مقدمه

این کتاب به شما کمک خواهد کرد تا به برنامه‌نویس قابل‌تری تبدیل شوید.

اینکه شما یک توسعه‌دهنده‌ی مجرد، عضوی از یک تیم پروژه‌‌ای بسیار بزرگ، یا مشاوری که با کلاینت‌های بسیار زیادی به طور همزمان کار می‌کند باشید، اهمیت چندانی ندارد. این کتاب به شما به عنوان یک فرد، کمک خواهد کرد تا کارهایتان را به شکل بهتری انجام دهید. این یک کتاب نظریه‌پردازی نیست- ما روی موضوعات عملی و کاربردی متمرکز می‌شویم و به شما کمک می‌کنیم از تجربیاتتان استفاده کرده تا تصمیمات بهتری اتخاذ نمایید. واژه‌ی «پراگماتیک»[[1]](#footnote-1) از ریشه‌ی لاتین «*پراگماتیکوس»[[2]](#footnote-2)* به دست آمده-«ماهر در کسب وکار»- که خود این واژه نیز از لغت یونانی *νιετταρπ*، به معنی «جهت انجام» مشتق شده است. این کتاب، کتابی درباره‌ی عمل کردن است.

برنامه‌نویسی یک فن است. در ساده‌ترین حالت خود، با استفاده از برنامه‌نویسی این امکان برای یک کامپیوتر فراهم می‌شود که آنچه را که نیاز دارید، اجرا کند (یا آنچه کاربرتان نیاز دارد اجرا کند). به عنوان یک برنامه‌نویس، شما بایستی بخشی از یک شنونده، بخشی از یک مشاور، بخشی از یک مفسر، و در نهایت بخشی از یک دیکتاتور باشید. باید تلاش کنید تا نیازهای کمیابی را تحت کنترل گرفته و راهی را برای بیان آن‌ها به گونه‌ای پیدا کنید که یک ماشین صرفا بتواند بین آن‌ها عدالت را برقرار کند. کارتان را به گونه‌ای مستند کنید که دیگران قادر به فهم آن باشند، و همچنین بایستی کارتان را به گونه‌ای مهندسی کنید که دیگران قادر باشند از دستاورد شما به عنوان مبنا برای موفقیت‌های بعدی خودشان بهره گیرند. حتی بیشتر از آن اینکه، باید تلاش‌کنید که تمامی این‌کارها را در مقابل تیک‌تاک بی‌امان زمان پروژه انجام دهید. هر روز معجزات کوچکی را انجام می‌دهید.

این یک شغل دشوار است.

افراد زیادی وجود دارند که به شما کمک ارائه می‌دهند. فروشندگان ابزارها، شگفتی‌هایی را که توسط محصولاتشان ارائه می‌شود بازاریابی می‌کنند. معلمان روش‌شناسی متعهد می‌شوند که تکنیک‌هایشان، نتایج را تضمین خواهد کرد. همه ادعا می‌کنند که زبان برنامه‌نویس آن‌ها بهترین است، و هر کدام از سیستم‌عامل‌های موجود، جوابی برای تمامی مشکلات قابل تصور است.

البته هیچ کدام از اینها صحیح نیست. جواب ساده‌ای برای آن‌ها وجود ندارد. هیچ چیزی تحت عنوان بهترین راه‌حل وجود ندارد، حال می‌خواهد یک ابزار باشد، می‌خواهد یک زبان باشد یا یک سیستم عامل. تنها چیزی که می‌تواند وجود داشته باشد سیستم‌هایی هستند که در مجموعه‌ شرایط‌های خاص، مناسب‌تر از بقیه هستند.

اینجاست که مصلحت‌گرایی وارد می‌شود. نبایست شما به هیچ تکنولوژی خاصی وابسته باشید، اما باید دارای پیش‌زمینه‌ای وسیع و پایگاهی از تجربه بوده تا به شما این امکان را بدهند که راه‌حل‌های خوبی را در شرایط خاص انتخاب کنید. پیش‌زمینه‌ی شما ریشه در درک‌تان از اصول اولیه‌ی علوم کامپیوتر دارد، و تجربه‌تان از طریق محدوده‌ی وسیعی از پروژه‌های عملی حاصل می‌شود. تئوری و تمرین با هم ترکیب می‌شوند تا شما را قوی سازند.

شما روش‌هایتان را بر طبق تناسب با شرایط و محیط موجود تنظیم می‌کنید. اهمیت نسبی تمامی عواملی را که بر روی یک پروژه تاثیر می‌گذارند تشخیص داده و از تجربه‌تان برای ایجاد بهترین راه‌حل‌ها استفاده می‌کنید. و این فرآیند را در حین رشد پروژه مرتبا تکرار می‌کنید. برنامه‌نویسان عملگرا کار را به اتمام رسانده، و به خوبی آن را انجام می‌دهند.

مخاطبان این کتاب چه کسانی هستند؟

مخاطب این کتاب کسانی هستند که می‌خواهند تبدیل به برنامه‌نویسان موثرتر و پربارتری گردند. شاید شما از اینکه پتانسیل‌های نهان خود را بکار نگرفته‌اید احساس ناامیدی می‌کنید. شاید همکارانی را مشاهده می‌کنید که به نظر می‌رسد از یک سری ابزارها برای پربارتر نشان دادن خودشان نسبت به شما بهره می‌گیرند. شاید کار فعلی شما مبتنی بر فناوری‌های قدیمی‌تری باشد، و شما می‌خواهید بدانید که چگونه ایده‌های جدید می‌توانند در بهبود کارهایی که انجام می‌دهید موثر باشند.

ما نه وانمود می‌کنیم که پاسخ تمامی (یا حتی بیشتر) سوالات را می‌دانیم، و نه اینکه کل ایده‌های ما در هر شرایطی قابل اجراست. تمام آن چیزی که می‌خواهیم بگوییم این است که اگر از روش‌های ما پیروی کنید، سریعتر تجربه کسب کرده، کارایی‌تان افزایش یافته، و در نهایت درک بهتری از کل فرآیند توسعه خواهید داشت و قادر خواهید بود که نرم‌افزارهای بهتری تولید کنید.

