر، با صرف هزینه بیش از بودجه تعیین‌شده و بدون برخی از قابلیت‌های لازم به سرانجام رسیده بودند. اگر تاریخ را ملاک قرار دهیم، شانس آقای اوباما برای آغاز به کار موفق HealthCare.gov نتها یک به ده بود.

روش چابک با عنایوین دیگری

مانند فرآیند سبک‌وزن یا بی‌قاعده نیز شناخته می‌شود.

من توان گفت که لحظه «اصلاحات» در مهندسی نرم‌افزار، انتشار **منشور چابک** در فوریه ۲۰۰۱ بود. گروهی از توسعه‌دهنگان نرم‌افزار برای توسعه یک چرخهٔ حیات نرم‌افزاری سبک‌وزن‌تر گرد هم آمدند. متن زیر دقیقاً همان اعلامیه‌ای است که اتحادیه روش چابک^{۵۰} به در «کلیساي طرح-و-ثبت» میخود:

«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 تعاملات بین آن‌ها به جای فرآیندها و ابزارها

• نرم‌افزاری که کار می‌کند به جای مستندسازی جامع

• همکاری با مشتری به جای چانه‌زنی در قرارداد

⁴⁴Shimon Peres

⁴⁵Programmer

⁴⁶Ariane 5 Rocket

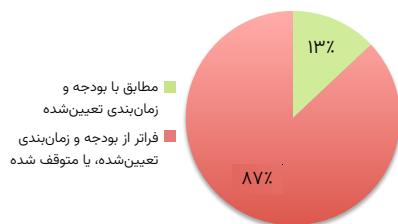
⁴⁷Therac-25 (radiation therapy machine)

⁴⁸Mars Climate Orbiter

⁴⁹Virtual Case File (FBI Software)

⁵⁰Agile Alliance

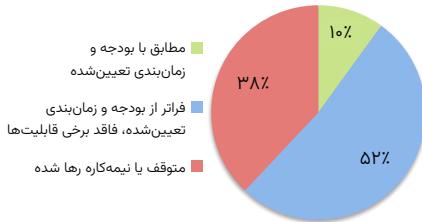
(ب) پروژه‌های نرم‌افزاری (تیلور ۲۰۰۰)



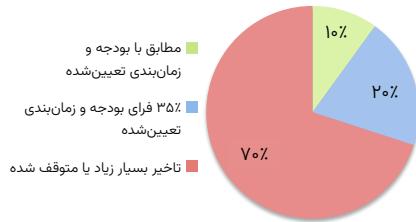
(الف) پروژه‌های نرم‌افزاری (جانسون ۱۹۹۵)



(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 تعیین‌شده‌شان تا سه برابر فراتر رفته‌اند و ۳۱٪ دیگر پروژه‌های نرم‌افزاری قبل از تکمیل، لغو و رها می‌شوند (جانسون ۱۹۹۵). در آمریکا هزینه این پروژه‌ها ۱۰۰ میلیارد دلار تخمین زده شده است. (ب) مطالعه دیگر توسط انجمن رایانه بریتانیا در سال ۲۰۰۰ نشان داد که فقط ۱۳٪ عدد از ۱۰۲۷ پروژه طبق بودجه و زمان‌بندی شان پیش رفتند. نیمی از پروژه‌ها مربوط به نگهداری و تبدیل اطلاعات گذشته بوده و نیمه دیگر مربوط به پروژه‌های ساخت سامانه‌های جدید بوده است، اما از ۱۳٪ پروژه موفق ۱۱٪ عدد از نوع اول و فقط سه عدد از نوع دوم بودند (تیلور ۲۰۰۰). (پ) مطالعه دیگری بر روی ۲۵۰ پروژه بزرگ، هر کدام با حجمی معادل بیش از یک میلیون خط به زبان سی، نیز نتایج نامیدکننده مشاهده داشت (جونز ۲۰۰۴). (ت) نتایج ملات انکیز پروژه‌های «بزرگ» (دست کم ۱۰ میلیون دلار) که در این مطالعه (جانسون ۲۰۱۳^b) بر روی ۵۰,۰۰۰ پروژه به دست آمده نشان می‌دهد که HealthCare.gov تنها ۱٪ شناسنامه موافقیت داشته است. (ث) و اما خیر خوب اینکه پروژه‌های کوچک (با هزینه کمتر از ۱ میلیون دلار) در همان مطالعه تا حد زیادی به موقع و در محدوده بودجه تعیین‌شده تکمیل شده‌اند، که انگیزه استفاده از روش‌های چاپک را پیشتر می‌کند.

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.»

گونه‌های مختلف روش چابک.

اساس این مدل دیگر توسعه نرم‌افزار، پذیرفتن تغییرات به عنوان یک واقعیت زندگی است: توسعه‌دهندگان باید به طور مستمر پیش‌نمونه‌ای را که ناقص است اما کار می‌کند، بهبود بخشنند تا زمانی که مشتری از نتیجه آن راضی شود، در حالی که مشتری برای هر تکرار بازخورد می‌دهد. مدل چابک برای کاهش اشتباوهای بر توسعه آزمون محور^{۵۱} (TDD) تاکید دارد که در آن قبل از شروع به نوشتمن برنامه، آزمون‌ها^{۵۲} نوشته می‌شوند. این مدل همچنین بر استفاده از روابط‌های کاربری^{۵۳} برای رسیدن به توافق و تایید نیازمندی‌های مشتری، و معیار سرعت^{۵۴} برای سنجش پیشرفت پرروژه تاکید دارد. همه این موارد را در فصول بعدی با جزئیات برسی خواهیم کرد.

کنت پک منسوب است.

از نظر طول عمر نرم‌افزاری، چرخه حیات نرم‌افزاری چابک‌تر صریح است که نسخه‌های جدید هر هفته یا دو هفته یک بار ارائه می‌شوند (حتی ممکن است برعی هر روز منتشر شوند). بنابراین بر خلاف روند مدل‌های طرح-و-ثبت، انتشار نسخه‌های جدید اتفاق ویژه‌ای محسوب نمی‌شود. در این روش فرض بر بهبود مستمر در تمام طول عمر نرم‌افزار است.

در قسمت قبل اشاره کردیم که ممکن است فرآیندهای طرح-و-ثبت برای تازهواردها خسته‌کننده و طاقت‌فرسا باشد، اما در مورد روش‌های چابک چنین نیست. این دیدگاه در یکی از اولین نقدهای روش چابک توسط یک مدرس مهندسی نرم‌افزار به خوبی دیده می‌شود:

زمانی که برنامه‌نویسی لذت‌بخش بود را به خاطر می‌آورید؟ آیا به همین دلیل بود که به رایانه و بعد هم به علوم رایانه علاقه‌مند شدید؟ آیا دانشجویان کارشناسی ارشد ما به همین علت به این رشته می‌آیند—جون دوست دارند برای رایانه برنامه‌نویسی کنند؟ خب، ممکن است روش‌های توسعه نرم‌افزار قابل احترام و امیدوارکننده‌ای وجود داشته باشند که برای این دسته از افراد ایده‌آل هستند. ...[روش چابک] لذت‌بخش و موثر است، چون در آن نه تنها فرآیند را در بالاترین اینبوه مستندسازی گیر نمی‌اندازیم، بلکه توسعه‌دهندگان در طول فرآیند توسعه با مشتریان به صورت رو در رو کار می‌کنند و از همان اوایل شروع کار نرم‌افزاری که کار می‌کند را تولید می‌کنند.

—رنی مک کاولی^{۵۵}، «روش‌های توسعه چابک، آماده برای بر هم زدن وضع موجود»، *SIGCSE Bulletin* ۲۰۰۱.

منشور چابک با تاکیدزدایی و کاستن از اهمیت برنامه‌ریزی، طراحی، مستندسازی و مشخصات الزام‌آور قراردادی، در تضاد با خرد متعارف و نظر بزرگان حوزه مهندسی نرم‌افزار حرکت کرد و به همین خاطر با آگوش باز از آن استقبال نشد (کورمیک ۲۰۰۱):

[منشور چابک] تلاش دیگری برای کم‌ارزش جلوه دادن رشته مهندسی نرم‌افزار است... در حرفة مهندسی نرم‌افزار دو گروه وجود دارند، مهندسان و هکرها^{۵۶}... به نظر من این چیزی بیشتر از تلاش برای مشروعیت بخشیدن به رفتارهای هکری نیست... حرفة مهندسی نرم‌افزار تنها زمانی بهبود می‌یابد که مشتریان از پرداخت پول برای نرم‌افزاری که طبق مفاد قرارداد کار نمی‌کند، سر باز زند... تغییر فرهنگ از فرهنگی که ذهنیت هکری را تشویق می‌کند به فرهنگی که بر روش‌های قابل پیش‌بینی مهندسی نرم‌افزار استوار شده تنها راه تبدیل مهندسی نرم‌افزار به یک رشتۀ آبرومند مهندسی است.

—استیون راتکین^{۵۷}، «منشوری که بدینی برمی‌انگیزد»، *IEEE Computer* ۲۰۰۱.

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

⁵¹ Test-Driven Development

⁵² Test

⁵³ User Story

⁵⁴ Velocity (Software Development Metric)

⁵⁵ Renee McCauley

⁵⁶ Hacker

⁵⁷ 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 پیچیده 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 قرهنگ 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 چرخه حیات چاپک (اگر پاسخ منفی باشد) یا یک چرخه 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 دانشجویی انجام می‌شوند مطرح می‌کیم، تقریباً همه پاسخها به سمت روش چاپک است. چنان که این کتاب هم نشان می‌دهد، نرم‌افزارهای مترباز ابزارهای عالی هستند که در اختیار دانشجویان قرار دارند (پرسش ۶). پرسش‌های ما از صنعت (رجوع شود به پیش‌گفتار) نشان داد که دانشجویان فارغ‌التحصیل واقعاً مهارت‌های برنامه‌نویسی خوبی دارند (پرسش ۹). پاسخ سایر هشت پرسش بهطور واضح برای پروژه‌های دانشجویی منفی است.

«خطوط نبرد ترسیم شده‌اند. خصومت‌ها بین اردوگاه‌های مسلح جامعه توسعه نرم‌افزار شدت گرفته است. این بار بانگ صفا‌آرایی این است: «XP! ... چیزی که XP (دوباره) کشف کرد همان دعواهی قدیمی بین جامعه نرم‌افزاری است: برنامه‌نویسی در مقابل مهندسی نرم‌افزار (یا به عبارتی هکرهای ژولیده در مقابل دانشمندان وارسته رایانه)».

حقیقین حوزه مهندسی نرم‌افزار به مقایسه 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 متواند مفید باشد. شکل ۱-۵ ده پرسش 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 حالی که شکل ۱-۴(ت) نتایج نالمیدکننده پروژه‌های نرم‌افزاری بزرگ را نشان می‌دهد که از روش چاپک استفاده نمی‌کنند، شکل ۱-۴(ث) موفقیت 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 برنامه زمان‌بندی و بودجه و با پیاده‌سازی همه قابلیت‌ها پیش‌رفته‌اند، این نتایج با مابقی نمودارهای شکل ۱-۴ در تضاد کامل است. موفقیت 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 ایکانامیک 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 چنان هم آسان نیست (استرلر و دیگران ۲۰۱۲).

بنابراین ما در این کتاب، در شش فصل توسعه 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^{۵۸} و محیط‌های برنامه‌نویسی آن از جمله زبان‌های برنامه‌نویسی رویی و جاوا اسکریپت^{۵۹} و چارچوب رویی آن ریز می‌پردازد.

چرخه حیات چاپک در واقع خانواده‌ای از روش‌های مختلف است و نه یک روش واحد. ما

⁵⁸Software as a Service

⁵⁹JavaScript

برنامه‌نویسی مفطر^{۶۰} (XP) را دنبال خواهیم کرد که شامل این موارد است: تکرارهای یک تا دو هفتگه‌ای، توسعه رفتارمحور^{۶۱} (رجوع شود به فصل ۷)، توسعه آزمون محور (رجوع شود به فصل ۸)، و برنامه‌نویسی دونفره^{۶۲} (قسمت ۲-۲). **اسکرام^{۶۳}** یک گونه محبوب دیگر است (قسمت ۱۰-۱)، که در آن تیم‌های خودسازمانده^{۶۴} از تکرارهای دو تا چهار هفته‌ای که **اسپرینت^{۶۵}** خوانده می‌شوند، استفاده می‌کنند و سپس برای برنامه‌ریزی برای روزانه برای برای تشخیص و برطرف کردن موانع و مشکلات کلیدی این روش جلسه‌های سریاپی روزانه برای دوباره به دور هم جمع می‌شوند.^{۶۶} یک ویژگی 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^{۶۷}** است که نشئت گرفته از فرآیند تولید بهنگام^{۶۸} شرکت تویوتا^{۶۹} است که نگاهی همچون یک خط لوله^{۷۰} و یا خط تولید به توسعه نرم‌افزار دارد. در این روش نفرات تیم نقش‌های ثابت دارند و هدف این است که تعداد نفرات تیم طوری متوازن شود که هیچ گلوگاهی با کارهای روی‌هم‌انباشته و منتظر پردازش وجود نداشته باشد. یک ویژگی معمول این روش، دیواری از کارت‌ها برای نشان دادن وضعیت همه 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^{۷۱}** 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 همه 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، چرخه 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. به جای آن از **سرعت** برای اندازه‌گیری پیشرفت استفاده می‌شود که در واقع برابر با آهنگ تکمیل و اضافه شدن امکانات جدید به پروژه است.

⁶⁰Extreme Programming

⁶¹Behavior-Driven Development

⁶²Pair Programming

⁶³Scrum

⁶⁴Self-Organizing (Team Management)

⁶⁵Sprint

⁶⁶ما از لفظ «خودسازمانده» در ترجمه Self-Organizing استفاده کردیم که توصیف دقیق‌تری نسبت به لفظ متدائل‌تر «خودمختار» است. تیم‌های خودسازمانده در چگونگی انجام کار خودمختار هستند، اما در اینکه چه کاری انجام دهند این اختیار را ندارند. در مقابل تیم‌های خودمختار (Autonomous) معمولاً استقلال و اختیار کاملی دارند تا تصمیم بگیرند چه کاری را انجام بدهند.

⁶⁷Kanban

⁶⁸Just-in-time

⁶⁹Toyota (Company)

⁷⁰Pipeline

⁷¹ScrumBan

■ 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 دائم 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 قبل از ویگاه ACA، زمزمه‌ها و درخواست‌های 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 برخورد کرده که عبارت است از فرهنگ و مقرراتی که برنامه‌های بزرگ، نظارت سطح بالا و یک روش توسعه و آزمون سنجیده و ترتیبی (مدل آبشاری) را ترجیح می‌دهد. برنامه‌هایی که از آن‌ها انتظار ارائه راه حل‌های کامل و تقریباً بی‌نقص وجود دارد و توسعه آن‌ها سال‌ها به طول می‌انجامد 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 توسعه تاریخی محصول به پایین‌ترین سطح عملی واگذار می‌شود و تیم مدیریت برنامه این انعطاف‌پذیری را دارد که محتوای هر مرحله از توسعه تدریجی را تغییر دهد تا با زمان‌بندی تحويل کار هماهنگ پیش بروند ... رویکردهای چابک به پذیرندگان خود این امکان را داده است که از غول‌های صنعتی جاافتاده و شناخته شده که با ساختارهای مدیریتی پیچیده، فرآیندمحور و متعلق به دوران انقلاب صنعتی احاطه شده بودند، پیش بگیرند. رویکردهای چابک به این خاطر موفق شده‌اند که استفاده‌کنندگان از این روش‌ها مسائل و مشکلات موثر بر مخاطرات در حیطه برنامه‌های فناوری اطلاعات را تشخیص داده‌اند و ساختارها و فرآیندهای مدیریتی خود را برای کاهش این مخاطرات تغییر داده‌اند.» (پژوهش ملی آمریکا ۲۰۱۵)

خودآزمایی ۱-۳-۱. درست یا نادرست: یک تفاوت بزرگ بین توسعه نرم‌افزار مدل مارپیچی و چابک ساخت پیش‌نمونه‌ها و تعامل با مشتریان در حین انجام فرآیند 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 در حالی که در مدل مارپیچی ممکن است حتی تا دو سال این اتفاق نیفتد. ■

خودآزمایی ۱-۳-۲. درست یا نادرست: یک تفاوت بزرگ بین مدل آبشاری و روش چابک این است که در روش چابک از جمع‌آوری نیازمندی‌ها استفاده نمی‌شود.

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

۱-۴ تضمین کیفیت نرم‌افزار: آزمون

و کاربران با 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

نرم افزار، کیفیت هم به معنی برآورده کردن نیازهای مشتری (راحتی در استفاده، دریافت چواب‌های صحیح، عدم خروج ناگهانی و غیره) و همچنین ساده و راحت بودن بهبود و اشکال زدایی^{۷۲} آن برای توسعه‌دهندگان است. ریشه **تضمین کیفیت** (QA)^{۷۳} نیز به صنعت تولید برمی‌گردد، و به فرآیندهای معیارهایی که منجر به تولید محصولات با کیفیت بالا می‌شوند، و همچنین به ایجاد فرآیندهای تولیدی که کیفیت را بالا می‌برند، اشاره دارد. پس تضمین کیفیت نرم افزار، به معنی حصول اطمینان از کیفیت بالای محصول درحال تولید و همچنین ایجاد فرآیندها و استانداردهایی در یک سازمان است که به تولید محصولات با کیفیت بالا منجر می‌شود. همانطور که در ادامه خواهیم دید، در برخی از فرآیندهای نرم افزاری طرح-و-ثبت حتی از یک تیم جداگانه QA برای بررسی و آزمودن کیفیت نرم افزار استفاده می‌کنند (قسمت ۱۰-۸).