چه‌چیزی شما را به یک برنامه‌نویس عملگرا تبدیل می‌کند؟

تمامی توسعه‌دهندگان، از لحاظ ضعف و قوت شخصی، اولویت‌ها وکج‌خلقی‌هایشان بی‌مانندند. در طول زمان، هر یک از آن‌ها محیط‌های شخصی مختص به خود را بوجود می‌آورند. این محیط‌ها، ویژگی‌های برنامه‌نویس را دقیقا همانگونه که انواع سرگرمی‌‌ها، طرز لباس پوشیدن، و مدل موی سر او نمایش می‌دهند، منعکس می‌نمایند. با اینحال، اگر شما یک برنامه‌نویس عملگرا باشید، آنگاه ویژگی‌های فراوانی را به اشتراک خواهید گذاشت:

* زودسازگار/تندسازگار. دارای قابلیت‌های غریزی در رابطه با فناوری‌ و تکنیک‌ها بوده، و دوست دارید سریعا توانایی‌تان را ارزیابی کنید. وقتی که مبحث جدیدی ارائه می‌شود، می‌توانید سریعا آن را درک کرده و با بقیه‌ی معلوماتتان ادغام کنید. اعتماد به نفس‌تان توسط تجربه شکل گرفته است.
* کنجکاو. تمایل به سوال پرسیدن دارید. که بسیار شسته رفته است- *چگونه این‌ کار را انجام دادی؟ آیا با این کتابخانه مشکل داشتی؟ این BeOS که زیاد درباره‌ی آن سخن می‌گویند چیست؟ لینک‌های نمادین چگونه پیاده‌سازی می‌شوند؟* شما در مقابل حقایق کوچک به مانند یک همسترید که هر کدام از این حقایق ممکن است ده سال بعد بر روی برخی از تصمیمات تاثیرگذار باشند.
* متفکر انتقادی. قبل از اثبات یک واقعیت، بندرت مسائل را ساده می‌پندارید. وقتی که همکاران می‌گویند «زیرا روش انجام آن اینگونه است،» یا یک فروشنده وعده‌ می‌دهد که راه‌حل تمامی مشکلاتتان را دارد، اینجاست که بوی یک چالش به مشامتان می‌رسد.
* واقع‌بین. سعی دارید که ماهیت پنهان تمامی مشکلاتی را که با آن‌ها مواجه می‌شوید درک کنید. این واقع‌بینی به شما حس خوبی نسبت به میزان دشواری مسائل، و مقدار زمان لازم برای انجام آن‌ها می‌دهد. درک اینکه یک فرآیند می‌تواند دشوار باشد یا زمان زیادی برای تکمیل نیاز داشته باشد به شما استقامت لازم برای ادامه‌ی کار را خواهد داد.
* همه فن حریف. به شدت تلاش می‌کنید تا با محدوده‌ی وسیعی از فن‌آوری‌ها و محیط‌های عملیاتی آشنا شده، و همچنین تلاش می‌کنید که شانه به شانه‌ی پیشرفت‌های جدید پیش بروید. با اینکه ممکن است کار فعلی شما نیازمند این باشد که یک متخصص باشید، همیشه این قابلیت را خواهید داشت که در حوزه‌ها و چالش‌های جدید پیشرفت کنید.

ما اساسی‌ترین مشخصه‌ها را تا آخر بیان کردیم. همه‌ی برنامه‌نویسان عملگرا این ویژگی‌ها را به اشتراک می‌گذارند. آن‌ها بقدری اساسی‌اند که می‌توانند به صورت نکات زیر بیان شوند:

نکته ۱

به حرفه‌تان(کارتان) علاقه نشان دهید.

به نظر ما توسعه‌ی نرم‌افزار بدون اهمیت به هرچه بهتر انجام دادن آن، بی‌هدف است.

نکته ۲

در مورد کارتان فکر کنید

برای تبدیل شدن به یک برنامه‌نویس عملگرا، می‌خواهیم شما را به تفکر درباره‌ی کاری که هم‌اکنون در حال انجام آن هستید دعوت کنیم. این یک ممیزی در رابطه با شیوه‌های موجود نیست که فقط یکبار تکرار شود --بلکه یک ارزیابی مکرر بسیار مهم برای تصمیماتی است که هر روز و در هر پروژه‌ای اتخاذ می‌کنید. هرگز کاری را بدون فکر کردن در مورد آن انجام ندهید. بطور مداوم فکر کرده، و کارتان را در زمان اجرا (زمان‌های واقعی) نقد کنید. شعار شرکت مجرب IBM تحت عنوان، **فکر کنید!،** همانشعار برنامه‌نویسان عملگراست.

اگر انجام اینکار برای شما سخت به نظر می‌رسد، اینجاست که شخصیت *واقع‌بین‌تان* را به نمایش می‌گذارید. اینکار زمان زیادی را از شما صرف می‌کند- که احتمال به یقین، در همین حالت هم باز، زمان‌ تحت فشار است. پاداش انجام اینکار عبارتست از ارتباطی تنگاتنگ‌تر با کاری که دوست دارید، ایجاد یک حس رمزآلود نسبت به محدوده‌ی روزافزونی از موضوعات، و لذت از پیشرفت و بهبودهای روزمره. در درازمدت، از آنجا که شما و تیم‌تان کارآمدتر شده، کدهایی می‌نویسید که پشتیبانی از آن‌ها ساده‌تر بوده، و زمان کمتری را صرف جلسات می‌کنید، زمانی را که سرمایه‌گذاری کرده‌اید به شما باز‌ خواهد گشت.

عملگرایان فردی، تیم‌های بزرگ

بعضی‌ها معتقدند که کارهای تیمی و پروژه‌های دشوار نباید به صورت انفرادی انجام شوند. به نظر این افراد "ساختار نرم‌افزار دارای یک نظام مهندسی است که اگر افراد تیم به صورت انفرادی تصمیم‌گیری کنند این نظم از بین خواهد رفت".

ما مخالف این عقیده هستیم.

بله، ساختار نرم‌افزار **باید** دارای یک نظام مهندسی شده باشد. با اینحال، این امر مانعی برای مهارت‌های فردی نمی‌باشد.کلیساهای بزرگ ساخته شده در اروپا در قرون وسطی را در نظر بگیرید. هر کدام حاصل تلاش‌های هزاران نفر در سال است که بیش از چندین دهه طول کشیده‌اند. تجربیات و دستاورد‌های بدست آمده به سازندگان بعدی منتقل شد، که آن‌ها نیز با افزودن هنر خودشان،کیفیت مهندسی سازه را ارتقا دادند. اما نجاران، سنگتراشان، کارورها، و شیشه‌برها همگی صنعتگرانی بوده‌اند که نیازهای مهندسی را، جهت تولید مجموعه‌ای کامل تدوین کرده‌اند. مجموعه‌ای که خالصانه ارتقاء قسمت مکانیکی سازه را در بر گرفته اند. این ناشی از اعتقاد صنعتگران به مشارکت‌های فردی‌شان بود که پروژه را پایدار می‌نمود:

*ما که فقط سنگ‌ها را برش می‌دهیم بایستی از قبل کلیساهایی جامع را تصویرسازی کنیم.*

* **عقیده‌ی یک کارگر معدن**

در میان ساختار کلی یک پروژه همیشه جایی برای مهارت‌فردی و فردگرایی وجود دارد. این امر به ویژه در وضعیت فعلی مهندسی نرم‌افزار صدق می‌کند. صد سال بعد، مهندسی ما ممکن است به همان اندازه قدیمی به نظر برسد که تکنیک‌های استفاده شده توسط سازندگان کلیساهای قرون وسطی برای مهندسین عمران حال حاضر به نظر می‌رسد، در حالیکه هنوز هم از مهارت‌های فردی ما با افتخار یاد خواهد شد.

این یک فرآیند پیوسته است

*جهانگردی که دانشکده ایتن انگلیس را ملاقات می‌کرد از باغبان پرسید که چگونه چمن‌ها را به این خوبی پرورش داده است. او پاسخ داد «ساده است، هر روز صبح شبنم‌ها را به وسیله‌ی یک قلم مو، از روی چمن‌ها پاک کرده، یک روز در میان چمن‌ها را کوتاه نموده، و یک بار در هفته آن‌ها را غلتک‌کاری می‌کنم.»*

*جهانگرد پرسید: «فقط همین؟»*

*باغبان پاسخ داد «صد در صد. این کار را 500 سال ادامه بده تا تو هم چمنی به این ظرافت داشته باشی.»*

چمن‌های عالی نیاز به مراقبت‌های روزانه بسیار کمی دارند، برنامه نویس‌های عالی هم به همین شیوه‌اند. مشاوران مدیریت بسیار خوششان می‌آید که از کلمه‌ی *کایزن* در مکالمه‌هایشان استفاده کنند. «کایزن» کلمه‌ای ژاپنیست که به معنی پیشرفت‌های بسیار کوچک اما مداوم است. این واژه به عنوان یکی از اصلی‌ترین دلایل دستاوردهای چشمگیر تولیدکنندگان ژاپنی در زمینه‌ی بهره‌وری و کیفیت در نظر گرفته شده و در سرتاسر جهان کپی‌برداری شد. کایزن در مورد فردیت نیز صدق می‌کند. هر روز، تلاش کنید تا مهارت‌هایتان را بهبود بخشیده و ابزارهای جدیدی را به مجموعه‌تان اضافه کنید. برخلاف چمن‌های دانشگاه ایتن، فقط ظرف چند روز نتیجه کار را مشاهده خواهید کرد. بعد از چند سال، از میزان شکوفایی تجربیات و رشد مهارت‌هایتان شگفت‌زده خواهید شد.

کتاب چگونه سازمان یافته است؟

این کتاب به صورت مجموعه‌ای از بخش‌های کوچک نوشته شده است. هر کدام از این بخش‌ها مجزا از سایر بوده، و بر یک موضوع خاص دلالت دارد. شما با لیستی از مراجعات متقابل (مراجعه از فهرستی به فهرست دیگر) برخورد می‌کنید که کمک می‌کنند هر کدام از موضوعات در محتوای مربوط به خود قرار گیرند. مختارید که بخش‌ها را به هر ترتیبی که دوست دارید بخوانید—این کتابی نیست که الزاما باید از ابتدا به انتها خوانده شود.

گهگاه با یک جعبه‌ی متنی با برچسب نکته‌ی nام مواجه می‌شوید(مانند نکته 1، «حرفه‌تان را پاس بدارید» در صفحه‌ی ایکس). به نظر ما نکات، به همان اندازه‌ی قسمت‌های برجسته‌ی متن، برخی ویژگی‌های مستقل خاص خود را دارند—روزانه با آن‌ها سر و کار داریم.

ضمیمه‌ی الف شامل مجموعه‌ای از منابع است: فهرست کتب، لیستی از لینک‌های مرتبط با صفحات وب، و لیستی از نشریه‌های دوره‌ای، کتاب‌ها، و سازمان‌های حرفه‌ای توصیه شده. از طریق کتاب، مراجعی را به لیست کتاب‌ها و لینک‌های مرتبط خواهید یافت-مانند به ترتیب [KP99] و [URL 18].

ما در هرکجا که مناسب بوده، چالش‌ها و تمارینی را قرار داده‌ایم. تمرین‌ها نسبتا دارای جواب‌های سر راستی هستند، درحالیکه چالش‌ها تقریبا یک جواب ثابت ندارند. برای اینکه ایده‌ای از تفکراتمان را به شما منتقل کنیم، پاسخ‌‌های خودمان به تمرین‌ها را در ضمیمه‌ی ب قرار داده‌ایم، اما تعداد کمی از آن‌ها دارای فقط یک جواب صحیحند. شاید چالش‌ها بتوانند پایه‌ی مباحث گروهی یا کار بر روی یک مقاله را در دروس برنامه‌نویسی پیشرفته شکل دهند.

چه چیزی در یک لغت نهفته است؟

*هامپتی دامپتی با لحنی متکبرانه گفت، «وقتی که من کلمه‌ای را به کار می‌برم، آن کلمه فقط معنی‌ای که من انتخاب می‌کنم را دارد – نه بیشتر و نه کمتر.»*

- لوئیز کارول، در آنسوی آینه

شما می‌توانید به صورت پراکنده در کتاب، بیت‌های کوچکی از اصطلاحات را مشاهده کنید - کلماتی کاملا انگلیسی که به منظور معنی بخشیدن به یک اصطلاح فنی تغییر یافته، یا کلمات ساختگی دهشتناک که معانی آن توسط دانشمندان علوم کامپیوتر از سر کینه توزی با زبان مربوطه انتخاب شده‌اند. اولین باری که ما از هر کدام از این اصطلاحات فنی استفاده کرده‌ایم، سعی نموده‌ایم که آن را تعریف کرده، یا حداقل به معنی آن اشاره‌ای داشته باشیم. با اینحال، مطمئنیم که برخی از آن‌ها به دست فراموشی سپرده شده، و بقیه، مانند شیء یا پایگاه‌داده‌های رابطه‌ای، به قدری بدیهی و پرکاربردند که تعریف کردن آن‌ها می‌تواند خسته‌کننده باشد. اگر به لغتی برخورد کردید که قبلا معنی آن را نمی‌دانستید، لطفا آن را بی‌خیال نشوید. سعی کنید که به وسیله‌ی وب یا یک دایره‌المعارف علوم کامپیوتر برای آن زمان گذاشته و معنی آن را پیدا کنید. و اگر فرصت داشتید، برای ما ایمیل فرستاده و بابت آن شکایت کنید، تا بتوانیم در ویرایش‌های بعدی تعاریف آن‌ها را در کتاب قرار دهیم.