فرآیند تعیین کیفیت نرم افزار شامل دو اصطلاح می‌شود که معمولاً به جای یکدیگر به کار می‌روند اما تفاوت ظریفی دارند (بوهم ۱۹۷۹):

۰ درستی‌سنجد^{۷۴}: آیا محصول را درست ساخته‌اید؟ (آیا محصول، طبق مشخصات^{۷۵} از قبیل تعریف شده ساخته شده؟)

۰ اعتبارسنجد^{۷۶}: آیا محصول درست را ساخته‌اید؟ (آیا این همان چیزی است که مشتری می‌خواهد؟ یا به عبارت دیگر، آیا مشخصات تعریف شده با خواسته مشتری تطابق دارد؟)

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

طریقه اصلی درستی‌سنجد و اعتبارسنجد، **آزمون** است. انگیزه اصلی انجام آزمون یافتن هرچه زودتر اشتباها توسط توسعه‌دهندگان است تا هزینه 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 توسعه نرم افزار است. به ترتیب از سطح پایین به بالا، ابتدا **آزمون واحد**^{۷۷} قرار می‌گیرد. این نوع آزمون به ما این اطمینان را می‌دهد که یک زویه^{۷۸} یا تابع^{۷۹} واحد، کاری را انجام می‌دهد که از آن انتظار می‌رود. سطح بعد **آزمون مازول**^{۸۰} است که تک‌تک واحدها را به صورت سراسری مورد آزمون قرار می‌دهد. به طور مثال، آزمون واحد در محدوده یک کلاس^{۸۱} واحد کار می‌کند، اما آزمون مازول تمام کلاس‌ها را بررسی می‌کند. بر روی این سطح، **آزمون یکپارچگی**^{۸۲} قرار دارد، که تضمین می‌کند واسطه‌های بین واحدها دارای فرضیات ثابت و سازگار با یکدیگر هستند و آن‌ها به درستی باهم ارتباط پرقرار می‌کنند. این سطح، به بررسی عملکرد واحدها نمی‌پردازد. در بالاترین سطح، **آزمون سامانه**^{۸۳} یا **آزمون پذیرش** قرار دارد که وظیفه 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^{۸۴} نام دارد.

⁷²Debugging

⁷³Quality Assurance (QA)

⁷⁴Verification

⁷⁵Specification

⁷⁶Validation

⁷⁷Unit Testing

⁷⁸Procedure (Programming)

⁷⁹Function (Programming)

⁸⁰Module Testing

⁸¹Class (Computer Programming)

⁸²Integration Testing

⁸³System Testing

⁸⁴Formal Methods

غیر عملی بودن آزمون جامع.
فرض کنید که آزمودن یک برنامه یک ناوتانیه طول می‌کشد و این برنامه فقط یک ورودی^{۷۴} دارد که ما می‌خواهیم آن را به طور کامل و جامع مورد آزمون قرار دهیم. (دیهی است که اکثر برنامه‌ها زمان بیشتری برای اجرا نیاز دارند و ورودی‌های آن‌ها نیز بیشتر است). فقط همین مورد ساده^{۷۵} ناوتانیه یا ۵۰۰ سال طول خواهد کشید.

همانطور که به طور خلاصه در قسمت ۳-۱ ذکر شد، انجام آزمون در گونه XP^{۸۵} 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 فلسفه 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 توسعه رفتارمحور (BDD^{۸۶}) استفاده می‌شود، که موضوع فصل ۷ این کتاب است. این روش همچنین برای آزمون واحد و آزمون ماثول از توسعه آزمون محور (TDD^{۸۷}) 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 درستی‌سننجی یک نرم‌افزار در داشتن مشخصات تعیین‌شده و همچنین اعتبارسننجی طراحی آن مطابق با خواسته‌های مشتری کمک می‌کند.
- با تقسیم آزمون به آزمون واحد، آزمون ماثول، آزمون یکپارچگی و آزمون سامانه یا آزمون پذیرش در جهت غالبه بر غیرعملی بودن انجام یک آزمون جامع تلاش می‌شود. هر آزمون سطح بالاتر، انجام آزمون‌هایی دقیق‌تر و جزئی‌تر را به سطح پایین‌تر واگذار می‌کند.
- روش توسعه چاپک با استفاده از نوشتمن آزمون‌ها پیش از نوشتمن کد و انجام توسعه رفتارمحور و توسعه آزمون محور بسته به سطح آزمون، بر سختی‌های آزمون غالبه می‌کند.

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

در فرآیند توسعه مدل آبشاری، انجام آزمون پس از پایان هر فاز و همچنین در یک فاز پایانی درستی‌سننجی 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 گونه XP از روش چاپک، ضمانت از طریق توسعه آزمون محور انجام می‌گیرد که در آن آزمون‌ها قبل از نوشتمن کد نوشتمنه می‌شوند. این دوین 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. مسلمًا یک میراث کد به آزمون بیشتری نیاز دارد.

خودآزمایی ۱-۴. می‌دانیم که تمامی روش‌های آزمون زیر در درستی‌سننجی به ما کمک می‌کنند؛ اما کدام مورد بیش از دیگران به اعتبارسننجی کمک می‌کند: آزمون واحد، آزمون ماثول، آزمون یکپارچگی و یا آزمون پذیرش؟

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

⁸⁵Extreme Programming

⁸⁶Behavior-Driven Development

⁸⁷Test-Driven Development

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

بر اساس قانون مور^{۸۸} 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 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 بهبود بهره‌وری خود ایجاد کردند:

۱-وضوح به کمک اختصار

۲-سنتز

۳-بازکاربرد

۴-خودکارسازی به کمک ابزارها

وضوح^{۸۹} به کمک اختصار^{۹۰} معنکس کننده یکی از پیش‌فرض‌های محرک و اصلی بهبود بهره‌وری برنامه‌نویسان است: هر چه کد برنامه راحت‌تر فهمیده شود و خواناتر باشد، إشکالات 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 طبیعی و با حروف یا نویسه‌هایی کمتری بنویسد. برای مثال هر دو خط زیر یک مفهوم باشند^{۹۱} ساده را نشان می‌دهند:

- 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 سردگمی 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 پیشگویی‌ها نگهداری آن را نیز آسان‌تر می‌کند.

روش دیگر برای افزایش وضوح، بالا بردن سطح تجزیه^{۹۲} است. این روش در ابتدا به تولید زبان‌های برنامه‌نویسی سطح بالاتری مانند فورترن^{۹۳} و کوبال^{۹۴} منجر شد. ظهور این زبان‌ها باعث ارتقای سطح تولید و مهندسی نرم‌افزار از زبان آسمبلی^{۹۵} برای یک رایانه خاص، به زبان‌های سطح بالایی شد، که می‌توانستند تنها با عوض کردن کامپایلر^{۹۶} بر روی رایانه‌های مختلف اجرا شوند.

با ادامه روند رو به رشد عملکرد سخت‌افزار، تعداد بیشتری از برنامه‌نویسان مشتاق بودند تا کارهایی را که قبل از خودشان انجام می‌دادند به کامپایلرها و سامانه‌های زمان اجرا^{۹۷} واگذار کنند.

⁸⁸Moore's Law

⁸⁹Clarity

⁹⁰Conciseness

⁹¹Assertion (Computer Programming)

⁹²Abstraction (Computer Science)

⁹³Fortran (Programming Language)

⁹⁴COBOL (Programming Language)

⁹⁵Assembly Language

⁹⁶Compiler

⁹⁷Runtime System

به طور مثال جاوا^{۹۸} و زبان‌های برنامه‌نویسی مشابه دیگری، مدیریت حافظه را خود به دست گرفتند؛ و بیزگی‌ای که در زبان‌های قدیمی‌تری مانند سی^{۹۹} و سی‌پلاس‌پلاس^{۱۰۰} بر عهده برنامه‌نویس گذاشته شده‌بود. زبان‌های اسکریپت‌نویسی^{۱۰۱} مانند پایتون^{۱۰۲} و روبی سطح تجرید را از این هم بالاتر بردند. از نمونه‌هایی از این سطح تجرید بالاتر می‌توان به مواردی مانند بازکاربرد^{۱۰۳}، که به برنامه اجازه می‌دهد تا خودش را تحت نظر بگیرد و یا **توابع مرتبه بالاتر**^{۱۰۴} نام برد، که امکان بازکاربرد رفتارهای سطح بالاتر را با ارسال توابع به عنوان آرگومان^{۱۰۵} به توابع دیگر فراهم می‌کند. این سطح تجرید 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^{۱۰۶} به دسته راهکارهایی برای افزایش بهره‌وری اشاره دارد که در آن‌ها کد به جای 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**^{۱۰۷} است که یک دستور گرافیکی برای ترکیب دو بیت‌پ^{۱۰۸} 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 خاص منظورهای 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 چرخدنده‌های «تولید کد» استفاده می‌کنیم. چارچوب روبی آن‌ریلز از امکانات زبان برنامه‌نویسی روبی برای **فرابرنامه‌نویسی**^{۱۰۹} به صورت گسترده استفاده می‌کند، که این امکان را به برنامه‌های روبی می‌دهد تا به طور خودکار در زمان اجرای کد سنتر کنند.



بازکاربرد^{۱۱۰} و یا استفاده مجدد از قسمت‌هایی که پیش‌تر طراحی شده‌اند به جای نوشتن همه‌چیز از صفر، سومین راهکار افزایش بهره‌وری است. از آنجایی که اعمال تغییرات کوچک در نرم‌افزار آسان‌تر از سخت‌افزار است، امکان استفاده مجدد از قسمت‌های 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 استفاده مجدد کنند. به دنبال آن، کتابخانه‌های استانداردی برای ورودی/خروجی^{۱۱۱} و توابع ریاضی به وجود آمدند تا برنامه‌نویسان بتوانند کدهایی را که افراد دیگر توسعه داده‌اند، مورد بازکاربرد قرار دهند.

رویه‌هایی که در کتابخانه‌ها^{۱۱۲} وجود دارند، امکان استفاده مجدد از پیاده‌سازی کارهای مشخصی را به وجود می‌آورند. ولی بیشتر برنامه‌نویسان می‌خواهند مجموعه‌ای **گرداوری** شده از کارها را مدیریت و مورد استفاده مجدد قرار دهند. از این رو، قدم بعدی در بازکاربرد پذیری نرم‌افزار، **برنامه‌نویسی**

⁹⁸Java (Programming Language)

⁹⁹C (Programming Language)

¹⁰⁰C++ (Programming Language)

¹⁰¹Scripting Language

¹⁰²Python (Programming Language)

¹⁰³Reflection (Computer Programming)

¹⁰⁴Higher-order Function

¹⁰⁵Argument (Programming)

¹⁰⁶Synthesis

¹⁰⁷Bit Blit

¹⁰⁸Bitmap

¹⁰⁹Metaprogramming

¹¹⁰Reuse

¹¹¹Input/Output (I/O)

¹¹²Library (Computing)

شی‌عگرا^{۱۱۳} بود. با استفاده از امکان ارث‌بری^{۱۱۴} 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 ارث‌بری امکان استفاده مجدد از پیاده‌سازی‌ها را فراهم می‌کرد، وجود یک روش و یا الگوی کلی برای انجام کارهایی که حتی پیاده‌سازی‌شان متفاوت بود نیز فرستی مناسب برای بازکاربرد ایجاد می‌کرد. اینجا بود که **الگوهای طراحی**^{۱۱۵}، 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**^{۱۱۶} یکی از این امکانات بود که ترکیب و درهم آمیختن تجربیدها را آسان می‌ساخت. **میکس‌این‌ها**^{۱۱۷} دیگر امکانی بودند که استفاده از کارکرد چندین میل^{۱۱۸} را بدون نیاز به ارث‌بری چندگانه و دردرس‌هایش، فراهم می‌ساختند. پایتون و روبي از نمونه زبان‌هایی هستند که امکانات را برای کمک به 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 همه 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^{۱۱۹} و ديو توماس^{۱۲۰}، ۱۹۹۹—



این رهنمود در این شعار خلاصه شده است: **خودت را تکرار مکن**^{۱۲۱}. در این کتاب ما از لفظ DRY و نماد حوله^{۱۲۲} 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 مرتبه بالاتر و سایر مکانیسم‌های مختلف برای افزایش استفاده مجدد از کد بهره می‌برند. روبي با دربرگرفتن پیشرفت‌های مهم در زبان‌های برنامه‌نویسی پا را فراتر از زبان‌هایی مانند پل^{۱۲۳} در پشتیبانی از پارادایم‌های برنامه‌نویسی^{۱۲۴} متعدد مانند برنامه‌نویسی شی‌عگرا و برنامه‌نویسی تابعی^{۱۲۵} می‌گذارد.

خودکارسازی^{۱۲۶}، 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 توسعه نرم‌افزار، کامپایلرها و مفسرها^{۱۲۷} هستند که همانطور که پیش‌تر اشاره شد، سطح تجريد را بالا می‌برند و کد نیز تولید می‌کنند. اما ابزارهای

۱۱۳ Object-oriented (Programming)

۱۱۴ Inheritance (OOP)

۱۱۵ Design Pattern

۱۱۶ Dynamic Typing (Type Checking)

۱۱۷ Mix-in (OOP)

۱۱۸ Method (OOP)

۱۱۹ Andy Hunt

۱۲۰ Dave Thomas

۱۲۱ Don't Repeat Yourself (DRY)

۱۲۲ از آنجایی که کلمه DRY مخفف معادل انگلیسی عبارت «خودت را تکرار مکن» است و از نظر لغوی به معنای خشک می‌باشد، نماد حوله برای آن انتخاب شده است.

۱۲۳ Perl (Programming Language)

۱۲۴ Programming Paradigm

۱۲۵ Functional Programming

۱۲۶ Automation

۱۲۷ Interpreter

فصل ۱. مقدمه‌ای بر توسعه چاپک و SAAS

بهره‌وری ظرفیتی مانند میکفایل‌ها^{۱۲۸} و سامانه‌های کنترل نسخه^{۱۲۹} (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 پادگرفتن طریقه استفاده از یک ابزار جدید در مقابل میزان صرفه‌جویی در وقت در صورت استفاده از آن است. قابل اعتماد بودن ابزار، کیفیت تجربه کاربری 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 اورهای بینایی 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. به‌همین 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**^{۱۳۰} به صورت خودکار روایت‌های کاربری را به آزمون‌های پذیرش تبدیل می‌کند و همچنین اینکه چطور **پیووتال ترکر**^{۱۳۱} به طور خودکار «سرعت» را که مقیاسی برای نرخ افزوده شدن قابلیت‌ها به نرم‌افزار است، محاسبه می‌کند. در فصل ۸، به معرفی **آر-اسپ**^{۱۳۲} می‌پردازیم که به خودکارسازی فرآیند آزمون واحد کمک می‌کند. و اما خبر بد اینکه شما باید کار با چندین ابزار جدید را باد 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 پروژه 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.



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 **بازتاب** 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.

¹²⁸Makefile

¹²⁹Version Control

¹³⁰Cucumber (Software tool)

¹³¹Pivotal Tracker

¹³²RSpec (Testing tool)

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

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

خودآزمایی ۱-۵-۱. کدام راهکار ضعیف‌ترین استدلال برای مزایای بهره‌وری کامپایلرها برای زبان‌های برنامه‌نویسی سطح بالا است:وضوح به کم اختصار، سنتر، بازکاربرد، 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 سنتر 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. ■

۶-۱ نرم افزار به صورت یک سرویس و معماری سرویس‌گرا

در اواسط دهه ۱۹۹۰، با گسترش وب^{۱۳۳} و افزایش تعداد مخاطبان آن، ایدهٔ جدید شروع به ظهر کرد: به جای اتکا به کاربران برای نصب نرم افزار بر روی رایانه‌های خود، چرا نرم افزار را به صورت مرکزی بر روی سرویهایی مبتنی بر اینترنت اجرا نکیم و به کاربران امکان دسترسی به آن را از طریق مرورگرهای شان بدھیم؟ سیلزفوس^{۱۳۴} شاید اولین شرکت بزرگی بود که به طور کامل از این مدل جدید که **نرم افزار به صورت یک سرویس (SaaS)** نام گرفت، استقبال کرد. نمونه‌هایی از^{۱۳۵} SaaS که امروزه بسیاری از

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

۱- از آنجایی که کاربران نیازی به نصب نرم افزار کاربردی^{۱۳۶} ندارند، دیگر لازم نیست نگران مناسب و سریع بودن سخت‌افزارشان و یا مناسب بودن نسخهٔ سیستم عامل^{۱۳۷} دستگاه‌هایشان باشند.

۲- داده‌های مرتبط با سرویس معمولاً^{۱۳۸} 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 دادن دستگاه نظیر تلفن یا تبلت^{۱۳۹} را ندارند.