با تمام این تفاسیر، ما تصمیم گرفتیم که از دانشمندان علوم کامپیوتر انتقام بگیریم. برخی از این اصطلاحات بسیار متناسب با مفاهیم‌اند، کلماتی که ما تصمیم گرفتیم از آن‌ها چشم‌پوشی کنیم. چرا؟ زیرا اصطلاحات موجود طبیعتا محدود به دامنه‌ای از مشکلات معین، یا مرحله‌ی خاصی از توسعه است. با اینحال، یکی از اساسی‌ترین فلسفه‌های کتاب اینست که بیشتر تکنیک‌هایی را ارائه‌شده فراگیرند مانند: مدلاسیون‌های اعمال شده بر روی کدها، طرح‌ها، مستندات، و سازمان تیم. وقتی که ما تصمیم گرفتیم که از اصطلاحات قراردادی در یک چارچوب گسترده‌تر استفاده کنیم، باعث سردرگمی شد—به نظر می‌رسید که نمی‌توانیم به بار اضافی ناشی از معنی اصلی واژه چیره گردیم. وقتی که این اتفاق رخ داد، ما با ابداع واژه‌های جدید توسط خودمان به کاهش انحطاط زبان کمک کردیم.

فصل ۱

فلسفه‌ی عملگرایی

چه چیزی یک برنامه‌نویس عملگرا را متمایز می‌سازد؟ به عقیده‌ی ما یک گرایش، یک سبک، یک فلسفه از مشکلات پیش‌رو و راه‌حل‌های آن‌هاست که او را متمایز می‌کند. این‌گونه برنامه‌نویسان فراتر از یک مشکل آنی تعمق نموده، همیشه سعی دارند تا مشکلات را در یک زمینه‌ی گسترده‌تر در نظر گرفته، و همیشه تلاش می‌کنند تا به یک ذهنیت بزرگ‌تر دست پیدا کنند. با همه‌ی این‌ها، چگونه می‌توان بدون در نظر گرفتن این زمینه‌ی گسترده‌تر، به یک برنامه‌نویس عملگرا تبدیل شد؟ چگونه شما می‌توانید مقایساتی هوشمندانه‌تر را انجام داده و تصمیماتی آگاهانه‌تر اتخاذ نمایید؟

یکی دیگر از کلیدهای موفقیت آن‌ها این است که مسئولیت تمامی کارهایی را که انجام می‌دهند به عهده می‌گیرند، که ما این مبحث را در بخش *گربه کدهایم را خورد* توضیح می‌دهیم. از آنجا که برنامه نویسان عملگرا مسئولیت‌پذیرند، وقتی که پروژه‌هایشان از سر غفلت بخوبی عمل نکند، اینگونه نیست که به شکلی ساکن و بی‌هدف یکجا نشسته و هیچ واکنشی نسبت به این مشکلات نداشته باشند. دربخش *آنتروپی نرم‌افزار*، به شما خواهیم گفت که چگونه پروژه‌هایتان را بدیع و تازه باقی بگذارید.

بسیاری از مردم، گاهی بدلایل خوب، و گاهی به علت یک قوه‌ی جبری قدیمی، تغییر را یک امر دشوار می‌یابند. در بخش *سوپ سنگ و قورباغه‌های آب‌پز*، ما یک استراتژی را به عنوان محرکی برای تغییر و (در منافع تعادل) یک داستان احتیاطی از دوزیستانی که از خطر ناشی از تغییرات تدریجی چشم‌پوشی کردند را ارائه‌ می‌دهیم.

یکی از مزایای درک زمینه‌ای که بر روی آن کار می‌کنید این است که ساده‌تر می‌توانید تشخیص دهید که تا چه حد بایستی نرم‌افزارتان خوب باشد. گاهی اوقات عالی تنها گزینه است، اما اغلب بحث مصلحت پیش می‌آید. ما این مبحث را در بخش *نرم‌افزارهای کارآمد* بررسی می‌کنیم.

قطعا شما بایستی مبانی گسترده‌ای از دانش و تجربه را در اختیار داشته باشید تا همگی این‌ها را مدیریت کنید. یادگیری، یک فرآیند مداوم و پیوسته‌ است. در بخش *وضعیت دانش شما*، در مورد برخی از استراتژی‌های تقویت نیروی جنبشی بحث خواهیم کرد.

در نهایت، هیچکدام از ما در خلاء کار نمی‌کنیم. همه‌ی ما زمان زیادی را در تعامل با دیگران صرف می‌کنیم. در بخش *ارتباط!* راه‌های هر چه بهتر انجام دادن این مسئله را فهرست می‌کنیم.

برنامه‌نویسی عملگرا ریشه در فلسفه‌ی تفکر عملگرا دارد. این فصل مبانی این فلسفه را در برگرفته است.

1. گربه کُدهایم را خورد

بزرگترین ضعف‌ها، ترس از ضعیف ظاهر شدن است.

J. B. Bossuet, Politics from Holy Writ, 1709

یکی از بنیادهای فلسفه‌ی عملگرایی، به عهده‌گرفتن مسئولیت خود و اعمال خود از لحاظ پیشرفت شغلی، پروژه‌ها، و کارهای رومزره‌تان می‌باشد. یک برنامه‌نویس عملگرا مسئولیت انجام کارهای خودش را بر عهده گرفته، و ترس از پذیرش جهالت‌ها و خطاهایش را ندارد.محض اطمینان، این بهترین جنبه‌ی برنامه‌نویسی نیست، اما بهرحال اتفاق می‌افتد-حتی در بهترین پروژه‌ها. علیرغم آزمون‌ها، مستندسازی‌های عالی، و فرآیند‌های اتوماتیک قوی، بازهم احتمال خطاهایی از قبیل تاخیر در تحویل، و ظهور مشکلات فنی پیش‌بینی نشده وجود دارند.

این مشکلات اتفاق می‌افتند، و از اینرو تا آنجایی که بتوانیم باید سعی کنیم به صورت حرفه‌ای با آن‌ها مقابله کنیم. یعنی به شکلی صریح و کاملاً صادقانه. ما می‌توانیم به قابلیت‌هایمان افتخار کنیم، اما می‌بایست در‌ رابطه با قصورمان—جهالت‌ها و اشتباهات‌مان-- صادق باشیم.

مسئولیت‌پذیر باشید

مسئولیت مفهومی است که معمولاً مشتاقانه با آن موافقید. شما برای اطمینان از صحت انجام کاری تعهد داده، اما لزوما بر روی تمامی جوانب آن کنترل مستقیم ندارید. علاوه بر انجام تمامی وظایف شخصی خود‌تان به بهترین حالت ممکن، بایستی وضعیت را در رابطه با خطرات احتمالی که خارج از کنترل شماست تجزیه و تحلیل نمایید. شما حق ندارید که مسئولیت یک وضعیت غیرممکن، یا وضعیتی را که احتمال خطر در آن بسیار بالاست بعهده بگیرید. بایستی برپایه‌ی اخلاق و قضاو‌ت خودتان تصمیم‌گیری کنید.

وقتی که مسئولیت یک پیامد را بعهده گرفتید، بایستی پاسخگوی آن نیز باشید. وقتی که اشتباهی مرتکب شدید (همانگونه که که همه‌ی ما اشتباه می‌کنیم) یا خطایی در قضاوتتان رخ داد، صادقانه آن را پذیرفته و سعی کنید گزینه‌های را پیشنهاد دهید.

کس یا چیز دیگری را مقصر ندانسته و یا بهانه نیاورید. فروشنده، زبان برنامه‌نویسی، مدیریت، یا همکارانتان را مقصر تمامی مشکلات جلوه ندهید. تمامی و یا بخشی از آن‌ها ممکن است نقشی داشته باشند، اما تصمیم با شماست که راه‌حل‌ها را ارائه داده یا اینکه بهانه‌های داغ بیاورید.

اگر خطری وجود دارد که فروشنده نمی‌خواهد به شما ابلاغ کند، آنگاه شما باید یک برنامه‌ی احتمالی را در نظر بگیرید. اگر دیسک خراب شده—و تمامی سور‌کدهایتان با آن از بین رفته—و نسخه‌ی پشتیانی از آن ندارید، این تقصیر شماست. اینکه به رئیس‌تان بگویید که «گربه کدهایم را خورد» قانع‌کننده نیست.

نکته ۳

به جای بهانه‌های توجیه ناپذیر، راه‌حل‌ ارائه دهید.

قبل از اینکه در مورد قابل اجرا نبودن، دیر شدن، یا از بین رفته هرچیزی با کسی سخنی به میان بگذارید، صبر کرده و به خودتان گوش فرا دهید. با اردک پلاستیکی یا گربه روی مانیتورتان صحبت کنید. آیا عذرهایتان منطقی به نظر می‌رسد یا احمقانه است؟ آیا قرار است برای رئیس‌تان چگونه به نظر برسند؟

سریعا مکالمه را در ذهنتان مرور کنید. طرف مقابل احتمال دارد در پاسخ چه بگوید؟ آیا آن‌ها سوالاتی مانند «آیا این روش را امتحان کرده‌ای...» یا «آیا این راه‌حل را در نظر گرفته‌ای؟» را از شما خواهند پرسید؟ شما در پاسخ چه خواهید گفت؟ آیا قبل از اینکه پیش آن‌ها رفته و خبرهای بد را اطلاع دهید، راه‌حلی باقی مانده است که امتحان نکرده باشید؟ بعضی اوقات، شما فقط با دانستن اینکه قرار است چه بگویند، آن‌ها را از دردسر دور می‌کنید.

به جای عذر و بهانه، گزینه‌هایی را تدارک ببینید. نگویید که قابل انجام نیست؛ توضیح دهید که برای خروج از بحران چه کارهایی می‌توان انجام داد. آیا باید کد را دور انداخت؟ آن‌ها را در زمینه‌ی اهمیت بهینه‌سازی و بهبود کدها (Refactoring) آموزش دهید. آیا نیاز است که جهت یافتن بهترین روش برای پیش‌روی، زمانی را به نمونه‌سازی اختصاص دهید. آیا می‌بایست اتوماسیون یا آزمایش بهتری را جهت جلوگیری از وقوع مجدد آن ارائه دهید؟ شاید به منابع بیشتری احتیاج داشته باشید. بیمی از سوال‌پرسیدن، یا پذیرفتن اینکه به کمک نیاز دارید نداشته باشید.

سعی کنید بهانه‌های غیرقابل قبول را قبل از اینکه با صدای بلند جار زده شوند آشکار نمایید. اگر مجبور شدید، ابتدا آن را با گربه‌تان در میان بگذارید. هرچه باشد، شاید درنهایت گربه‌ی کوچکتان بهانه‌ها را گردن گیرد....

بخش‌های مرتبط

* نمونه‌های اولیه و یادداشت‌های چسبان
* بهینه‌سازی و بهبود کدها
* کدهایی که براحتی قابل آزمایشند
* اتوماسیون همه‌جا حاضر
* تست‌های ظالمانه

چالش‌ها

* اگر فردی—مانند تحویلدار بانک یا مکانیک خودرو، یا منشی دفتری—با یک بهانه‌ی غیر قابل قبول نزد شما بیاید، چگونه عکس‌العمل نشان می‌دهید؟ در نهایت در مورد آن‌ها و شرکتشان چه نتیجه‌گیری می‌کنید؟

1. آنتروپی[[3]](#footnote-3) نرم‌افزار

اگرچه توسعه‌ی نرم‌افزار از تمام قوانین فیزیک مصون است، اما آنتروپی به شدت به ما آسیب می‌رساند.

آنتروپی یک اصطلاح در علم فیزیک است که اشاره به "اختلال و بی‌نظمی" در یک سیستم دارد. متاسفانه، قوانین ترمودینامیک تضمین می‌کنند که بی‌نظمی در جهان، تمایل دارد که به حداکثر مقدار خود برسد. زمانی‌که اختلالی در نرم‌افزاری بروز پیدا می‌کند، برنامه‌نویس از آن به عنوان «پوسیدگی نرم‌افزار»[[4]](#footnote-4) یاد می‌کند.

عوامل زیادی در به وجود آوردن این خرابی نقش دارند. به نظر میرسد که یکی از مهمترین دلایل آن در زمان کار روی یک پروژه، دلیل روانشناسی و یا فرهنگی باشد. حتی اگر شما یک تیم تک‌نفره باشید، روانشناسی پروژه‌تان می‌تواند امری بسیار ظریف و حساس باشد. علیرغم بهترین افراد دخیل در یک پروژه و برنامه‌های مفصل، باز هم یک پروژه می‌تواند در زمان حیاتش، خرابی و تباهی را تجربه کند و برخلاف آن پروژه‌های دیگری هم هستند که، علیرغم مشکلات فراوان و تنزلات مداوم‌‌شان، به شکل موفقیت آمیزی در مقابل خصلت بی‌نظمی طبیعی خود مبارزه کرده و به گونه‌ای مدیریت می‌شوند که به خوبی ظاهر گردند.