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

۱۳۳ Web (World Wide Web)

۱۳۴ Salesforce

۱۳۵ Software as a Service

۱۳۶ Search Engine

۱۳۷ Word Processor (Software)

۱۳۸ Application (Software)

۱۳۹ Operating System

۱۴۰ Tablet

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	پاچ بی	زند
۲۰۰۷	روبی	سیناترا

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

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^{۱۴۱} در یک محیط یکسان و کنترل شده سخت‌افزاری و نرم‌افزاری (سیستم عامل) که توسط توسعه‌دهنده انتخاب شده اجرا می‌شود. این امر باعث اجتناب از دردرس 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^{۱۴۲} (API) خارجی را نقض نکنند. علاوه بر این، توسعه‌دهندگان می‌توانند نسخه‌های جدید نرم‌افزار کاربردی را بر روی تعداد کمی از کاربران به طور 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 ساده 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 کردہ‌ایم که گروهی آن را ساخته‌اند که چرخهٔ حیات چاپک را به خوبی پذیرفته‌اند، بنابراین ابزارهای مرتبط با آن به خوبی از روش چاپک پشتیبانی می‌کنند. اگر از قبل با روبی و یا ریلز آشنایی ندارید، این فرصت خوبی است تا مهارتی مهم در مهندسی نرم‌افزار را تمرین 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 جامعهٔ کاربران ریلز این است که مشارکت‌کنندگان آن به طور معمول با ابداع ابزارهای جدید برای خودکارسازی کارهایی که قبلاً به صورت دستی انجام می‌شوند، بهره‌وری را بهبود می‌بخشند.

توجه داشته باشید که ارتقا مدام SaaS، به دلیل اینکه تنها یک نسخه از نرم‌افزار وجود دارد، با چرخهٔ حیات توسعهٔ چاپک کاملاً هم‌خوانی دارد. از این رو، شرکت‌هایی همچون آمازون، ای‌پی^{۱۴۳}،

^{۱۴۱}Server

^{۱۴۲}Application Programming Interface

^{۱۴۳}eBay

فیسبوک^{۱۴۴}، گوگل^{۱۴۵} و دیگر سرویس‌دهندگان SaaS، همگی بر چرخهٔ حیات چاپک متکی هستند. همچنین شرکت‌های نرم افزاری سنتی‌تر همچون مایکروسافت^{۱۴۶} بیش از پیش و به صورت فرایندهای از روش چاپک در توسعهٔ محصولاتشان استفاده می‌کنند. فرآیند چاپک 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^{۱۴۷} SaaS، توسعهٔ دهنده‌گان می‌توانند از **کتابخانه‌های نرم افزاری**، حاوی کدهایی برای انجام وظایف مشترک بین بسیاری از نرم افزارهای کاربردی، استفاده گسترده کنند. از آنجایی که این کتابخانه‌ها را اغلب دیگران می‌نوشتند (به اصطلاح کتابخانه‌های شخص ثالث)، مزیت بازکاربرد نرم افزاری را با خود به همراه داشتن. در اواسط دهه ۲۰۰۰ در دنیای SaaS شروع به شکل‌گیری کرد: ظهور **معماری سرویس‌گرا (SOA)**، که در آن یک سرویس SaaS می‌توانست از سرویس‌های دیگری که توسط 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^{۱۴۸} نامیده شدند. نمونه‌های ارجح امروزی از این دسته از سرویس‌ها شامل پردازش کارت اعتباری، جست‌وجوی اینترنتی، مسیریابی برای راننده‌گی و بسیاری موارد دیگر هستند. با مستحکم شدن استانداردهایی برای ارائه و تعامل با چنین سرویس‌های خارجی، سرانجام مزیت مهم بازکاربرد نرم افزاری برای SaaS از راه رسید. فصل ۳ به جزئیات بیشتری در مورد SOA^{۱۴۹} و ریزسرویس‌ها می‌پردازد.

البته، ما هنوز باید به یک تفاوت عمدی بین SaaP و SaaS بپردازیم، و آن زیرساخت سخت‌افزاری است که نرم افزارهای کاربردی بر روی آن اجرا می‌شوند. در مورد SaaP، این ساخت‌افزار از رایانه‌های شخصی^{۱۵۰} میلیون‌ها کاربر تشکیل شده است. در قسمت بعدی، زیرساخت سخت‌افزاری که ممکن می‌سازد، بررسی می‌کنیم.

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

خودآزمایی ۱-۶-۱. برخی از محبوب‌ترین نرم افزارهای کاربردی شرکت گوگل که به صورت SaaS می‌شوند عبارت‌اند از جست‌وجوگر گوگل، گوگل میز، جی‌میل، تقویم گوگل و گوگل داکس. برای هر یک از این نرم افزارهای کاربردی، یک مزیت ارائه‌آن به عنوان SaaP به جای SaaS 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
- ۲- از دست نرفتن داده‌ها: جی‌میل، تقویم گوگل

¹⁴⁴Facebook

¹⁴⁵Google

¹⁴⁶Microsoft

¹⁴⁷Software as a Product

¹⁴⁸Microservice

¹⁴⁹Service Oriented Architecture

¹⁵⁰Personal Computer

۳- همکاری کاربران: گوگل داکس

- ۴- مجموعه داده‌های بزرگ و در حال تغییر: جستجوگر گوگل، گوگل مپز، گوگل نیوز و یوتیوب
- ۵- نرم‌افزار به صورت متمرکز در یک محیط واحد: جستجوگر گوگل
- ۶- عدم نیاز به حضور در محل به هنگام به روزرسانی نرم‌افزار کاربردی: گوگل داکس

خودآزمایی ۲-۶-۱ درست یا نادرست: اگر شما از فرآیند توسعه چاپک برای توسعه نرم‌افزارهای کاربردی 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 چاپک هستند. ■

۷-۱ به کاراندازی SaaS: رایانش ابری

اگر رایانه‌هایی که من از آنها صحبت کردام و طرفدارشان هستم در آینده حضور داشته باشند، آنگاه شاید روزی برسد که رایانش^{۱۵۱} همچون سامانه تلفن به عنوان یک ابزار و خدمات همگانی در بیاید... این رایانش همگانی می‌تواند پایه و بنیان یک صنعت جدید و مهم را بنا گرارد.

ـ جان مک‌کارتی^{۱۵۲}، در جشن صد سالگی مؤسسه فناوری ماساچوست^{۱۵۳} (MIT) در سال ۱۹۶۱

SaaS سه خواسته را برای زیرساخت فناوری اطلاعات^{۱۵۴} (IT) مطرح می‌کند:

- ۱- ارتباط، اینکه به هر مشتری امکان تعامل با سرویس داده شود.
- ۲- مقیاس پذیری^{۱۵۵}؛ اینکه مرکز سرویس‌دهنده باید بتواند با وجود نوسانات تقاضا در طول روز و در زمان‌های شلوغ سال همچنان کار کند. همچنین در مورد سرویس‌های جدید، باید بتوانند با افزایش سریع کاربران به راحتی کنار بیایند.
- ۳- دسترس پذیری^{۱۵۶}؛ اینکه هم سرویس و هم حامل ارتباطات باید بی‌وقفه در دسترس باشد: همه روزه و در هر ۲۴ ساعت از شبانه‌روز (که با 24×7 «نشان داده می‌شود»). استاندارد طلایی در مورد دسترس پذیری که توسط سامانه تلفن همگانی ایالات متحده تعیین شده است، در دسترس بودن در ۹۹,۹۹۹٪ زمان (معروف به «پنج تا نه») یا چیزی در حدود ۵ دقیقه قطعی در سال است. ویگاه آمازون چهار تا نه را هدف قرار داده است، که حتی برای سرویسی که به خوبی اداره می‌شود دشوار است.

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 گران‌قیمت 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 رایانه بزرگ آسان‌تر بوده است—رویکردی برخلاف آن خیلی زود صنعت را تصاحب کرد. مجموعه‌هایی از تعدادی رایانه کوچک و ارزان‌قیمت که به وسیله سوئیچ‌های

جان مک‌کارتی (۱۹۲۷-۲۰۱۱) در سال ۱۹۷۱ جایزهٔ تورینگ را دریافت کرد. وی مخترع زبان برنامه‌نویسی لیسب بود و از پیشگامان استفاده از تکنیک اشتراک زمانی بر روی رایانه‌های بزرگ بود. با آمدن خوشه‌هایی از سخت‌افزارهای معمولی و گسترش شبکه‌های سریع، رویابیش برای ایجاد «رایانش همگانی» اشتراکی به حقیقت پیوست.



^{۱۵۱}Computing

^{۱۵۲}John McCarthy

^{۱۵۳}Massachusetts Institute of Technology

^{۱۵۴}Information Technology

^{۱۵۵}Scalability

^{۱۵۶}Availability

ایترنوت^{۱۵۷} معمولی به هم متصل شده بودند، که بعداً به **خوشة**^{۱۵۸} معروف شدند، چندین مزیت نسبت به رویکرد قبلی یا همان سخت‌افزار «آهن گنده»^{۱۵۹} داشت:

- به علت اتکای آن‌ها به سوئیچ‌های ایترنوت برای اتصال به یکدیگر، خوشه‌ها بسیار مقیاس‌پذیرتر از سرورهای مرسوم سنتی هستند. در ابتدا خوشه‌ها شامل ۱۰۰۰ رایانه می‌شدند. مراکز داده^{۱۶۰} 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 بتواند با موقوفیت هزاران سرور را راهاندازی و اداره کنند. به طور مشخص، برخی از مراکز داده برای رسهولت از **ماشین‌های مجازی**^{۱۶۱} استفاده می‌کنند. یک ناظر ماشین مجازی^{۱۶۲} نرم‌افزاری است که یک رایانه واقعی را چنان با موقوفیت شبیه‌سازی و تقلید می‌کنند که شما می‌توانید حتی یک سیستم عامل را بر روی تجرید ماشین مجازی‌ای که ارائه می‌دهد اجرا کنید (پوپک و گلدبرگ^{۱۶۳}). هدف این است که این شبیه‌سازی با سریار^{۱۶۴} کم صورت گیرد و یکی از کاربردهای معمول آن برای ساده‌سازی توزیع نرم‌افزار در یک خوشه است. به این ترتیب، چندین نرم‌افزار کاربردی می‌توانند سخت‌افزار را پکدیگر به اشتراک بگذارند، بهطوری که هر نرم‌افزار کاربردی بر این باور است که بر روی یک نسخه مجزا و اختصاصی سیستم عامل اجرا می‌شود. اگر بنا بر این باشد که نرم‌افزارهای کاربردی سخت‌افزار، استفاده از **مجازی‌سازی**^{۱۶۵} در سطح سیستم عامل است. یک نمونه از این نوع از مجازی‌سازی، استفاده از ابزار محبوب داکر^{۱۶۶} است که به هر نرم‌افزار کاربردی اجازه می‌دهد تا در کانتینر^{۱۶۷} 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^{۱۶۸}، حافظه و فضای ذخیره‌سازی برای خوشه‌ها بسیار کمتر از روش «آهن گنده» است، شاید چیزی در حدود بیست برابر کمتر (باروسو و هلتسله ۲۰۰۹).

- اگرچه سرورها و سامانه‌های ذخیره‌سازی سنتی نسبت به قطعات مورد استفاده در خوشه‌ها قابل اطمینان‌تر هستند، اما زیرساخت نرم‌افزاری خوشه با بهره‌گیری حداکثری از افزونگی^{۱۶۹} 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.

با رشد مراکز داده اینترنتی، تعدادی از سرویس‌دهنده‌گان به این نتیجه رسیدند که هزینهٔ سرانه آن‌ها به مراتب کمتر از هزینه‌ای است که دیگران برای راهاندازی مراکز داده کوچک‌ترشان می‌پردازند.

^{۱۵۷}Ethernet

^{۱۵۸}Computer Cluster

^{۱۵۹}اصطلاح آهن گنده که ریشه آن به دهه ۱۹۷۰ میلادی برمی‌گردد، به معنی یک رایانه بسیار بزرگ، گران‌قیمت و سریع است و معمولاً به رایانه‌های بزرگ‌مانند بزرگ‌رایانه شرکت آی‌بی‌ام و یا آندرایانه شرکت کری اشاره دارد.

^{۱۶۰}Datacenter

^{۱۶۱}Virtual Machine

^{۱۶۲}Virtual Machine Monitor

^{۱۶۳}Overhead

^{۱۶۴}Virtualization

^{۱۶۵}Docker

^{۱۶۶}Container (Virtualization)

^{۱۶۷}Processor (Computing)

^{۱۶۸}Redundancy

فصل ۱. مقدمه‌ای بر توسعه چاپک و SAAS

این عمدتاً به علت صرفه به مقیاس ^{۱۶۹} است، زمانی است که شما تعداد ۱۰۰,۰۰۰ رایانه را در یک زمان خریداری و اداره می‌کنید. آن‌ها همچنین از بهره‌وری ^{۱۷۰} بالاتری سود می‌برند چراکه شرکت‌های بسیاری می‌توانند این مراکز داده عظیم را، که (پاروسو و هلتسله ۲۰۰۹) ^{۱۷۱} رایانه انبارگون ^{۱۷۲} نامیدند، به اشتراک گذاشته باشند. 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» میزان مصرف ^{۱۷۳} کسب درآمد کنند.

نتیجه آن با نام‌های سرویس‌های ابری عمومی، رایانش همگانی و یا اغلب رایانش ابری ^{۱۷۴} شناخته می‌شود. این‌گونه سرویس‌ها رایانش، فضای ذخیره‌سازی و ارتباطات را در ازای مقداری اندک پول برای هر ساعت ارائه می‌دهند (آزمبراست و دیگران ^{۱۷۵}). 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 ^{۱۷۶}، موتور اجرای برنامه گوگل ^{۱۷۷} و مایکروسافت آزور ^{۱۷۸}، نمونه‌هایی بیش رو از مدل رایانش به صورت پرداخت به میزان مصرف هستند که به نوعی «بینهایت مقیاس‌پذیر» ^{۱۷۹} بهشمار می‌روند. سرویس ابری همگانی یعنی اینکه امروزه هر شخصی با داشتن یک کارت اعتباری و یک ایدهٔ خوب می‌تواند یک شرکت عرضه SaaS 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 دهه 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.

¹⁶⁹Economies of Scale

¹⁷⁰Utilization

¹⁷¹Warehouse Scale Computer

¹⁷²Pay-as-you-go

¹⁷³Cloud Computing

¹⁷⁴Amazon Web Services (AWS)

¹⁷⁵Google App Engine

¹⁷⁶Microsoft Azure

¹⁷⁷Scalable

لوئیز باروسو، از مدیران ارشد شرکت گوگل و برنده جایزه ACM/IEEE Eckert-Mauchly در سال ۲۰۲۰، در ابتدای سخنرانی خود برای پذیرش این جایزه، تاریخچه مختصراً از رایانه‌های انبارگون را ارائه می‌دهد.^۱

بازی فارم‌ویل رکوردهای جدیدی چون ۱ میلیون کاربر تنها ۴ روز پس از معرفی شدند، ۱۰ میلیون کاربر بعد از ۲ ماه و ۷۵ میلیون کاربر بعد از ۹ ماه را از خود به جای گذاشت. (پیش از این بیشترین تعداد کاربرین ثبت شده یک بازی شبکه‌های اجتماعی، ۵ میلیون نفر بود). سرویس رایانش ابری ارتجاعی آمازون استفاده می‌کرد و به راحتی توانست محبوبیت خود را با پرداخت هزینه برای استفاده از خوش‌های بزرگ‌تر ادامه دهد.

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 مقرون 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. ■

۱-۸ به کاراندازی SaaS: مرورگرها و تلفن همراه

از حدود سال ۱۹۹۴ موقعيت خیره‌کننده وب به سرعت منجر به حذف تدریجی بسیاری از نرم‌افزارهای کاربردی SaaS دسکتاب^{۱۷۸} شد. رابطه‌های کاربری^{۱۷۹} کلاینت^{۱۸۰} اختصاصی سرویس‌های پولی مانند ای‌اوال^{۱۸۱} و کامپیوسر^{۱۸۲} با درگاه‌های وب^{۱۸۳} رایگانی مانند یاهو! جایگزین شدند. نرم‌افزارهای کاربردی SaaS تخصصی برای دسترسی به خدمات مبتنی بر اینترنت، مانند یودارا^{۱۸۴} برای رایانمه، با سرویس‌های رایانمه مبتنی بر مرورگر^{۱۸۵} مانند هات‌میل جایگزین شد. حتی نرم‌افزارهای کاربردی ای همچون مایکروسافت ورد نیز از سوی رقبای مبتنی بر مرورگر خود همچون گوگل داکس^{۱۸۶} تحت فشار قرار گرفتند. بنابراین مرورگر به یک کلاینت همگانی و فراگیر تبدیل شد: هر وبگاه می‌توانست تمامی اطلاعات لازم برای رندر^{۱۸۷} و نمایش داده شدن رابط کاربری‌اش را در هنگام بازدید به مرورگر ارائه دهد.