چه چیزی تفاوت ایجاد می‌کند؟

در بخش‌های مرکزی شهر، برخی از ساختمان‌ها زیبا و تمیز هستند، در حالیکه بقیه ویرانه‌هایی کثیف‌اند. چرا؟ محققان زمینه‌ی جرم و جنایت و فروپاشی شهری، به یک مکانیسم روانی و فیزیولوژیکی جالب دست یافتند که این مکانیسم می‌تواند سریعا ساختمان‌های تمیز، سالم، و مسکونی را تبدیل به خرابه‌های متروکه نماید.

پنجره‌ی شکسته

یک پنجره‌ی شکسته که به مدت قابل توجهی تعمیر نشود، حس واگذاری و متارکه را در ساکنان آن ساختمان القا می‌کند—حس یا نیرویی که به آن ها می گوید به ساختمان اهمیت ندهید. بنابراین پنجره‌ی دیگری شکسته می‌شود. مردم شروع به ریخت و پاش می‌کنند. نقاشی‌های دیواری (گرافیتی) پدیدار می‌گردند. آسیب‌های ساختاری جدی آغاز می‌شوند. در یک زمان نسبتا کوتاه، ساختمان به حدی آسیب می‌بیند که مالک آن تمایلی به تعمیر آن نخواهد داشت، و حس متارکه‌ی ساختمان به واقعیت بدل می‌‌شود.

نظریه‌ی «پنجره‌ی شکسته»، ادارات پلیس نیویورک و دیگر شهرهای بزرگ را تحت تاثیر قرار داد به گونه‌ای که کوچکترین مشکلات را ریشه‌کن کنند تا بتوانند از مشکلات بزرگتر در امان باشند و این امر تاثیرگذار بود: تحت کنترل گرفتن پنجره‌های شکسته، نقاشی‌های دیواری، و دیگر تخلفات کوچک، به شکلی جدی آمار جرم و جنایت را کاهش داد.

نکته ۴

پنجره‌های شکسته را تعمیر کنید

«پنجره‌های شکسته (طراحی‌های بد، تصمیمات نادرست، یا کدهای ضعیف) را به حال خود رها نکنید». به محض اینکه متوجه هر کدام از آن‌ها شدید، سریعا بازسازی‌شان نمایید. اگر وقت کافی برای بازسازی کامل نداشتید، پس آن‌ها را *مهر و موم کرده و ببندید*. می‌توانید کدهای مشکل‌دار را به طور موقت از طریق «تبدیل کد به توضیح»[[5]](#footnote-5) غیر فعال کنید یا پیغامی تحت عنوان «هنوز پیاده‌سازی نشده است» را نمایش دهید و یا از داده‌های مصنوعی کمک بگیرید. برای پیشگیری از خسارت بیشتر و نشان دادن اینکه کنترل اوضاع در دستانتان است، اقداماتی را انجام دهید.

ما سیستم‌های کارا و کامل زیادی دیده‌ایم که به محض اینکه اولین مشکل در آن‌ها رخ داده است سریعا کیفیت خود را از دست داده‌اند. همچنین عوامل دیگری نیز وجود دارند که می‌توانند بر روی فرسایش نرم‌افزار تاثیرگذار باشند و ما در جاهای دیگر به طور خلاصه به آن‌ها رسیدگی خواهیم کرد؛ اما سهل‌انگاری نسبت به سایر عوامل، *سریع‌تر* باعث فرسایش می‌شود.

شاید فکر کنید که هیچ‌کس زمان کافی برای بررسی و ترمیم پنجره‌های شکسته‌ی درون یک پروژه را ندارد. اگر قرار است به این نوع تفکر ادامه دهید، پس بهتر است برای تبدیل‌شدن محله -- که همان کدها هستند-- به یک زباله‌دان برنامه‌ریزی کرده یا اینکه برنامه‌ی جابه‌جایی به محله‌ی دیگری --شروع پروژه‌ی دیگری-- را در سر داشته باشید. اجازه ندهید که بی‌نظمی پیروز شود.

سر و کله زدن با مشکلات

در مقابل پنجره‌ی شکسته، داستان یکی از وابستگان ثروتمند «اندی[[6]](#footnote-6)» وجود دارد. خانه‌ی او بدون لکه و تمیز، زیبا، پر از عتیقه‌جات با ارزش و اثر‌های هنری و غیره بود. یک روز، تابلو فرش او که کاملا نزدیک شومینه آویخته شده بود، آتش گرفت.اداره‌ی آتش‌نشانی برای جلوگیری از خسارت –و نجات خانه—سریعا وارد عمل شدند. اما قبل از اینکه شلنگ‌های بزرگ و کثیف‌شان را وارد خانه کنند، برای تا کردن و خارج نمودن گلیمی که بین درب جلویی و منبع آتش قرار گرفته بود، متوقف شدند –در حالیکه همان لحظه آتش شعله می‌کشید.

آن‌ها نمی‌خواستند گلیم را خراب کنند.

البته این مثال بسیار افراطی است اما در رابطه با نرم‌افزار نیز باید به همین ترتیب عمل شود. پنجره‌ای شکسته –قطعه کدی که بد طراحی شده است، تصمیم مدیریتی غلطی که تیم باید در طول پروژه با آن کنار آید— به تنهایی می‌تواند شروعی برای افول باشد. اگر خودتان را در حال کار روی پروژه‌ای یافتید که مشکلات یا نواقصی در آن وجود داشتند، سریعا این نگرش به ذهنتان خطور می‌کند که «کل کد چرند است و من هم به همین روال را ادامه‌ خواهم داد». این که پروژه تاکنون به خوبی اجرا شده است، اهمیتی ندارد. در اولین آزمایشی که منجر به «نظریه‌ی پنجره‌ی شکسته» شد، خودرویی به مدت یک هفته در منطقه‌ای رها گشت درحالیکه کسی آن را لمس نکرد. اما به محض اینکه یکی از شیشه‌های خودرو شکست، ظرف چند ساعت، ماشین لخت و زیر و رو شد.

همچنین، اگر شما هم خودتان را در تیم و پروژه‌ای یافتید که از همان ابتدا بی نقص بود—ماهرانه نوشته شده، خوب طراحی شده، و آراسته بود—به احتمال زیاد مانند آتش‌نشانان، توجه ویژه‌ای خواهید کرد که ظرافت آن را به هم نزنید. حتی اگر شعله‌های آتش زبانه بکشند (ضرب‌الاجل، تاریخ انتشار، نسخه‌ی نمایشی نمایشگاه، و غیره)، دوست ندارید اولین نفری باشید که آن را نابسامان می‌کند.