¹⁷⁸ Desktop

¹⁷⁹ User Interface

¹⁸⁰ Client

¹⁸¹ AOL

¹⁸² CompuServe

¹⁸³ Web Portal

¹⁸⁴ Eudora (Email Client)

¹⁸⁵ Browser (Web)

¹⁸⁶ Google Docs (Online Word Processor)

¹⁸⁷ Render (Computing)

این کار با استفاده از HTML^{۱۸۸}، یا همان زبان نشانه‌گذاری ابرمنتنی، صورت می‌گرفت. همانطور که در ادامه خواهیم دید، جاوااسکریپت بعداً به عنوان راهی برای غنی‌تر کردن تجربه تعاملی^{۱۸۹} صفحات وب^{۱۹۰} وارد صحنه شد، اما محتوای واقعی قابل مشاهده صفحه همواره از HTML^{۱۹۱} تشکیل شده است. همانطور که از نام آن پیداست، HTML^{۱۹۲} نمونه‌ای از یک زبان نشانه‌گذاری^{۱۹۳} است: متن را با نشانه‌گذاری (حاشیه‌نویسی‌هایی در مورد ساختار متن) به صورتی ترکیب می‌کند که تشخیص نحوی و قواعدی این دو از هم آسان باشد. از نظر فن، ۵ XML^{۱۹۴} (نسخه پرکاربرد کنونی) در واقع فقط یک نوع سند است که می‌تواند به زبان XML^{۱۹۵} 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 با منوی کشویی انتخاب‌ها، و مطابق با یک عنصر HTML بدون فرزند هستند. اغلب، یک جزء از صفحه شامل یک عنصر div است که عناصر دیگری را نیز در داخل خود جای داده است. عناصر div اغلب عناصری که از نظر منطقی با هم مرتبط هستند (و ممکن است تودرتو باشند) را باهم گروه‌بندی می‌کند. ظاهر نه تنها به تایپوگرافی^{۱۹۶} 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^{۱۹۷} است که در سال ۱۹۹۶ به عنوان راهی برای مرتبط کردن اطلاعات مربوط به رندر بصری با عناصر HTML معرفی شد. مفهوم کلیدی CSS عبارت است از انتخابگر^{۱۹۸}—عبارتی که با یک یا بیشتر از عناصر HTML در یک سند مطابقت دارد. حتی اگر شما توسعه‌دهنده‌ای نیستید که مسئول ظاهر بصری 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^{۱۹۹} استفاده می‌شود و به شما امکان می‌دهد صفحات وب تعاملی و غنی بسازید. در حالی که چندین روش وجود دارد که یک انتخابگر می‌تواند با یک عنصر class^{۱۹۷} طبیق داده شود، اما معمول‌ترین روش مرتبط کردن انتخابگر با خصوصیت^{۱۹۸} مربوط به عناصر است، زیرا در یک صفحه، چندین عنصر از انواع مشابه یا متفاوت می‌توانند خصوصیات کلاس یکسانی را به اشتراک بگذارند. شکل ۱-۷ 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:

- وقتی مرورگر صفحه را بارگیری می‌کند، به دنبال یک یا چند عنصر link^۱ می‌گردد، که باید فرزندان عنصر head^۲ از سند head^۳ باشند. این عناصر شیوه‌نامه‌ها^{۱۹۹} را مشخص می‌کنند که باید از آن‌ها در کنار این سند HTML استفاده شوند. در این مثال، خصوصیت href^۴ از عنصر

به عنوان یادآوری، در ابتدای فصل و در صفحه پیش‌نیازها و مقاهم، مطالبی را برای یادگیری خودآموز مبانی اولیه HTML و CSS پیشنهاد کرده‌ایم.

¹⁸⁸HyperText Markup Language

¹⁸⁹Interactive

¹⁹⁰Webpage

¹⁹¹Markup Language

¹⁹²Extensible Markup Language

¹⁹³Typography

¹⁹⁴Cascading Style Sheets

¹⁹⁵Selector (CSS)

¹⁹⁶jQuery

¹⁹⁷Element

¹⁹⁸Attribute

¹⁹⁹Stylesheet (Web Development)

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

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <link rel="stylesheet" href="https://getbootstrap.com/docs/4.0/dist/css/
5              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
23                 predators
24             </li>
25         </ul>
26     </div>
27 </body>
28 </html>

```

شکل ۷-۷: یک سند ۵ HTML در ساده‌ترین حالت یک فایل متنی 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 بروکر فته است از SGML (زبان شناسه‌گذاری تعمیم‌بافته استاندارد)، که در واقع استانداردسازی مذوونی است از زبان نشانه‌گذاری تعمیم‌یافته 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.

پیوند^{۲۰۰} (یا همان مقصود و هدفش) به شیوه‌نامه اصلی چارچوب بوت استرپ^{۲۰۱} اشاره دارد که در ادامه به آن می‌پردازیم.

- مرورگر، هر شیوه‌نامه CSS^{۲۰۲} 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، زنگ‌ها و موارد دیگر را مشخص کنند.

- هنگام نمایش صفحه، مرورگر قوانین تعریف شده CSS را با عناصر منطبق در هر صفحه نمایش داده شده مطابقت می‌دهد. در این مثال، چارچوب بوت استرپ قوانین پایه‌ای برای شیوه نمایش هر نوع عنصر (h1، h2، p، وغیره) را ارائه می‌کند، و خصوصیات class در عناصر مختلف برای مطابقت با انتخابگرهای خاص CSS در چارچوب بوت استرپ آورده شده‌اند تا با استفاده از آن‌ها قالب‌بندی خاصی به عناصر صفحه داده شود.

در حالی که قواعد CSS ساده است، ساخت شیوه‌نامه‌هایی که از نظر بصیری جذاب باشند، نیازمند مهارت‌های طراحی و تایپوگرافی است. آن دسته از ما که قادر آن مهارت‌ها هستند، بهتر است که از شیوه‌نامه‌هایی موجود طراحی شده توسط حرفه‌ای‌ها استفاده کنند. مجموعه‌هایی از این شیوه‌نامه‌هایی از پیش طراحی شده را تحت عنوان چارچوب‌های CSS می‌شناسند. این مجموعه‌ها گاهی شامل کدهایی به زبان جاوا اسکریپت نیز هستند تا جلوه‌های بصیری بیشتری (مثل انیمیشن و یا محو کردن) را که نمی‌توان به تنها بـ CSS پوشش داد، به نمایش بگذارند. این نوع از مجموعه‌ها تحت عنوان چارچوب‌های پیشنهادی^{۲۰۳} شناخته می‌شوند. یکی از این چارچوب‌های پیشنهادی پرکاربرد که ما نیز در این کتاب بارها به آن اشاره می‌کنیم چارچوب بوت استرپ است، که یک پروژه متن باز^{۲۰۴} ارائه شده توسط شرکت توییتر^{۲۰۵} است. یک چارچوب CSS خوب، حداقل چهار مزیت اصلی زیر ارائه می‌دهد:

- 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.

- ارائه یک تعریف استعاری شطرنجی از ابعاد صفحه برای مشخص کردن چیدمان اجزا در آن. مثلاً در مورد بوت استرپ، 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.

- پشتیبانی از طراحی وب واکنش‌گرا^{۲۰۶} 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، بهطور خودکار به عنوان گزینه‌های عمودی که روی هم سوار شده‌اند نمایش داده می‌شود، حتی اگر به صراحت چنین دستورالعمل‌هایی را ارائه نکرده باشد. وبگاه <https://getbootstrap.com/docs/4.0/examples/navbars/> نمونه‌هایی از نحوه رفتار نوارهای پیمایش در بوت استرپ را با تغییر اندازه صفحه نمایش نشان می‌دهد.

وبگاه باع ذن CSS نشان

می‌دهد چگونه محتوای HTML
پیکسان را می‌توان با استفاده از
شیوه‌نامه‌های CSS به طور
چشمگیری متفاوت ساخت.

²⁰⁰Link (Web Hyperlink)

²⁰¹Bootstrap (Front-end Framework)

²⁰²Frontend

²⁰³Open Source (Software)

²⁰⁴Twitter

²⁰⁵Responsive Web Design

- پشتیبانی از دسترسی‌پذیری^{۲۰۶} برای کاربران دارای معلولیت. به طور مثال، محتوایی را که باید از نظر بصری پنهان باشد، می‌توان طوری ارائه داد که همچنان برای فناوری‌های کمکی مانند صفحه‌خوان‌ها قابل دسترسی باشد.

بارگشت به SaaS؛ گسترش
 نرم‌افزارهای کاربردی قابل نصب توسط کاربر یک مزیت عمده در این کار بسیار موفق شد و متعاقباً به طور گسترده مورد کپی‌برداری قرار گرفت. تا سال ۲۰۱۷، تنها ده سال بعد، حدود یک سوم جمعیت جهان گوشی‌های هوشمند^{۲۰۹} داشتند، که در مجموع بیشتر از دسکتاپ‌ها یا لپ‌تاپ‌ها^{۲۱۰} از وبگاه‌ها بازدید می‌کردند (انگ ۲۰۱۸). تا حد خوبی، سبکها و نحوی کاربران باید یک بار دیگر به صورت دستی نرم‌افزارهای کاربردی خود را در صورت یافتن اشکال امنیتی یا زمانی که دستگاه‌های خود را ارتقا می‌دهند، به روزرسانی کنند. و البته این به روزرسانی‌ها در مورد نرم‌افزارهای کاربردی تلفن همراه بسیار بیشتر از نرم‌افزارهای گذشته است. به طور مثال، در سال ۲۰۱۴، نرم‌افزار کاربردی تلفن همراه توییتر به طور متوسط هر ۲۰ روز یک بار به روزرسانی می‌شد.^۹

چارچوب‌های CSS/HTML با سلطه دستگاه‌های تلفن همراه اهمیت ویژه‌ای پیدا کرده‌اند. شرکت اپل^{۲۰۷} در سال ۲۰۰۷ آیفون^{۲۰۸} را معرفی کرد. اگرچه قطعاً این اولین گوشی هوشمندی نبود که امکان نصب نرم‌افزارهای کاربردی جانبی یا مرور وب را فراهم می‌کرد، اما اولین گوشی بود که در این کار بسیار موفق شد و متعاقباً به طور گسترده مورد کپی‌برداری قرار گرفت. تا سال ۲۰۱۷، تنها ده سال بعد، حدود یک سوم جمعیت جهان گوشی‌های هوشمند^{۲۰۹} داشتند، که در مجموع بیشتر از دسکتاپ‌ها یا لپ‌تاپ‌ها^{۲۱۰} از وبگاه‌ها بازدید می‌کردند (انگ ۲۰۱۸). تا حد خوبی، سبکها و تنظیمات تعریف شده توسط CSS که با دقت طراحی شده‌اند می‌توانند محتوای HTML یکسانی را در طیف وسیعی از اندازه‌های مختلف صفحه قابل استفاده کنند. به همین دلیل، در حالی که شکل ۱-۸ نشان می‌دهد که چندین رویکرد برای توسعه برنامه‌های کلاینت وجود دارد، ما در این کتاب توصیه به ساخت نرم‌افزارهای کاربردی «اول برای موبایل» می‌کنیم که با استفاده از HTML 5، CSS و جاوااسکریپت ایجاد شده‌اند و شاید در ادامه آن‌ها را به برنامه‌های وب پیشرو^{۲۱۱} ارتقا دهید (قسمت ۱۰-۶). این رویکرد از ابزارهای گسترده موجود در آن اکوسیستم، به ویژه چارچوب‌های نمایش مانند بوت‌استرپ، کتابخانه‌های دستکاری و انجام تغییرات بر روی DOM^{۲۱۲} مانند چی‌کوئری (قسمت ۴-۶) و ابزارهای آزمونی مانند جَزْمِین^{۲۱۳} (قسمت ۸-۶) بهره می‌برد.

على‌رغم تفاوت‌های موجود در روش‌های ساخت SaaS موبایل یا دسکتاپ، همه‌این نرم‌افزارهای کاربردی 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 دورست^{۲۱۴} استفاده می‌کنند.

206 Accessibility

207 Apple

208 iPhone

209 Smartphone

210 Laptop

211 Progressive Web Application

212 Document Object Model

213 Jasmine (JavaScript Testing Tool)

214 Remote (Networking)

معایب	مزایا
وبگاه واکنش‌گرا و یا «اول برای موبایل» («نرم‌افزار کاربردی چندصفحه‌ای»)	
عدم امکان فهرست شدن بر روی فروشگاه‌های نرم‌افزار کاربردی عدم امکان دسترسی به امکانات پیشرفته سخت‌افزار آن سکو مممولاً در صورت قطع اتصال به اینترنت ضعیف عمل می‌کند بسته به پیچیدگی نرم‌افزار کاربردی، کارایی ممکن است به میزان قابل توجهی از نسخه مشابه که به صورت خاص برای آن سکو ساخته شده است، کمتر باشد	استفاده از همان ابزارها، چارچوبها و زبان‌های برنامه‌نویسی مشابه برای نرم‌افزارهای کاربردی دسکتاپ قابل استفاده بر روی دستگاه‌های مختلف، پس نیازی به توسعه و نگهداری چند نسخه مجزا نیست کاربر هرگز نیاز به نصب بروزرسانی‌ها ندارد می‌توان آن را جویی ساخت که حتی در صورت قطع اتصال به اینترنت بتوان از آن استفاده کرد قراردادن آیکن مربوطه بر روی صفحه اصلی دستگاه کاربر به عنوان یک نشانک وب

(PWA) برنامه وеб پیشرو

همان مزایا و معایب ویگاه واکنش‌گرا، اما حتی در صورت قطع ارتباط با اینترنت همچنان می‌تواند به خوبی عمل کند

نرم‌افزار کاربردی اختصاصی سکوی اندروید (جاوا) یا آی‌اواس (آجکتیو-سی)

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 بهموضع بروزرسانی‌ها باید به خود کاربران متکی بود باید از نسخه‌های قدیمی پشتیبانی کرد تا زمانی که اکثر کاربران نسخه نصب شده خود را به روزرسانی کرده باشند پشتیبانی از چندین سکو سخت‌افزاری ممکن است نیاز به حفظ و نگهداری چندین مجموعه کد داشته باشد	بهترین کارایی امکان فهرست شدن بر روی فروشگاه‌های نرم‌افزار کاربردی دسترسی تضمین شده به تمام امکانات سخت‌افزار آن سکو
---	---

شکل ۱-۸: سه رویکرد برای پیاده‌سازی کلاینت تلفن همراه. امروزه، اکثریت قریب به انفاق 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 ابرمنتنی) 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 برگیرنده محتوا هستند. به طور کلی، عناصر ساختار منطقی بخش‌های سند را توصیف می‌کنند، اما نه اینکه چگونه سند هنگام زدن و نمایش روی صفحه نشان داده می‌شود.
- **شیوه‌نامه آبشاری (CSS)** یک زبان شیوه‌نامه است که ویژگی‌های بصری عناصر را در یک صفحه وب توصیف می‌کند. یک شیوه‌نامه مجموعه‌ای از ویژگی‌های بصری را با انتخابگرهایی که با یک 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 دهنند.
- 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 نرم‌افزارهای کاربردی SaaS را به خود اختصاص می‌دهند. یکی از راه‌های ساخت نرم‌افزارهای کاربردی کلاینت «اول برای موبایل» 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 نرم‌افزارهای کاربردی کلاینت «اول برای موبایل» ساخت نرم‌افزارهای کاربردی قابل نصب و اختصاصی برای سکویهای مختلف است. این‌گونه نرم‌افزارهای کاربردی ممکن است امکان دسترسی به برخی از امکانات خاص دستگاه را بدهند 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.

گریس موری هاپر (۱۹۰۶-۱۹۹۲)
 یکی از اولین برنامه‌نویسان بود.
 او اولین کامپیوتر 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 او نام‌گذاری شد:
 پیواس اس هاپر



خودآزمایی ۱-۸-۱. چگونه می‌توانید اطمینان حاصل کنید که شیوه‌نامه‌های CSS یکسانی برای همه صفحات و بگاه یا نرم‌افزار کاربردی شما استفاده می‌شود؟
 ◊ هر یک از سندهای HTML باید شامل پیوندهای مربوط به شیوه‌نامه‌های خود باشد، بنابراین شما می‌بایست مطمئن شوید که عناصر `<link>` یکسانی در عنصر `<head>` همه صفحات و بگاه یا نرم‌افزار کاربردی شما وجود دارد. ■

۹-۱ زیباکد در مقابل میراث کد

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

—گریس موری هاپر^{۲۱۵}

برخلاف سخت‌افزار، از نرم‌افزار انتظار می‌رود تا در طول زمان دچار رشد و تکامل شود. در حالی که طراحی‌های سخت‌افزار پیش از ساخته شدن و ورود به بازار باید کامل شده باشند، طراحی اولیه

نرم‌افزار می‌تواند به راحتی وارد بازار شده و در طول زمان ارتقا یابد. اساساً، هزینه ارتقا بعد از ورود به بازار برای سخت‌افزار نجومی و برای نرم‌افزار قابل قبول و مقرنون به صرفه است. بنابراین نرم‌افزار می‌تواند به یک جاودانگی فناورانه دست یابد، در حالی که سخت‌افزارها نسل به نسل کهنه شده و از بین می‌روند، یک نرم‌افزار می‌تواند در طول زمان بهتر و بهتر شود. تنها رفع اشکالات باعث **تکامل نرم‌افزار** نیست، بلکه افزودن ویژگی‌های جدید بنا به درخواست کاربران، تطبیق با نیازهای در حال تغییر کسب و کار، بهبود کارایی^{۲۱۶} و سازگار شدن با محیط‌های جدید همگی از محرك‌های تکامل نرم‌افزار هستند. مشتریان نرم‌افزار انتظار دارند تا در طول زمانی که از یک نرم‌افزار استفاده می‌کنند، در مورد نسخه‌های جدید آن اطلاع پیدا کرده و بتوانند آن‌ها را نصب کنند؛ یا حتی مشکلات نسخه‌های جدید را گزارش دهند تا توسعه‌دهندگان بتوانند آن‌ها را تصحیح کنند. آن‌ها حتی گاهی مبلغ اضافی به صورت سالیانه تحت عنوان هزینه نگهداری پرداخت می‌کنند تا این امکان بهره‌مند شوند.

همانطور که نویسندهان هموارند تا آیچه خلق کرده‌اند و زاده تخیل‌شان است تا مدت‌ها خوانده شود که لقب کتاب کلاسیک را به خود بگیرد (چیزی حدود ۱۰۰ سال برای یک کتاب)، مهندسین نرم‌افزار هم باید امیدوار باشند که عمر چیزی که خلق کردند تا مدت‌ها ادامه یابد. البته نرم‌افزار نسبت به کتاب این برتری را دارد که در طول زمان می‌تواند بهبود یابد. در حقیقت، یک نرم‌افزار بادوام را معمولاً دیگران نگهداری می‌کنند و بهبود می‌بخشند و مسئولیت را از دوش خالقین اصلی اثر برمی‌دارند.

حال می‌رسیم به چند اصطلاح جدید که ما در طول کتاب از آن‌ها استفاده می‌کنیم. اصطلاح **میراث کد**^{۲۱۷} به نرم‌افزاری اشاره دارد که با وجود قدیمی بودن، همچنان مورد استفاده قرار می‌گیرد چون پاسخگوی نیازهای مشتری است. شصت درصد هزینه‌های نگهداری نرم‌افزارها صرف افزودن قابلیت جدید به نرم‌افزارهای قدیمی یا میراث‌کدها می‌شود، در حالی که تنها ۱۷٪ هزینه‌ها برای رفع اشکالات خرج می‌شوند، پس این میراث‌کدها در واقع نرم‌افزارهای موفقی هستند.

لفظ «میراث» یک بار منفی با خود دارد و نشان‌دهنده این است که تکامل کد به دلیل نازبیابی‌های موجود در طراحی و یا استفاده از فناوری‌های قدیمی دشوار است. در مقابل میراث کد، ما اصطلاح «زیبایگد»^{۲۱۸} را برای کدهای بادوام که به راحتی قابل بهبود هستند، به کار می‌بریم. 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 مشتری را ریخته می‌شود. ما مثال‌هایی که به زیبایگد می‌انجامند را با نماد تابلوی مونا لیزا نشانه‌گذاری می‌کنیم. به همین ترتیب، مثال‌های مربوط به میراث کد با چرتکه مشخص می‌شوند که یک وسیله محاسباتی قدیمی اما ماندگار است که در طول سال‌ها با تغییرات اندکی همراه بوده است.

در فصول بعدی کتاب مثال‌هایی از زیبایگد و میراث کد را به شما نشان می‌دهیم، به این امید که الهام‌بخش شما باشند تا طراحی‌های خودتان را جویی انجام دهید که بعدها را به راحتی بتوانند تکامل پیدا کنند. جای تعجب است که با وجود پذیرش گسترده اهمیت بهبود میراث‌کدها و نرم‌افزارهای قدیمی، این مبحث به طور معمول در درس‌ها و کتاب‌های دانشگاهی نادیده گرفته می‌شود. ما به سه دلیل در کتاب به این‌گونه نرم‌افزارها می‌پردازیم؛ اول اینکه شما می‌توانید هزینه ساخت یک برنامه را با یافتن کدهای موجود و بازکاربرد آن‌ها کاهش دهید. یک منبع خوب برای این کار، نرم‌افزارهای متنه‌باز است. دوم اینکه آموختن روش تولید کدی که بتواند به راحتی توسط نسل‌های 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 روش‌های طرح-و-ثبت، در مدل چاپک، شما از تکرار دوم به بعد به طور مداوم در حال بازبینی کد هستید تا آن را بهبود بخشید و امکانات جدید به آن بیفزایید. بنابراین مهارت‌هایی که در مدل چاپک می‌آموزید دقیقاً همان‌هایی هستند که برای تکامل میراث‌کدها به آن‌ها نیاز دارید (فارغ از اینکه چگونه ساخته شده است) و این استفاده دوگانه از تکنیک‌های مدل چاپک، کار ما را برای پوشش میراث‌کدها در همین یک کتاب بسیار ساده‌تر می‌کند.

قدیمی‌ترین برنامه زنده مورد استفاده احتمالاً موكا ^{۲۱۹} با MOCAS (مخف عبارت مکانیزه کردن سرویس‌های مدیریت قرارداد) است که در سال ۱۹۵۸ توسط وزارت دفاع آمریکا خریداری و تا سال ۲۰۰۵ نیز همچنان از آن استفاده می‌شد.



چرتکه هنوز در بسیاری از مراطق جهان استفاده می‌شود، با اینکه هزاران سال عمر دارد.

²¹⁶Performance

²¹⁷Legacy Code

²¹⁸Beautiful Code

چکیده: نرم افزار موفق می تواند چندین دهه عمر کند و از آن انتظار می رود که تکامل پیدا کرده و بهبود یابد. برخلاف بخش سخت افزاری رایانه ها که در زمان تولید به کمال و مرحله پایانی طراحی رسیده است و پس از چند سال از رده خارج می شود. یکی از اهداف کتاب این است که به شما آموزش دهد چگونه شناسن تویلید زیباگد را بالا ببرید تا نرم افزار شما بتواند عمری طولانی و مفید داشته باشد.

خودآزمایی ۱-۹-۱. برنامه نویسان به ندرت قصد نوشتن کد بد را دارند. با توجه به ایده های مطرح شده در قسمت ۱-۵ در مورد بهره وری، به طور خلاصه توضیح دهید چگونه نرم افزاری که مدت ها پیش و هنگام نوشته شدن با کیفیت محسوب می شده است، ممکن است امروز به عنوان نرم افزاری قدیمی که نگهداری آن بسیار دشوار است قلمداد شود.

به دلیل افزایش مداوم سطح تجرید ابزارهای نرم افزاری، امروزه توسعه دهنده ها اغلب می توانند 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 سخت تر است، حتی اگر در زمان نوشتن آنها ممکن است بسیار مدرن و آخرین فناوری روز تلقی شده باشند. بدون شک کدی که ما امروز می نویسیم در چند دهه دیگر به عنوان کدی قدیمی، منسخه و از مد افتاده در نظر گرفته خواهد شد!

۱۰-۱ گذری بر این کتاب و نحوه استفاده از آن

همانطور که در ابتدای این فصل در مفاهیم و پیش نیازها توضیح داده شد، برای تبدیل شدن به یک مهندس نرم افزار ماهر، علاوه بر درک مفهومی مطالب، تمرين عملی فراوان نیز نیاز است. بنابراین، هدف ما در هر فصل این است که به شما مبانی مفهومی لازم را برای کار روی تمرين ها، جایی که یادگیری واقعی اتفاق می افتد، ارائه دهیم.

بقیه کتاب به دو بخش تقسیم شده است. بخش اول نرم افزار به صورت یک سرویس را توضیح می دهد و بخش دوم با تاکید بر روی مدل چاپک، به توضیح توسعه نرم افزار به شکل امروزی می پردازد. فصل ۳، بخش اول کتاب را با توضیحی در مورد معماری نرم افزارهای کاربردی SaaS آغاز می کند. این فصل همچنین به این موضوع می پردازد که چطور وب از مجموعه ای از صفحات ایستا^{۲۱۹} به اکوسیستمی از سرویس هایی با مشخصه واسطه های برنامه نویسی «مبتنی بر REST»^{۲۲۰} تبدیل شد. از آنجایی که زبان ها و چارچوب های برنامه نویسی به سرعت در حال تکامل هستند، ما باور داریم آموختن چگونگی یادگیری زبان ها و چارچوب های برنامه نویسی مهارتی بالارش تر از صرفاً دانستن یک زبان و چارچوب برنامه نویسی خاص است. بدین ترتیب، فصل ۲ روش مدنظر ما را برای کسب این مهارت معرفی می کند. در این راه، ما از زبان برنامه نویسی روبی به عنوان مثالی برای آن دسته از برنامه نویسان که پیش از این با زبان برنامه نویسی مدرن دیگری همچون جاوا یا پایتون آشنایی دارند، استفاده می کنیم.

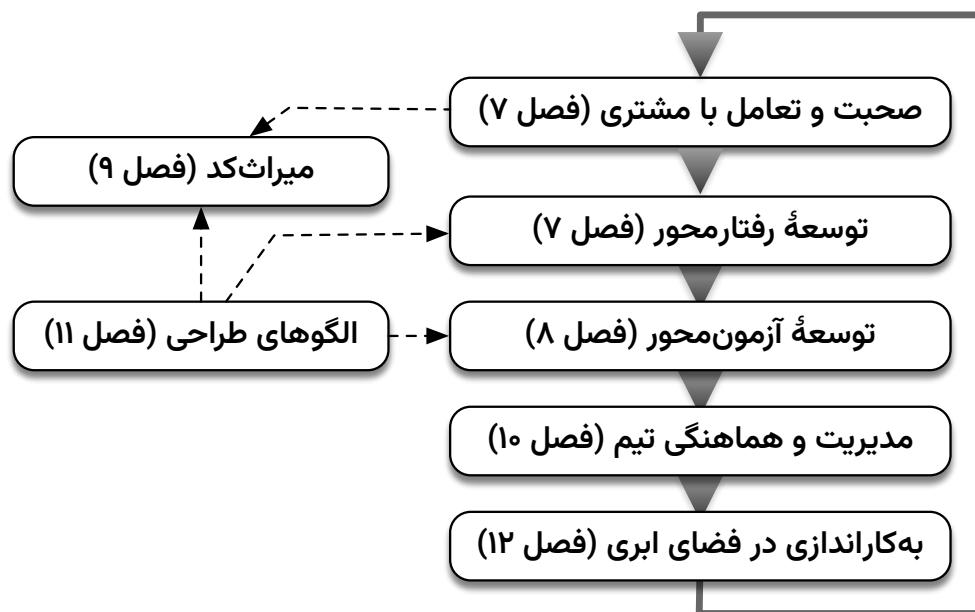
امروزه دلیل اصلی یادگیری یک زبان برنامه نویسی جدید اغلب میل به استفاده از یک چارچوب برنامه نویسی متکی بر آن زبان است. یک چارچوب خوب هم یک معماری برنامه سازی خاص را تقویت می کند و هم از ویژگی های یک زبان خاص نهایت استفاده را می کند تا توسعه را در صورت انتسابی با آن معماری آسان کند. فصل ۴ به معرفی مفاهیم پایه ای چارچوب ریزل^{۲۲۱} و قلب تپنده آن یعنی معماری مدل-تما-کنترلر^{۲۲۲} می پردازد. فصل ۵ به بررسی امکانات پیشرفته تر این چارچوب می پردازد و همچنین با عمق بیشتری نشان می دهد که این چارچوب چگونه از ویژگی های زبان روبی نهایت استفاده را می کند. ما محتویات آموزشی مربوط به ریزل را به این علت بین دو فصل تقسیم

219 Static

220 RESTful

221 Rails (Simplified verion of Ruby on Rails)

222 Model-View-Controller (Software Architecture)



شکل ۱-۹: یک تکرار از چرخه حیات نرم‌افزاری چاپک و رابطه آن با فصول این کتاب در بخش دوم، پیکان‌های خطچین نشان‌دهنده روابط فرقی و جانی بین مراحل یک تکرار هستند. در مقابل پیکان‌های نویز، روید و جریان معمول را نشان می‌دهند. همانطور که بیشتر نیز اشاره شد، فرآیند چاپک هم بر روی نرم‌افزارهای کاربردی جدید و هم بر روی نرم‌افزارهای کاربردی قدیمی‌تر و میراث‌کدها قابل انجام است.

کردیم تا کسانی که می‌خواهند هرچه سریع‌تر شروع به نوشتن یک نرم‌افزار کاربردی بکنند، تنها با مطالعه فصل ۴ بتوانند دست به کار شوند. اگر از پیش با رویی و یا ریز 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 جاواسکریپت، چارچوب جی‌کوئری و ابزار آزمون جزمین می‌پردازد. قسمت توضیحات مربوط به جزمین فرض می‌کند که شما آشنایی ابتدایی با فرآیند آزمون دارید، پس اگر حس کردید که مطالب آن قسمت کمی برای شما جدید و ناشنا هستند، می‌توانید مطالب آن قسمت را بعد از مطالعه فصل ۸ بررسی کنید. همانطور که چارچوب ریلز قدرت و بهره‌وری زبان رویی در تولید قسمت سمت سرور نرم‌افزار کاربردی SaaS را تقویت می‌کند، چارچوب جی‌کوئری نیز قدرت و بهره‌وری زبان جاواسکریپت 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 به کاراندازی یک نرم‌افزار کاربردی 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 توصیه **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 **روایتهای کاربری** صورت می‌گیرد و در فصل ۷ خواهیم دید چگونه می‌توان این روایتهای 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 و پیشگاهی‌ها می‌پردازد. برای دنبال کردن و محاسبه این مقدار، ابزاری به نام **پیووچال ترکر** در این فصل نیز معرفی می‌شود.