بخش ‌های مرتبط

* سوپ سنگ و قورباغه‌ی آب پز
* بازسازی
* تیم‌های عملگرا

چالش

* از طریق بررسی محاسبات «اطرافیان»، به تقویت تیم‌تان کمک کنید. دو یا سه «پنجره‌ی شکسته» (اِشکال) را انتخاب کرده و با همکارانتان درباره‌ی نوع مشکل و چگونگی رفع آن بحث کنید.
* آیا می‌توانید بگویید که چه زمانی شیشه‌ای برای اولین بار خواهد شکست؟ عکس‌العمل شما چیست؟ اگر این شکستن ناشی از تصمیم شخص دیگری یا به دستور یکی از مدیران بوده باشد، چه کاری می‌توانید برای رفع آن انجام دهید؟

1. سوپ سنگ و قورباغه‌ی آب‌پز

سه سرباز که از میدان جنگ به خانه می‌برگشتند، گرسنه بودند. وقتی دهکده‌ی پیش رویشان را دیدند، ذوق زده شدند —آن‌ها مطمئن بودند که اهالی دهکده به آن‌ها طعامی برای خوردند خواهند داد. اما وقتی که به آنجا رسیدند، متوجه شدند که تمامی درها قفل و تمامی پنجره‌ها بسته شده‌اند. در اثر سال‌های طولانی جنگ، دهکده دچار قحطی غذا شده و هر چه که داشتند را انبار کرده بودند.

سربازان با عزم راسخ، دیگ آبی را جوشانده و با احتیاط سه سنگ را درون آن انداختند. اهالی دهکده که شگفت‌زده شده‌ بودند به تماشای آن‌ها آمدند.

یکی از سربازان گفت «این سوپ سنگ است». روستاییان پرسیدند «آیا محتویات آن فقط همین است؟» سرباز پاسخ داد «بدون شک – اگرچه، بعضی‌ها می‌گویند که تعدادی هویج مزه‌اش را بهتر خواهد کرد...» یکی از روستاییان سریع فرار کرد و بلافاصله با سبدی از هویج‌هایی که انبار کرده بود برگشت.

چند دقیقه بعد دوباره اهالی روستا پرسیدند «همین کافی است؟»

سرباز گفت «خب، چند عدد سیب زمینی هم آن را بهتر و خوشمزه‌تر خواهد کرد.» دوباره سریع یکی از اهالی دوید تا سیب‌زمینی تهیه کند.

در طول یک ساعت بعد، سربازان محتویات بیشتری را مطرح نمودند که باعث بالاتر بردن کیفیت سوپ می‌شد؛ شامل: گوشت گوساله، تره‌فرنگی، نمک و سبزیجات. هر بار یکی از روستاییان سریع می‌دوید و بخشی از ذخایرش را برای سربازان می‌آورد.

در پایان کار، آن‌ها موفق شدند که دیگی بزرگ از سوپ آب پز تهیه کنند. سربازها سنگ‌ها را از سوپ بیرون کشیدند و سپس با تمامی اهالی دهکده دور هم نشسته و از اولین سوپ کاملی که ماه‌ها بود هیچ‌کدام طعام نکرده بودند، لذت بردند.

در اینجا چند نتیجه‌ی اخلاقی در این داستان وجود دارد. اهالی دهکده توسط سربازها فریب خوردند از آن جهت که سربازها از کنجکاوی مردم روستا برای دستیابی به غذا استفاده کردند. اما مهمتر از آن، سربازها مانند واسط عمل کرده و اهالی روستا رو دور هم جمع کردند تا مشترکاً بتوانند چیزی را تولید کنند که اهالی به تنهایی از انجام آن عاجز بودند—نتیجه‌ای حاصل از همکاری. در واقع نتیجه برد-برد بود.

زمان‌هایی وجود دارند که شاید بخواهید از این عمل سربازها تقلید کنید. ممکن است در موقعیتی باشید که دقیقاً می‌دانید چه کاری باید انجام شود و چگونه باید انجام گردد. کل سیستم مقابل دیده‌گانتان است—می‌دانید که انجام چه کاری درست است. اما به محض اینکه می‌خواهید رشته‌ی کل امور را در دست بگیرید، با وقفه‌ها و نگاه‌های خیره و حیران مواجه می‌شوید. افراد، هیئت‌های بررسی را تشکیل می‌دهند، بودجه نیازمند تائید است، اوضاع پیچیده گشته و همه می‌خواهند از دارایی‌هایشان محافظت کنند. گاهی این را «واماندگی سرآغاز» ‌گویند.

وقت آن است که سنگ‌ها را برملا کنید. ببینید چه درخواستی منطقی به نظر می‌رسد. به خوبی به آن پر و بال دهید. بعد از آنکه اوضاع کاملا تحت کنترلتان در آمد، آن را به مردم نشان داده و بهت‌زده‌شان نمایید. سپس با گفتن «البته، اگر .... را هم به آن اضافه کنیم بهتر خواهد شد.»، وانمود کنید که نبود آن اهمیت چندانی هم ندارد. سپس با کمال خونسردی عقب نشسته و منتظر بمانید تا خودشان عملکردی را که مد نظرتان است از شما تقاضا کنند. مردم پیوستن به موفقیت‌های پیش‌رو را آسانتر می‌یابند. به آن‌ها چشم‌اندازی از آینده را نشان دهید آنگاه خواهید دید که اطرافتان را احاطه می‌کنند.[[7]](#footnote-7)

نکته ۵

واسطه‌گر تغییر باشید.

از جنبه‌ی اهالی دهکده

همچنین از سوی دیگر، داستان سوپ سنگ داستانی راجع به نیرنگ تدریجی و همراه با ملایمت است؛ داستانی است درباره‌ی تمرکز بسیار قوی. تمرکز کل اهالی دهکده فقط روی سنگ‌ها است و در آن لحظه هیچ چیز دیگری در دنیا برایشان حائز اهمیت نیست. ما همه هر روز در دام اینگونه حقه‌ها گرفتار می‌شویم. بدون آنکه بدانیم، چیزهای مختلفی حواسمان را جلب خود می‌کنند.

همگی، اثرات آن را نیز دیده‌ایم. پروژه‌ها به تدریج و با بیرحمی، کاملا از کنترل خارج می‌شوند. بسیاری از فجایع نرم‌افزار، در ابتدا آنقدر کوچک‌اند که قابل توجه نیستند و بیشتر هزینه‌های تحمیلی پروژه‌ها، همه به یکباره در یک روز اتفاق می‌افتند. خصوصیات سیستم‌ها، ویژگی به ویژگی تغییر جهت می‌یابند درحالی‌که از سوی دیگر، مدام کدها وصله و پینه می‌شوند تا این‌که هیچ اثری از کدهای بدیع آغازین باقی نمی‌ماند. اغلب، انباشته شدن همین چیزهای کوچک است که تیم‌ها و روحیه‌ی افراد را از هم می‌پاشد.