فصل ۸، **توسعه آزمونمحور (TDD)** را پوشش می‌دهد. این فصل نحوه نوشتن یک کد خوب و



آزمون‌پذیر^{۲۲۳} را شرح می‌دهد و به معرفی یک ابزار آزمون به نام آر-اسپیک برای نوشتن آزمون‌های واحد می‌پردازد. ابزارهای دیگری به نام‌های گارد^{۲۲۴} برای خودکارسازی اجرای آزمون‌ها و سیمپل‌کاو^{۲۲۵} برای اندازه‌گیری میزان پوشش آزمون نیز در این فصل معرفی می‌شوند.

فصل ۹ چگونگی برخورد با کدهایی را که از قبل نوشته شده‌اند که شامل میراث‌کدها و بحث ارتقا آن‌ها نیز می‌شود، توضیح می‌دهد. این فصل نشان می‌دهد که چطور از BDD و TDD هم برای فهم کد و هم بازسازی 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 کارا ارائه می‌شود. همچنین این فصل به تشریح سامانه کنترل نسخه گیت^{۲۲۶} و سرویسی مربوط به آن با نام گیت‌هاب^{۲۲۷} می‌پردازد تا نشان دهد چگونه اعضاً 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 حل‌هایی 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 روبي می‌توان الگوهای طراحی را مورد استفاده مجدد و بازکاربرد قرار داد. در این فصل رهنمودهایی نیز درباره توشنن کلاس‌های خوب داده شده است. این فصل شما را به اندازه کافی با UML^{۲۲۸} (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 درباره فرآیند توسعه چاپک هستند، در مورد معماری نرم‌افزار^{۲۲۹} است. ما باور داریم که این ترتیب به شما کمک می‌کند که بتوانید یک تکرار چاپک را سریع‌تر آغاز کنید و فکر می‌کنیم که هرچه تکرارهای بیشتری را انجام دهید، چرخه حیات چاپک را بهتر خواهید فهمید. با این حال همانطور که شکل ۹-۱ نشان می‌دهد، از آنجایی که الگوهای طراحی از اصول بنیادین در فرآیندهای BDD و TDD هستند، تسلط بر آن‌ها برای نوشتن و یا بازسازی کد بسیار مفید است.

فصل ۱۲ توصیه‌هایی عملی ارائه می‌دهد چگونه ابتدای 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 و قابلیت اطمینان^{۲۳۰} می‌پردازد که مختص به کاراندازی SaaS هستند.

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

پُفک. همانطور که کنفوپسیوس^{۲۳۱} گفت: «می‌شنویم و فراموش می‌کنم، می‌بینم و به 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^{۲۳۲} (پُفک) را که در متن این کتاب گنجانده شده‌اند، انجام دهید. هر تمرين پُفک

۲۲۳Testable

۲۲۴Guard (Testing tool)

۲۲۵SimpleCov (Testing tool)

۲۲۶Coverage (Testing)

۲۲۷Refactoring (Code)

۲۲۸Git (Version Control System)

۲۲۹GitHub

۲۳۰Unified Modeling Language

۲۳۱Software Architecture

۲۳۲Reliability

۲۳۳Confucius

۲۳۴Coding/Hands-On Integrated Projects (CHIPS)

۲۳۵ما در این کتاب عبارت «پروژه‌ها و فعالیت‌های عملی/کدنویسی» را به صورت Coding/Hands-On Integrated Projects می‌دانیم.

حاوی راهنمایی و نکات قابل توجهی برای خودآموزی است که برای تکمیل آن تمرين باید انجام دهید. اگر از این کتاب در کنار مطالب درسی آتلاین رایه شده بر روی گُدیو^{۲۳۶} استفاده می‌کنید (چه در محیط کلاس درس، چه به صورت خودآموز یا در دوره آموزش از راه دور اِدگُس^{۲۳۷}، جاپه‌جایی بین مطالب آموزشی محتوامحور (COD^{۲۳۸}) و پروژه‌ها و فعالیت‌های عملی/کدنویسی (پُفَّک) 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 مراجعه کنید.

واژه‌شناسی. در حین فرو رفتن در این اکوسیستم غنی، با بسیاری از اصطلاحات فنی جدید (و واژگان باب روز^{۲۳۹}) رویه‌رو 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^{۲۴۰}، 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 بیش 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 تصورات نادرست رایج یا مشکلاتی را توضیح می‌دهد که اگر هوشیار نباشد، به راحتی ممکن است در دام بیفتید و آن‌ها را تجربه کنید. پس از آن نکات پایانی برای ارائه متابعی 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 کپی کردن کدی از پیش نوشته شده نیست. متن این کتاب (COD) مبانی مفهومی کافی را به شما می‌دهد تا بتوانید بر روی پُفَّک (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 اِدگُس)، 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.

خودآزمایی ۱-۱۵. کدام یک برای یادگیری سریع فرآیند توسعهٔ SaaS از همگی مهم‌تر است: درک مبانی مفهومی، خواندن کد، یا نوشتن کد؟

◊ همگی مهم هستند. با کپی 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

ترجمه کردیم و سرnam «پُفَّک» را به عنوان معادل فارسی کوتاه‌نوشت CHIPS که در نسخه اصلی کتاب مورد استفاده قرار گرفته است، در نظر گرفتیم.

²³⁶Codio

²³⁷edX

²³⁸Content-Oriented Didactic

²³⁹Buzzword

²⁴⁰Kindle

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 ثبیت شدن درک شما از مبانی مفهومی کمک می‌کند. ■

۱۱-۱ باورهای نادرست و خطرهای پنهان

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

—سناتور موریس اودال^{۲۶۱}

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 مباحثت فصل 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 همه‌جیز بهترین نیست. برای مثال روش چاپک در مورد نرم‌افزارهای امنیتی و حساس ممکن است ناکارآمد باشد.

تجربه ما 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 متداولوzi مورد استفاده، رعایت می‌کنید. تمامی فضول در بخش دوم کتاب با مقایسه‌های از دید روش طرح-و-ثبت تمام می‌شوند تا بهتر بتوانید این اصول را فرا بگیرید و همچنین بتوانید در صورت نیاز از سایر متداولوzi‌ها استفاده کنید.

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



خطر پنهان: نادیده گرفتن هزینه طراحی نرم‌افزار

از آنجایی که تولید ابیوه و نشر نرم‌افزار هزینه‌چندانی 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 هزینه‌طراحی و آزمون نرم‌افزار را که ممکن است قسمت عمده‌ای از کل هزینه پروژه نرم‌افزاری باشد، نادیده می‌گیرد. نداشتن هزینه‌تولید 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 سانتایانا^{۲۴۲}

مهندسي نرم‌افزار يك رشتة مهندسي نسبتا جوان است، اما رشتة‌هاي است که به سرعت رو به‌سوی پيشرفت دارد. اگر سعى كنيد فناوري‌هاي نرم‌افزاری را بیاموزيد در حالی که زمينه‌هاي تاریخی را که در شکل‌گيري آن‌ها نقش داشته‌اند نادیده بگيريد، در خطر انتخاب‌هاي ناآگاهانه درباره ابزارهایی که باید استفاده کنید قرار می‌گيريد، و یا بدتر از آن، اصطلاحاً «چرخ را دوباره اختراع می‌کنید» بدون آن‌که از تجربیات دیگران درس بگيريد. به عنوان مثال، اگر با همکاران خود در مورد مناسب بودن استفاده از نود جي اس^{۲۴۳} به عنوان سرور نرم‌افزار کاربردي خود بحث می‌کنيد، اما با بحث‌هاي طولاني مدت مربوط به «ريسمان‌ها^{۲۴۴}» در مقابل رويدادها^{۲۴۵} در جامعه توسعه‌دهنگان نرم‌افزارهای سيستمي آشنا نیستيد، در بهترین حالت ممکن است يك بحث ناآگاهانه داشته باشيد و در بدترین حالت به سرعت گرفتار بدختي خواهيد شد. به طور مشابه، خرشن کاربرد^{۲۴۶} در پايگاه‌هاي داده^{۲۴۷} «NoSQL»^{۲۴۸} منعكس‌کننده همان اتفاقات است که در نهايیت منجر به ابداع **مدل رابطه‌ای** و چيره شدن آن بر **مدل سلسنه‌مراتبي**^{۲۴۹} می‌شود که پيش از آن معرفی شده بود و به شدت شبیه به پايگاه‌هاي داده ساده و ابتدائي NoSQL است. اختراع مجدد چرخ هميشه لزوماً چيز بدی نیست. گاهی اوقات چرخ موجود واقعاً برای نيازهای شما مناسب نیست - همانطور که اين گفته منتبه به داگلاس کرافورد^{۲۵۰} بیان می‌کند: «چيز خوبی که در مورد اختراع مجدد چرخ است، این است که می‌توانید يك چرخ گرد برای خود بسازيد». ما اميدواريم که يادگيرندگان اين مطالب چند دقيقه بيشر و وقت بگذراند تا ديدگاه جامعه‌تری به دست آورند؛ در مورد اينکه چرا چيزهای مختلفی که مورد بحث قرار می‌گيرند آن‌گونه که الان هستند پديد آمده‌اند (و یا چرا جور دیگري نیستند). ما بر اين باوريم که اين نه تنها به شما کمک می‌کند تا تصميم بگيريد که آيا اختراع مجدد چرخ در حالت خاصي خوب است یا خير، بلکه به شما کمک می‌کند تا از دليستگي شديد به فناوري و پرستيدن آن اجتناب کنيد. يعني اينکه شما بخواهيد يك فناوري جديد و به نظر جذاب و محبوب را مهم و بالريش برای يادگيري تلقی کنيد صرفاً به دليل اينکه جديد (يا سريع يا سبك و يا هر چيز دیگري) است، بدون چشم‌اندازی مستدل و آگاهی از نقاط قوت و ضعف آن يا اينکه چگونه بر ايده‌هاي مشابهی که قبلًا بررسی شده‌اند بنا می‌شود.

⚠ خطر پنهان: تمركز پيش از حد بر يادگيري سريع يك چارچوب خاص

فناوري‌هاي ساخت و توسعه نرم‌افزار با چنان سرعتي در حال تغيير و تکامل هستند که حتی چهار سال زمان هم کافي است تا چيزی که در گذشته به عنوان «فناوري داغ و تازه» شناخته می‌شد اين عنوان را به چيزی جديفتر و اگذر کرده باشد. در الواقع، از اولين نسخه اين کتاب در سال ۲۰۱۳ «فناوري داغ» برای ساخت قسمت پيشين يك نرم‌افزار کاربردي از پروتوتايپ^{۲۵۱} به جي‌کوئري، سپس به آنگولار^{۲۵۲}، امبر جي اس^{۲۵۳}، بکبون جي اس^{۲۵۴} و ری‌اکت^{۲۵۵} تغيير کرده است، و اکنون ويو

²⁴²George Santayana

²⁴³Node.js

²⁴⁴Thread

²⁴⁵Event (Computing)

^{۲۴۶}اصطلاح «خرشن کاربرد» و یا شکل کلی‌تر آن «خرشن گستره»، به حالت اشاره دارد که يك پروژه در طول زمان با افزونی کاربردهای زیادي از گستره تعریف شده اولیه آن فراتر می‌رود و مدیریت آن تا حدی از کنترل خارج می‌شود. در مورد پروژه‌های نرم‌افزاری اين روند منجر به پيچيدگي پيش از حد نرم‌افزار و اصطلاحاً «نفح نرم‌افزار» می‌شود.

²⁴⁷Database

²⁴⁸Relational Model (Database Model)

²⁴⁹Hierarchical Model (Database Model)

²⁵⁰Douglas Crockford

²⁵¹Prototype (JS Web Framework)

²⁵²Angular (Web Framework)

²⁵³Ember.js (Web Framework)

²⁵⁴Backbone.js (Web Framework)

²⁵⁵React (JS Library)

چی اس^{۲۵۶} 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 سوال که کدام چارچوب یا زبان برنامه‌نویسی برای یک پروژه معین بهترین گزینه است.

۱۲-۲. 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 آمده‌اند؟ 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 بپاموزد؟ این قسمت‌ها هرگز حاوی محتواهایی 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 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 گفت پک^{۲۵۷} 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 بهجای 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 بنویسید.

- اگر بررسی و مرور کردن کد خوب است، پس به‌طور مداوم کد را مورد بررسی قرار دهید، بهصورت دو نفره برنامه‌نویسی کنید، به شکلی که دو نفر باهم پای یک رایانه بنشینند و بهصورت نوبتی کد یکدیگر را بررسی کنند.

- مایکل سوانی^{۲۵۸}، مصاحبه با گفت پک، (سواین ۲۰۰۱)

این نقل قول بهخوبی 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 پیش‌نمونه‌ای ناتمام از نرم افزار را که الیه کار می‌کند، در هر یک یا دو هفته یکبار ببیند. کد مربوط به آزمون‌ها را قبل از نوشتن 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 به‌طور مداوم مورد بازبینی و بررسی قرار می‌گیرد. روش چاپک تنها طی ده-دوازده سال از یک متدولوژی نرم افزاری نامتعارف (و مُرتد) فراتر رفته و تبدیل به مدل غالب توسعه نرم افزار شده است؛ وقتی این مدل با معماری 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 هم‌افزاری خوبی در بین آن‌ها وجود دارد. توسعه چاپک به معنی پیشرفت مستمر در حین همکاری نزدیک با مشتری است و SaaS بر روی یک بستر رایانش ابری به مشتری این اجازه را می‌دهد تا در هر زمان و بی‌درنگ از آخرین نسخه نرم افزار استفاده کند، در نتیجه چرخه بازخورد تنگ‌تر می‌شود و بازخورد سریع‌تر صورت می‌گیرد (به فصول ۷ و ۱۲ رجوع کنید). SaaS بر روی بستر رایانش ابری با الگوی طراحی مدل-نما-کنترل‌گر (به فصل ۱۱

²⁵⁶Vue.js (JS Web Framework)

²⁵⁷Kent Beck

²⁵⁸Michael 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) بر روی بستر رایانش ابری، (۲) توسعه چاپک و (۳) چارچوبها و ابزارهایی کارا و غنی.

رجوع کنید) که توسط چارچوب‌های غنی SaaS ارائه می‌شود، بسیار سازگار است (به فصول ۳، ۴ و ۵ رجوع کنید). ابزارها و چارچوب‌های کارا و غنی که برای پشتیبانی از توسعهٔ چابک طراحی شده‌اند، موافع را از سر راه عمل به مدل چاپک برمی‌دارند (به فصول ۷، ۸ و ۱۰ مراجعه کنید). ما بر این باوریم که این سه «جواهر درخشان» که «مثلث طلایی» را تشکیل می‌دهند و همچنین شالودهٔ 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 زمان‌بندی و بودجهٔ تعیین شدهٔ اولیه انجام می‌گیرد.

مثلث طلایی کمک می‌کند تا ماهیت نوآورانهٔ جامعهٔ کاربران ریلز را بهتر توضیح دهیم. جامعه‌ای که مرتباً ابزارهای جدیدی را می‌سازد به راحتی بهره‌وری را بهبود می‌بخشد. ما تقریباً مطمئن هستیم که نسخه‌های آیندهٔ این کتاب شامل ابزارهایی خواهد بود که هنوز به وجود نیامده‌اند و آن قدر مفید هستند که نمی‌توانیم تصور کنیم چطور زمانی کارمان را بدون آن‌ها پیش می‌بردیم!

به عنوان مدرس و از آنچایی که بسیاری از دانشجویان روش‌های طرح-و-ثبت را بسیار کسالت‌آور می‌دانند، ما از اینکه پاسخ‌ها به ۱۰ پرسش اشاره شده در شکل ۱-۵، به شدت استفاده از روش چاپک را برای انجام پروژه‌های دانشجویی توصیه می‌کند، 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 گزینهٔ بهتری باشد یا بعضی از مشتریان انجام کار را ملزم به استفاده از این روش می‌کنند و همچنین پرداختن به آن به توضیح بهتر روش چاپک کمک می‌کند. بنابراین در اوآخر هر فصل از بخش دوم کتاب، قسمتی را قرار داده‌ایم که مطالب مربوط به آن فصل را از منظر روش طرح-و-ثبت بیان می‌کند.

به عنوان پژوهشگر، ما معتقدیم که نرم افزارهای آینده به صورت روزافزون بر روی بستر زیرساخت‌های ابری ساخته و به آن‌ها وابسته می‌شوند. بنابراین از آنچایی که روش چاپک سازگاری بسیار خوبی با این بسترهای دارد، پیش‌بینی می‌کنیم که این روش در آینده بسیار محبوب‌تر خواهد شد. از این رو ما اکنون در نقطهٔ زمانی خوبی قرار داریم که آیندهٔ جذابی هم برای یادگیری و هم برای آموزش توسعهٔ نرم افزار در پیش رو داریم. چارچوب‌های کارا و غنی مانند ریلز به شما کمک می‌کنند تا در زمان کوتاهی و با انجام دادن، ارزش این تکنولوژی را درک کنید. هدف اصلی ما از نوشتند این کتاب این است که افراد بیشتری را از این فرصت استثنایی باخبر کنیم تا بتوانند از آن بهره‌مند 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. خوش‌های از رایانه‌های معمولی مدت‌ها پایهٔ اصلی SaaS بودند، اما رایانش ابری نحوهٔ استفاده از آن خوش‌های را تغییر داد. تا اواخر دههٔ ۱۹۹۰، معمول بود که یک رایانهٔ خاص به یک نرم افزار کاربردی SaaS خاص اختصاص داده شود که بر روی آن از پیش تمام پیش‌نیازها و اجزای نرم افزاری مورد نیاز برای اجرای آن نرم افزار کاربردی نصب شده باشد. در مقابل، از اواسط دههٔ ۲۰۰۰، فناوری **ماشین مجازی** این امکان را برای یک رایانهٔ فیزیکی فراهم کرد که بتواند وانمود کند چندین رایانه است، به طوری که نرم افزارهای در حال اجرا بر روی هر رایانهٔ مجازی باور داشتند که روی سخت‌افزار واقعی اجرا می‌شوند. مانند بسیاری دیگر از فناوری‌های مرتبط با SaaS، ماشین‌های مجازی برای دهه‌ها وجود داشتند—در این مورد، حداقل از دههٔ ۱۹۶۰—اما کاهش هزینه‌های سخت‌افزار و سلطه معماری اینتل^{۲۵۹} بر سرورها، ماشین‌های مجازی با کارایی بالا را به ابزاری کاربردی تبدیل کرد تا با استفاده از آن بتوان از چندین نرم افزار کاربردی SaaS مختلف بر روی یک رایانه میزبانی کرد، حتی اگر آن‌ها به سیستم‌های عامل و بسته‌های نرم افزاری متفاوتی نیاز داشته باشند. به طور معمول، یک نرم افزار کاربردی SaaS فقط به نوع ماشین مجازی که در آن اجرا می‌شود اهمیت می‌دهد و می‌تواند تا حد زیادی از جزئیات سخت‌افزار و سیستم عاملی که ماشین مجازی روی آن میزبانی می‌شود، بی‌اطلاع بماند. از اواسط دههٔ ۲۰۰۰، تکامل بیشتر فناوری ماشین مجازی منجر به چارچوب‌های **کانتینری** سُبُکی مانند داکر شد که بسته‌های نرم افزاری را از یکدیگر جدا و ایزوله می‌کند در حالی که یک هستهٔ^{۲۶۰} سیستم عامل واحدی را به اشتراک می‌گذارد. جدیدترین مرحله از مجازی‌سازی، تابع به صورت یک سرویس FaaS^{۲۶۱} است، که در آن توسعه‌دهنده اکنون

²⁵⁹Intel²⁶⁰Kernel (Operating System)²⁶¹Function as a Service

فقط کد یک یا چند تابع را مشخص می‌کند و به ازای فراخوانی^{۲۶۲} تابع هزینه پرداخت می‌کند. نمونه اولیه این مدل از مجازی‌سازی، سرویس لامبда^{۲۶۳} از سرویس‌های وب آمازون^{۲۶۴} است. اگرچه از FaaS با عنوان «رایانش بی‌سرور»^{۲۶۵} نیز یاد می‌شود، البته واقعاً بدون سرور نیست، زیرا توابع باید به هر حال در جایی اجرا شوند. تمایز اصلی آن با SaaS این است که توسعه‌دهندگان با یک پُشته نرم‌افزاری^{۲۶۶} متشکل از سرور نرم‌افزار کاربردی، سرور HTTP^{۲۶۷} و غیره سروکار ندارند؛ آن‌ها فقط کد توابع را می‌نویسند. رایانش بی‌سرور هنوز در حال تکامل است و بسته به نوع نرم‌افزار کاربردی ای که قرار است پیاده‌سازی و به کاراندازی شود، هم مزایا و هم معایب دارد (کاسترو و دیگران ۲۰۱۹).

ما بر این باوریم که اگر مطالب این کتاب را یاد بگیرید و در کنار آن تکالیف و فعالیت‌های پیشنهادی (پُفک‌ها) را انجام دهید، همانطور که روش‌های معقول و اصولی مهندسی نرم‌افزار را می‌آموزید و آن‌ها را در دنبال می‌کنید، می‌توانید نسخه‌های (ساده‌شده) خودتان از سرویس‌های نرم‌افزاری محبوبی مانند فارم‌ویل^{۲۶۸} یا توییتر را بسازید. اگرچه توانایی ساخت سرویس‌های شبیه به سرویس‌های موفق 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 ببینیم 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.

Barry W. Boehm. A spiral model of software development and enhancement. in ACM SIGSOFT Software Engineering Notes, 1986.

²⁶²Call (Function/Method)

²⁶³Lambda (AWS)

²⁶⁴Serverless Computing

²⁶⁵Software Stack

²⁶⁶Hypertext Transfer Protocol

²⁶⁷FarmVille (Online Game)

در زبان برنامه‌نویسی محترم و ارجمند لیسب، توابع را اصطلاحاً لامبدا می‌نامیدند، زیرا این زبان به شدت از دستگاه‌های صوری حساب لامبدا الهام گرفته شده بود.

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.

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.

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.

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.

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.

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#Guides۱
http://www.youtube.com/watch?v=DeBi2ZxUZiM۲
http://www.youtube.com/watch?v=kYUrqdUyEpI۳
https://youtu.be/AP71VJphbSk۴
https://getbootstrap.com/docs/4.0/examples/navbars/۵
https://sensortower.com/blog/25-top-ios-apps-and-their-version-update-frequencies۶
https://www.w3.org/TR/2011/WD-html5-20110525/offline.html۷
http://developers.slashdot.org/story/08/05/11/1759213/c۸
https://aws.amazon.com/lambda۹