نکته ۶

چشم‌انداز کلی را از یاد نبرید.

صادقانه می‌گویم که ما هرگز چنین چیزی را امتحان نکرده‌ایم اما می‌گویند که اگر قورباغه‌ای را درون آب‌جوش بیندازید، سریع بیرون می‌پرد و به همان جایی که قبلا بوده برمی‌گردد. با اینحال، اگر شما قورباغه را در یک ظرف آب سرد گذاشته و به‌تدریج آن را حرارت دهید، قورباغه هرگز متوجه افزایش دمای تدریجی نمی‌شود و آنقدر در آب می‌ماند تا بپزد.

به یاد داشته باشید که مسئله‌ی قورباغه در اینجا با مسئله‌ی پنجره‌ی شکسته در بخش ۲ متفاوت است. در نظریه‌ی پنجره‌ی شکسته، مردم اراده‌شان را برای مبارزه با بی‌نظمی از دست می‌دهند زیرا پی برده‌اند که هیچ کس دیگری به نظم اهمیتی نمی‌دهد؛ اما قورباغه اصلا متوجه تغییر نمی‌شود.

به سان قورباغه نباشید. تمرکزتان همیشه روی چشم‌انداز کلی باشد. مدام آنچه در اطرافتان رخ می‌دهد را

بررسی کنید نه آنچه شخصا در حال انجامش هستید.

بخش‌های مرتبط:

* بی‌نظمی نرم‌افزار
* برنامه‌نویسی تصادفی
* فاکتورگیری مجدد
* فرورفتگی ضروری
* تیم‌های عملگرا

چالش‌ها

* «جان لیکوس»[[8]](#footnote-8)، درحالیکه مشغول بررسی مجدد پیش نویس همین کتاب بود، این مسئله را عنوان کرد: سربازها به صورت تدریجی اهالی دهکده را فریب می‌دهند، اما تغییری که آن‌ها واسطش هستند به نفع همه تمام می‌شود. با اینحال، وقتی که قورباغه را به صورت تدریجی حرارت می‌دهید، به آن آسیب می‌رسانید. آیا می‌توانید تعیین کنید هنگامی که واسطه‌گر تغییری هستید از کدام یک از دو نوع سوپ سنگ یا قورباغه استفاده کرده‌اید؟ آیا این داوری حاصل قضاوت ذهنی است یا قضاوت عینی؟

1. نرم‌افزاهای نسبتا خوب

در تلاش برای بهتر شدن، اغلب هر آنچه که خوب است را هم نابود می‌کنیم.

- شاه لیر[[9]](#footnote-9) ۱.۴

لطیفه‌ای قدیمی (نسبتا قدیمی) درباره‌ی یکی از شرکت‌های آمریکایی وجود دارد که می‌گوید این شرکت ۱۰۰،۰۰۰ مدار مجتمع را به تولیدکنندگان ژاپنی سفارش می‌دهد. این سفارش، با توجه به نرخ مدارهای معیوب تخصیص یافته بود: یک تراشه در هر ۱۰،۰۰۰. چندهفته بعد سفارش از راه رسید: یک جعبه‌ی بزرگ حاوی هزاران مدار مجتمع و یک جعبه‌ی کوچک شامل فقط ۱۰ مدار. روی جعبه‌ی کوچک برچسبی تحت این عنوان بود: «این‌ها تراشه‌های معیوب هستند.»

ای کاش ما هم تا این حد روی کیفیت کنترل داشتیم. اما دنیای واقعی به ما این امکان را نمی‌دهد که نرم‌افزارهای به راستی بی‌نقص و بالاخص بدون ایراد را به تعداد فراوان تولید کنیم. زمان، فن‌آوری و زود تغییری، همگی بر علیه ما هم قسم شده‌اند.

با این‌حال، لازم نیست که این موضوع ما را نا‌ امید کند. همان‌گونه که «اِد یوردن» در [مقاله‌ای نرم‌افزاری](http://www.amazon.com/Structured-Design-Fundamentals-Discipline-Computer/dp/0138544719) در IEEE[[10]](#footnote-10) عنوان کرده است، شما می‌توانید خودتان را به نوشتن نرم‌افزاری نسبتا خوب مقید کنید -- نسبتا خوب برای کاربرانتان، برای کسانی که در آینده آن را نگهداری می‌کنند و برای آرامش روحی خودتان. آن‌گاه در‌می‌یابید که به مراتب بازدهیتان بالاتر رفته و کاربرانتان راضیتر و خوشحالتر خواهند بود. همچنین، ممکن است به خوبی دریابید که برنامه‌هایتان برای تکامل کوتا‌ه‌ مدتتر آن‌ها درواقع بهتر است.

قبل از اینکه پیشتر برویم، باید مشخص کنیم که در چه زمینه‌ای قرار است حرفی برای گفتن داشته باشیم. اصلاح «نسبتا خوب» به معنی کدهای ناشیانه و ضعیف نیست. هر سیستمی برای این‌که موفق باشد باید نیازهای کاربرانش را تامین کند. منظور ما صرفا این است که به کاربران فرصتی داده شود تا در فرآیند تصمیم‌گیری در مورد این‌که چه‌موقع آنچه شما تولید کرده‌اید به شرایط مطلوب رسیده است، مشارکت نمایند.

1. Pragmatic [↑](#footnote-ref-1)
2. Pragmaticus [↑](#footnote-ref-2)
3. Entropy [↑](#footnote-ref-3)
4. Software Rot [↑](#footnote-ref-4)
5. منظور همان کامنت‌گذاری در برنامه‌نویس است. [↑](#footnote-ref-5)
6. Andy [↑](#footnote-ref-6)
7. وقتی که این کار را انجام می‌دهید ممکن است با جمله‌ی دریاسالار دکتر گریس هاپر (Grace Hopper) مواجه شوید که می‌گوید: «گفتن ببخشید از گفتن اجازه هست، آسانتر است.» [↑](#footnote-ref-7)
8. John Lakos [↑](#footnote-ref-8)
9. King Lear، از نمایشنامه‌های تراژیک نوشته‌ی ویلیام شکسپیر. [↑](#footnote-ref-9)
10. یک سازمان بین‌المللی حرفه‌ای و ناسودبر است. خواست این مؤسسه کمک به پیش‌برد فناوری به طور گسترده و حوزه‌های وابسته به مهندسی برق و کامپیوتر و هم‌چنین زمینه‌های وابسته به طور خاص است. این مؤسسه بیش از ۴۰۰ هزار نفر عضو در ۱۶۰ کشور جهان دارد که ۴۵ درصد این اعضا خارج از ایالات متحده هستند. [↑](#footnote-ref-10)