```

بخش اول

نرم افزار به صورت یک سرویس: چارچوبها و زبانها

چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب [مراجعه کنید](#).

پیوندها

<https://saasbook.info/translation-fa>[†]

ریزسرویس‌ها، API‌ها و REST

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب^۱ مراجعه کنید.

پیوندها

[https://saasbook.info/translation-fa^۱](https://saasbook.info/translation-fa)

ریلز، یک چارچوب مدل-نما-کنترل‌گر

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب^۱ مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^۱

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^۱ مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^۱

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^۱ مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^۱

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^۱ مراجعه کنید.

پیوندها

[https://saasbook.info/translation-fa^۱](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^۱ مراجعه کنید.

پیوندها

[https://saasbook.info/translation-fa^۱](https://saasbook.info/translation-fa)

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^۱ مراجعه کنید.

پیوندها

[https://saasbook.info/translation-fa^۱](https://saasbook.info/translation-fa)

تیم‌های چابک

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزسازی‌های آتی به وبگاه کتاب^{۱۰} مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^{۱۰}

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^۱ مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^۱

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^۱ مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa>^۱

پس‌گفتار

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب^{۱۰} مراجعه کنید.

پیوندها

[https://saasbook.info/translation-fa^{۱۱}](https://saasbook.info/translation-fa)

واژه‌نامهٔ انگلیسی به فارسی

A

Abstraction (Computer Science)	تجزید (علوم رایانه)
Acceptance Testing	آزمون پذیرش
Accessibility	دسترسی‌پذیری
Active Server Pages (ASP.NET)	صفحه‌های سرور فعال
Affordable Care Act	لایحهٔ مراقبت مقررین به صرفه
Agile	چاپک
Agile Alliance	اتحادیهٔ روش چاپک
Agile Manifesto	منشور چاپک
Authentication	احراز هویت
Automation	خودکارسازی
Availability	دسترسی‌پذیری
Amazon Web Services (AWS)	سرویس‌های وب آمازون
Amazon.com	آمازون (شرکت حوزهٔ فناوری)
Andy Hunt	آندری هانت
Android (OS)	اندروید
Angular (Web Framework)	آنگولار
AOL	ای اوال
App Store	فروشگاه نرم‌افزار کاربردی
Apple	اپل
Application (Software)	نرم‌افزار کاربردی
Application Programming Interface	واسط برنامه‌نویسی
Aqueduct of Segovia	آب‌گذر سگوویا
Argument (Programming)	آرگومان (برنامه‌نویسی)
Ariane 4 Rocket	موشک آریان ۴
Ariane 5 Rocket	موشک آریان ۵
Armando Fox	آرماندو فاکس
Assembly Language	زبان اسیمبلی
Assertion (Computer Programming)	بايسنتگي (برنامه‌نویسی)
Attribute	خصوصیت

B

Backbone.js (Web Framework)	بکبون جی اس
Backend	پسین
Beautiful Code	زیباگد
Behavior-Driven Development	توسعه رفتارمحور
Bug (Computer)	اشکال
Buzzword	واژه باب روز
Business Modeling	مدل سازی کسب و کار
Big Design Up Front	طراحی عظیم از ابتدا
Bit Blit	بیت بلیت
Bitmap	بیت‌مپ
Bookmark (Web)	نشانک
Bootstrap (Front-end Framework)	بوتاسترب
British Computer Society	انجمن رایانه بریتانیا
Browser (Web)	مرورگر

C

C (Programming Language)	سی
C# (Programming Language)	سی شارپ
C++ (Programming Language)	سی پلاس پلاس
Call (Function/Method)	فراخوانی
Capability Maturity Model	مدل بلوغ قابلیت
Carnegie Mellon University	دانشگاه کارنگی ملون
Cascading Style Sheets	شیوه‌نامه آشماری
CGI Group (Information Technology Company)	شرکت فناوری گروه سی جی آی
Cucumber (Software tool)	کیوکامبر (ابزاری برای توسعه رفتارمحور)
Character (Computing)	نویسه (رایانش)
Clarity	وضوح
Class (Computer Programming)	کلاس (برنامه‌نویسی)
Client	کلاینت
Cloud Computing	رایانش ابری
COBOL (Programming Language)	کوبال (زبان برنامه‌نویسی)
Codebase	مجموعه کد
Coding/Hands-On Integrated Projects	پروژه‌ها و فعالیتهای عملی/کدنویسی (CHIPS)
Codio	گدیو
Coverage (Testing)	پوشش
CompuServe	کامپیوسرو
Computer	رایانه
Computer Cluster	خوشه
Computer Programming	برنامه‌نویسی رایانه‌ای
Computing	رایانش
Compiler	کامپایلر
Conciseness	اختصار

Confucius	کنفوسیوس
Configuration	پیکربندی
Construction (RUP phase)	ساخت (فازی از آریوپی)
Container (Virtualization)	کانتینر
Content-Oriented Didactic (COD)	آموزشی محتوامحور
Cray	کری

D

Dave Thomas	دیو توماس
David Patterson	دیوید پترسون
Database	پایگاه داده
Datacenter	مرکز داده
Debugging	اشکال‌زدایی
Develop (Software)	توسعه
Developer (Software)	توسعه‌دهنده
Deployment (Software)	به‌کاراندازی
Design Pattern	الگوی طراحی
Desktop	دسکتاپ
Dynamic Typing (Type Checking)	نوع‌دهی پویا
Django	جنگو
Document (Computing)	سند
Document Type Declaration (XML Instruction)	اعلان نوع سند
Documentation	مستندسازی
Docker	دایکر
Douglas Crockford	دالگلاس کراکفرد
Download	بارگیری
Donald Knuth	دانلد کنوث
Don't Repeat Yourself (DRY)	خودت را تکرار مکن

E

eBay	ای‌بی
Economies of Scale	صرفه به مقیاس
edX	ای‌دکس
Eudora (Email Client)	یودارا
Event (Computing)	رویداد
Exception	استثنای
Extensible Markup Language	زبان نشانه‌گذاری گسترش‌پذیر
Extreme Programming	برنامه‌نویسی مفرط
Elaboration (RUP phase)	تحلیل جزئیات (فازی از آریوپی)
Elastic Compute Cloud (EC2)	سرویس رایانش ابری ارجاعی آمازون
Element	عنصر

Email	رایانame
Ember.js (Web Framework)	امبر جی اس
Empire State Building	ساختمان امپیر استیت
Encoding (Computing)	کدگذاری
Enterprise Java Beans (EJB)	بین‌های سازمانی جاوا
Eric Braude	اریک براد
Error (Computing)	خطا
Ethernet	ایترنط

F

Facebook	فیسبوک
FarmVille (Online Game)	فارم ویل
Feature Creep	خرش کاربرد
Function (Programming)	تابع (برنامه‌نویسی)
Function as a Service	تابع به صورت یک سرویس
Functional Programming	برنامه‌نویسی تابعی
File	فایل
Floating Point	ممیز شناور
Formal Methods	روش‌های صوری
Formal System	دستگاه صوری
Fortran (Programming Language)	فورترن (زبان برنامه‌نویسی)
Framework	چارچوب
Fred Brooks, Jr.	فرد بروکس جونیور
Frontend	پیشین

G

George Santayana	جورج سانتایانا
Gerald Weinberg	جرالد واینبرگ
Guard (Testing tool)	گارد
Git (Version Control System)	گیت
GitHub	گیت‌هاب
Gmail (Google Mail Service)	جي ميل
Google	گوگل
Google App Engine	موتور اجرای برنامه گوگل
Google Calendar (Digital Calendar Service)	تقویم گوگل
Google Docs (Online Word Processor)	گوگل داکس
Google Maps (Web Mapping Service)	گوگل مپز
Google News (News Aggregator Service)	گوگل نیوز
Google Search	جستجوگر گوگل
Grace Murray Hopper	گریس موری هاپر

H

Hacker	هکر
Hardware	سخت‌افزار
HyperText Markup Language	زبان نشانه‌گذاری ابرمتنی
Hierarchical Model (Database Model)	مدل سلسله‌مراتبی
Higher-order Function	تابع مرتبه بالاتر
Hotmail	هات‌میل

I

IBM	آی‌بی‌ام
IBM's Generalized Markup Language	زبان نشانه‌گذاری تعمیم‌یافته آی‌بی‌ام
Icon (Computing)	آیکن
Inception (RUP phase)	آغازین
Information Technology	فناوری اطلاعات
Inheritance (OOP)	ارثبری
Input/Output (I/O)	ورودی/خروجی
Integrated Development Environment	محیط توسعهٔ یکپارچه
Integration	یکپارچه‌سازی
Integration Testing	آزمون یکپارچگی
Intel	اینتل
Interactive	تعاملی
Interface (Computing)	واسط (رایانش)
Interpreter	تفسر
iOS	آی‌اواس
iPhone	آیفون
Iteration	تکرار

J

Java (Programming Language)	جاوا (زبان برنامه‌نویسی)
JavaScipt	جاوا‌اسکریپت
JavaServer Pages (Jakarta Server Pages)	صفحات سرور جاکارتا
Jay Leno	چی لنو (کمدین)
Jasmine (JavaScript Testing Tool)	چَزمین
Just-in-time	بهنگام
John McCarthy	جان مک‌کارتی
jQuery	چی‌کوئری

K

Kayak (Travel Search Engine)	کایاک
Kanban	کانبان
Kent Beck	کنت بک
Kernel (Operating System)	هسته
Kindle	کیندل

L

Lambda (AWS)	لامبدا
Lambda Calculus	حساب لامبدا
Laptop	لپ‌تاپ
Legacy Code	میراث‌کد
Luiz Barroso	لوئیز باروسو
Library (Computing)	کتابخانه (رایانش)
Life Cycle	چرخه حیات
Link (Web Hyperlink)	پیوند
Lisp (Programming Language)	لیسپ
Literate Programming	برنامه‌نویسی ادبیانه

M

Mainframe Computer	بزرگ‌رایانه
Maintenance	نگهداری
Makefile	میکافایل
Marcus Vitruvius Pollio	مارکوس ویتروویوس پولیو
Markup Language	زبان نشانه‌گذاری
Mars Climate Orbiter	مدارگرد اقلیمی مریخ
Massachusetts Institute of Technology	مؤسسه فناوری ماساچوست
Metaprogramming	فرا برنامه‌نویسی
Method (OOP)	مِتُد
Michael Swaine	مایکل سوین
Microservice	ریزسرویس
Microsoft	مايكروسافت
Microsoft Azure	مايكروسافت آزور
Microsoft Word	مايكروسافت ورد
Microsoft's .NET Framework	چارچوب دات‌نیت شرکت مايكروسافت
Mix-in (OOP)	میکس‌این
MOCAS	موکاس
Model-View-Controller (Software Architecture)	مدل-نما-کنترل‌گر
Module Testing	آزمون مازول
Mona Lisa (Leonardo da Vinci's Painting)	مونا لیزا
Moore's Law	قانون مور

Morris Udall موریس اودال

N

Node.js نود جی اس

O

Object (Programming)	شیء (برنامه‌نویسی)
Objective-C (Programming Language)	آجکتیو-سی
Object-oriented (Programming)	شیءگرا (برنامه‌نویسی)
Overhead	سریار
Online	آنلاین
Open Source (Software)	متن باز
Operating System	سیستم عامل

P

Package (Program Distribution)	بسته
Pay-as-you-go	پرداخت به میزان مصرف
Pair Programming	برنامه‌نویسی دونفره
Performance	کارایی
Perl (Programming Language)	پرل (زبان برنامه‌نویسی)
Personal Computer	رایانه شخصی
Python (Programming Language)	پایتون
PHP (Programming Language)	پیاج بی (زبان برنامه‌نویسی)
Pivotal Tracker	پیووتال ترکر (ابزار مدیریت پروژه)
Pipeline	خط لوله
Plan-and-Document	طرح و ثبت
Platform (Computing)	سکو
Procedure (Programming)	رَوِیه
Process	فرآیند
Processor (Computing)	پردازنده
Program (Computer)	برنامه
Programmer	برنامه‌نویس
Programming	برنامه‌نویسی
Programming Language	زبان برنامه‌نویسی
Programming Paradigm	پارادایم برنامه‌نویسی
Progressive Web Application	برنامه و ب پیشرو
Prototype	پیش‌نمونه
Prototype (JS Web Framework)	پروتوتاپ

Q

Quality Assurance (QA) تضمین کیفیت

R

Rails (Simplified verion of Ruby on Rails) ریلز (صورت ساده شده روی آن ریلز)
 Rational Unified Process فرایند یکپارچه رشنال
 React (JS Library) ریاکت
 Real-time بلدرنگ
 Redundancy افزونگی
 Refactoring (Code) بازسازی
 Reflection (Computer Programming) بازتاب
 Reusability بازکاربرد پذیری
 Reuse بازکاربرد
 Relational Model (Database Model) مدل رابطه‌ای
 Reliability قابلیت اطمینان
 Remote (Networking) دوردست
 Render (Computing) رندر
 Renee McCauley رنی مک کاولی
 Responsive (Web Design) واکنش‌گرا
 Responsive Web Design طراحی وب واکنش‌گرا
 RESTful RESTful مبتنی بر
 Ruby (Programming Language) روی
 Ruby On Rails روی آن ریلز
 Runtime System سامانه زمان اجرا
 Robert Laganiere رابرت لaganier
 Rolling Stones گروه موسیقی رولینگ استونز
 RSpec (Testing tool) آر-اسپیک

S

Salesforce سیلفرورس (شرکت حوزه فناوری)
 Scalability مقیاس پذیری
 Scalable مقیاس پذیر
 Scale مقیاس
 Scope Creep خوش گستره
 Scrum اسکرام
 ScrumBan اسکرامبان
 Scripting Language زبان اسکریپت‌نویسی

Search Engine	موتور جستجو
Selector (CSS)	انتخابگر
Self-Organizing (Team Management)	خدسازمانده
Server	سرور
Serverless Computing	رایانش بی‌سرور
Service-oriented Architecture	معماری سرویس‌گرا
Serial	ترتیبی
Supercomputer	آبرایانه
Switch (Networking)	سوئیچ
Synthesis	سنتر
System	سامانه
System Testing	آزمون سامانه
Shimon Peres	شیمون پرز
SimpleCov (Testing tool)	سیمپل‌کاو
Sinatra	سیناترا
Smartphone	گوشی هوشمند
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	نفح نرم‌افزار
Software Development	توسعهٔ نرم‌افزار
Software Engineering	مهندسی نرم‌افزار
Software Stack	پُشنجهٔ نرم‌افزاری
Specification	مشخصه
Spiral Model	مدل مارپیچ
Spring	اسپرینگ
Sprint	اسپرینت
Stakeholder	ذی‌نفع
Standard Generalized Markup Language	زبان نشانه‌گذاری تعمیم‌یافتهٔ استاندارد
Static	ایستا
Steven Ratkin	استیون راتکین
Stylesheet (Web Development)	شیوه‌نامه

T

Tablet	تبلت
Tag	تگ
Technology	فناوری
Test	آزمون
Test Case	آزمایه
Testable	آزمون‌پذیر
Test-Driven Development	توسعهٔ آزمون محور
Tester	آزمون‌گر
Testing	آزمون

Turing Award	جایزهٔ تورینگ
Twitter	توییتر
Typography	تایپوگرافی
Therac-25 (radiation therapy machine)	ماشین پرتو درمانی تراک-۲۵
Thread	ریسمان
Time-sharing	اشتراک زمانی
Timothy Lethbridge	تیموقی لیثبریج
Toyota (Company)	توبیوتا
Transition (RUP phase)	گذار

U

Unified Modeling Language (UML)	زبان مدل‌سازی یکپارچه
University of Nebraska	دانشگاه نبراسکا
Unit Testing	آزمون واحد
US National Academies	آکادمی‌های ملی آمریکا
Use case	مورد استفاده
User Interface	رابط کاربری
User Story	روایت کاربری
USS Hopper	یواس‌اس هاپر
Utilization	بهره‌وری

V

Validation	اعتبارسنجی
Velocity (Software Development Metric)	سرعت
Verification	درستی‌سنجی
Version Control	کنترل نسخه
Vue.js (JS Web Framework)	ویو جی اس
Virtual Case File (FBI Software)	نرم‌افزار پرونده مجازی افبی‌آی
Virtual Machine	ماشین مجازی
Virtual Machine Monitor	ناظر ماشین مجازی
Virtualization	مجازی‌سازی

W

Warehouse Scale Computer	رایانه انبارگون
Waterfall Model	مدل آبشاری
Web (World Wide Web)	وب
Web Portal	درگاه وب
Webpage	صفحه وب

Website	وبگاه.....
Weinberg	وینبرگ.....
Wikimedia Commons	ویکی‌انبار.....
Wikipedia	ویکی‌پدیا.....
Word Processor (Software)	واژه‌پرداز (نرم‌افزار).....
Workflow	گردش کار.....

Y

Yahoo!	یاهو!.....
YouTube (Online Video Sharing Platform)	یوتیوب.....

Z

Zend	زند.....
------------	----------

واژه‌نامهٔ فارسی به انگلیسی

آ

Objective-C (Programming Language)	آبجکتیو-سی
Aqueduct of Segovia	آب‌گذر سگوویا
RSpec (Testing tool)	آر-اسپیک
Argument (Programming)	آرگومان (برنامه‌نویسی)
Armando Fox	آرماندو فاکس
Test Case	آزمایه
Test	آزمون
Acceptance Testing	آزمون پذیرش
System Testing	آزمون سامانه
Module Testing	آزمون ماذول
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
Icon (Computing)	آیکن

۱

Supercomputer	آبرایانه
Apple	اپل
Agile Alliance	اتحادیهٔ روش چاپک
Authentication	احراز هویت

Conciseness	اختصار.....
edX	ایدکس.....
Inheritance (OOP)	ارثبری.....
Eric Braude	اریک براود.....
Sprint	اسپرینت.....
Spring	اسپرینگ.....
Exception	استثناء.....
Steven Ratkin	ستفان راتکین.....
Scrum	اسکرام.....
ScrumBan	اسکرامبان.....
Time-sharing	اشتراك زمانی.....
Bug (Computer)	اشکال.....
Debugging	اشکال‌زدایی.....
Validation	اعتبارسنجی.....
Document Type Declaration (XML Instruction)	اعلان نوع سند.....
Redundancy	افزوونگی.....
Design Pattern	الگوی طراحی.....
Ember.js (Web Framework)	امبر جی اس.....
Selector (CSS)	انتخابگر.....
British Computer Society	انجمن رایانه بریتانیا.....
Android (OS)	اندروید.....
Andy Hunt	اندی هانت.....
Angular (Web Framework)	انگولار.....
AOL	ای او ال.....
eBay	ای بی.....
Ethernet	ایترنوت.....
Static	ایستا.....
Intel	اینتل.....

ب

Download	بارگیری.....
Reflection (Computer Programming)	بازتاب.....
Refactoring (Code)	بازسازی.....
Reuse	بازکاربرد.....
Reusability	بازکاربردپذیری.....
Assertion (Computer Programming)	بايستگی (برنامه‌نویسی).....
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 مفطر
Mainframe Computer	بزرگ‌رایانه
Package (Program Distribution)	بسته
Backbone.js (Web Framework)	بک‌بون جی‌اس
Real-time	بلادرنگ
Bootstrap (Front-end Framework)	بوت‌استرپ
Utilization	بهره‌وری
Deployment (Software)	به‌کاراندازی
Just-in-time	بهنگام
Bit Blit	بیت‌بلیت
Bitmap	بیت‌مپ
Enterprise Java Beans (EJB)	بین‌های سازمانی جاوا

پ

Programming Paradigm	پارادایم برنامه‌نویسی
Python (Programming Language)	پایتون
Database	پایگاه داده
Pay-as-you-go	پرداخت به میزان مصرف
Processor (Computing)	پردازنده
Perl (Programming Language)	پرل (زبان برنامه‌نویسی)
Prototype (JS Web Framework)	پروتوتایپ
Coding/Hands-On Integrated Projects	پیوژه‌ها و فعالیتهای عملی/کدنویسی (CHIPS)
Backend	پسین
Software Stack	پُشنئَ نرم‌افزاری
Coverage (Testing)	پوشش
PHP (Programming Language)	پیاج‌پی (زبان برنامه‌نویسی)
Prototype	پیش‌نمونه
Frontend	پیشین
Configuration	پیکربندی
Link (Web Hyperlink)	پیوند
Pivotal Tracker	پیووتال ترکر (ابزار مدیریت پروژه)

ت

Function (Programming)	تابع (برنامه‌نویسی)
Function as a Service	تابع به صورت یک سرویس
Higher-order Function	تابع مرتبه بالاتر
Typography	تایپوگرافی
Tablet	تبلت
Abstraction (Computer Science)	تجزیید (علوم رایانه)
Elaboration (RUP phase)	تحلیل جزئیات (فازی از آریوپی)

Serial	ترتبی
Quality Assurance (QA)	تضمین کیفیت
Interactive	تعاملی
Google Calendar (Digital Calendar Service)	تقویم گوگل
Iteration	تکرار
Tag	تگ
Develop (Software)	توسعه
Test-Driven Development	توسعه آزمون محور
Behavior-Driven Development	توسعه رفتار محور
Software Development	توسعه نرم افزار
Developer (Software)	توسعه دهنده
Toyota (Company)	توبیوتا
Twitter	توبیتر
Timothy Lethbridge	تیموتی لیثبریج

ج

John McCarthy	جان مک‌کارتی
Java (Programming Language)	جاوا (زبان برنامه‌نویسی)
JavaScript	جاوا اسکریپت
Turing Award	جایزه تورینگ
Gerald Weinberg	جرالد واینبرگ
Jasmine (JavaScript Testing Tool)	جَزَمِین
Google Search	جستجوگر گوگل
Django	جنگو
George Santayana	جورج سانتایانا
Jay Leno	چی لنو (کمدین)
jQuery	چی کوئری
Gmail (Google Mail Service)	چی میل

چ

Agile	چارچک
Framework	چارچوب
Microsoft's .NET Framework	چارچوب دات نت شرکت ماکروسافت
Life Cycle	چرخهٔ حیات

ح

Lambda Calculus	حساب لامبدا
-----------------------	-------------

خ

Feature Creep	خرش کاربرد
Scope Creep	خرش گستره
Attribute	خصوصیت
Pipeline	خط لوله
Error (Computing)	خطا
Don't Repeat Yourself (DRY)	خودت را تکرار مکن
Self-Organizing (Team Management)	خودسازمانده
Automation	خودکارسازی
Computer Cluster	خوشه

د

Docker	داکر
Douglas Crockford	داگلاس کراکفرد
Carnegie Mellon University	دانشگاه کارنگی ملون
University of Nebraska	دانشگاه نبراسکا
Donald Knuth	دانلد کنوث
Verification	درستی‌سنجی
Web Portal	درگاه وب
Availability	دسترسی‌پذیری
Formal System	دستگاه صوری
Desktop	دسکتاپ
Remote (Networking)	دوردست
Dave Thomas	دیو توماس
David Patterson	دیوید پترسون

ذ

Stakeholder	ذینفع
-------------------	-------

ر

Robert Laganiere	رابرت لaganier
User Interface	رابط کاربری
Email	رایانمہ
Computing	رایانش
Cloud Computing	رایانش ابری
Serverless Computing	رایانش بی‌سرور
Computer	رایانه

Warehouse Scale Computer	رايانه انبارگون
Personal Computer	رايانه شخصی
Render (Computing)	رندر
Renee McCauley	رینی مک کاولی
User Story	روایت کاربری
Ruby (Programming Language)	روبی
Ruby On Rails	روبی آن ریلز
Formal Methods	روش‌های صوری
Event (Computing)	رویداد
Procedure (Programming)	رویه
React (JS Library)	ری‌اکت
Microservice	ریزسرویس
Thread	ریسمان
Rails (Simplified verion of Ruby on Rails)	ریلز (صورت ساده شده روبی آن ریلز)

ز

Scripting Language	زبان اسکریپت‌نویسی
Assembly Language	زبان آسمبلی
Programming Language	زبان برنامه‌نویسی
Unified Modeling Language (UML)	زبان مدل‌سازی یکپارچه
Markup Language	زبان نشانه‌گذاری
HyperText Markup Language	زبان نشانه‌گذاری ابرمتنی
IBM's Generalized Markup Language	زبان نشانه‌گذاری تعمیم‌یافته آی‌بی‌ام
Standard Generalized Markup Language	زبان نشانه‌گذاری تعمیم‌یافته استاندارد
Extensible Markup Language	زبان نشانه‌گذاری گسترش‌پذیر
Zend	زند
Beautiful Code	زیباکد

س

Construction (RUP phase)	ساخت (فازی از آریوپی)
Empire State Building	ساختمان امپایر استیت
System	سامانه
Runtime System	سامانه زمان اجرا
Hardware	سخت‌افزار
Overhead	سر بر
Velocity (Software Development Metric)	سرعت
Server	سرور
Elastic Compute Cloud (EC2)	سرویس رایانش ابری ارجاعی آمازون
Amazon Web Services (AWS)	سرویس‌های وب آمازون
Platform (Computing)	سکو
Synthesis	سنتر

Document (Computing)	سند
Switch (Networking)	سوئیچ
C (Programming Language)	سی
C# (Programming Language)	سی شارپ
C++ (Programming Language)	سی پلاس پلاس
Operating System	سیستم عامل
Salesforce	سینزفورس (شرکت حوزهٔ فناوری)
SimpleCov (Testing tool)	سیمپل کاو.
Sinatra	سیناترا

ش

CGI Group (Information Technology Company)	شرکت فناوری گروه سی جی آی
Object (Programming)	شیء (برنامه‌نویسی)
Object-oriented (Programming)	شیء‌گرا (برنامه‌نویسی)
Shimon Peres	شیمون پرز
Stylesheet (Web Development)	شیوه‌نامه
Cascading Style Sheets	شیوه‌نامه آبشاری

ص

Economies of Scale	صرفه به مقیاس
JavaServer Pages (Jakarta Server Pages)	صفحات سرور جاکارتا
Webpage	صفحه وب
Active Server Pages (ASP.NET)	صفحه‌های سرور فعال

ط

Big Design Up Front	طراحی عظیم از ابتدا
Responsive Web Design	طراحی وب واکشن‌گرا
Plan-and-Document	طرح-و-ثبت

ع

Element	عنصر
---------------	------

ف

FarmVille (Online Game)	فَارْمَوِيل
File	فَایل
Process	فَرآیند
Metaprogramming	فَراپرَنَامَهْ نُوِيْسِي
Call (Function/Method)	فَرَاخَوَانِي
Rational Unified Process	فَرَائِينَدِ يَكْبَارْچَه رِشَنَال
Fred Brooks, Jr.	فِرْدِ بُروْكْس جُونِيُور
App Store	فَناوِرِي
Technology	فَناوِرِي اطْلَاعَات
Information Technology	فَناوِرِي تِكنُولُوْجِي
Fortran (Programming Language)	فُورْتَرن (زِيانِ برنَامَهْ نُويْسِي)
Facebook	فِيسيُوك

ق

Reliability	قابلیت اطمینان
Moore's Law	قانون مور

ک

Performance	کاراِي
Compiler	کامپایلر
CompuServe	کامپیوسرُو
Kanban	کانْ باَن
Container (Virtualization)	کانتِينر
Kayak (Travel Search Engine)	کایاك
Library (Computing)	كتابخانه (رایانش)
Encoding (Computing)	کدگذاری
Codio	گُدو
Cray	کرَي
Class (Computer Programming)	کلاس (برنَامَهْ نُويْسِي)
Client	کلَائِينْت
Kent Beck	کِنْت بِك
Version Control	کنترل نسخه
Confucius	کنفوسيوس
COBOL (Programming Language)	کوبال (زِيانِ برنَامَهْ نُويْسِي)
Kindle	کيندل
Cucumber (Software tool)	کيوکامبر (ابزارِي برای توسعهٔ رفتارمحور)

گ

Guard (Testing tool)	گارد
Transition (RUP phase)	گذار
Workflow	گردش کار
Rolling Stones	گروه موسیقی رولینگ استونز
Grace Murray Hopper	گریس موری هاپر
Smartphone	گوشی هوشمند
Google	گوگل
Google Docs (Online Word Processor)	گوگل داکس
Google Maps (Web Mapping Service)	گوگل مَیز
Google News (News Aggregator Service)	گوگل نیوز
Git (Version Control System)	گیت
GitHub	گیت‌هاب

ل

Lambda (AWS)	لامبدا
Affordable Care Act	لایحهٔ مراقبت مفرون به صرفه
Laptop	لپ‌تاپ
Luiz Barroso	لوئیز باروسو
Lisp (Programming Language)	لیسپ

م

Marcus Vitruvius Pollio	مارکوس ویتروویوس پولیو
Therac-25 (radiation therapy machine)	ماشین پرتو درمانی تراک-۲۵
Virtual Machine	ماشین مجازی
Microsoft	مایکروسافت
Microsoft Azure	مایکروسافت آزور
Microsoft Word	مایکروسافت وُرد
Michael Swaine	مایکل سوانین
RESTful	REST مبتنی بر
Method (OOP)	مُنْد
Open Source (Software)	متن باز
Virtualization	ماجاری‌سازی
Codebase	مجموعهٔ کد
Integrated Development Environment	محیط توسعهٔ یکپارچه
Mars Climate Orbiter	مدارگرد اقلیمی مریخ
Waterfall Model	مدل آبشاری
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 کسب و کار
Model-View-Controller (Software Architecture)	مدل-نما-کنترلر گر
Datacenter	مرکز داده
Browser (Web)	مروگر
Documentation	مستندسازی
Specification	مشخصه
Service-oriented Architecture	معماری سرویس‌گرا
Software Architecture	معماری نرم‌افزار
Interpreter	مفسر
Scale	مقیاس
Scalable	مقیاس‌پذیر
Scalability	مقیاس‌پذیری
Floating Point	ممیز شناور
Agile Manifesto	منشور چاپک
Google App Engine	موتور اجرای برنامه گوگل
Search Engine	موتور جست‌وجو
Use case	مورد استفاده
Morris Udall	موریس اودال
Massachusetts Institute of Technology	مؤسسه فناوری ماساچوست
Ariane 4 Rocket	موشک آریان ۴
Ariane 5 Rocket	موشک آریان ۵
MOCAS	موکاس
Mona Lisa (Leonardo da Vinci's Painting)	مونا لیزا
Software Engineering	مهندسی نرم‌افزار
Legacy Code	میراث‌کد
Mix-in (OOP)	میکس‌این
Makefile	میک‌فایل

ن

Virtual Machine Monitor	ناظر ماشین‌های مجازی
Software	نرم‌افزار
Software as a Service	نرم‌افزار به صورت یک سرویس
Software as a Product	نرم‌افزار به صورت یک محصول
Virtual Case File (FBI Software)	نرم‌افزار پروندهٔ مجازی افبی‌آی
Application (Software)	نرم‌افزار کاربردی
Bookmark (Web)	نشانک
Software Bloat	نفح نرم‌افزار
Maintenance	نگهداری
Node.js	نود جی‌اس
Dynamic Typing (Type Checking)	نوع‌دهی پویا
Character (Computing)	نویسه (رایانش)

۹

Buzzword	واژهٔ باب روز
Word Processor (Software)	واژه‌پرداز (نرم‌افزار)
Interface (Computing)	واسطهٔ (رایانش)
Application Programming Interface	واسطهٔ برنامه‌نویسی
Responsive (Web Design)	واکنش‌گرا
Weinberg	واینبرگ
Web (World Wide Web)	وب
Website	وبگاه
Input/Output (I/O)	ورودی/خروجی
Clarity	وضوح
Wikimedia Commons	ویکی‌انبار
Wikipedia	ویکی‌پدیا
Vue.js (JS Web Framework)	ویو جی اس

۵

Hotmail	هات‌میل
Kernel (Operating System)	هسته
Hacker	هکر

۵

Yahoo!	یاهو!
Integration	یکپارچه‌سازی
USS Hopper	بیواس‌اس هاپر
YouTube (Online Video Sharing Platform)	یوتیوب
Eudora (Email Client)	بودارا

فهرست اختصارات

A

ACA Affordable Care Act

B

BDD Behavior-Driven Development

C

COD Content-Oriented Didactic
CSS Cascading Style Sheets

D

DOM Document Object Model

F

FaaS Function as a Service

H

HTML HyperText Markup Language
HTTP Hypertext Transfer Protocol

R

RUP Rational Unified Process

S

SaaS Software as a Service
SaaP Software as a Product
SOA Service Oriented Architecture

T

TDD Test-Driven Development

U

UML Unified Modeling Language

X

XML Extensible Markup Language
XP Extreme Programming