1

به دو دلیل شما در حال خواندن این کتاب هستید: یک: شما یک برنامه نویسید. دو: می خواهید برنامه نویس بهتری شوید. خوب است. ما به برنامه نویس های بهتری نیاز داریم.

این یک کتاب در مورد برنامه نویسی خوب است. این کتاب با کد پر شده است. ما به کد از جنبه های مختلفی نگاه خواهیم کرد.

از دیدگاه بالا به آن نگاه میکنیم، از دیدگاه پایین، و از درون به بیرون آن را بررسی میکنیم. وقتی کار ما تمام شد، در مورد کد اطلاعات زیادی داریم. به علاوه، قادر خواهیم بود تفاوت میان کد خوب و کد بد را بیان کنیم. ما خواهیم دانست که چگونه یک کد خوب بنویسیم. و خواهیم دانست که چگونه یک کد بد را به یک کد خوب تبدیل کنیم.

کد وجود خواهد داشت

ممکن است شخصی بگوید یک کتاب در مورد کد به نوعی مربوط به زمان های گذشته است. کد دیگر مسئله اصلی نیست. در عوض ما باید در مورد مدلها و نیازمندیها نگران باشیم. در واقع بعضی از افراد پیشنهاد کرده اند که ما به پایان دوران کدنویسی نزدیک هستیم. به زودی تمام کدها به جای نوشته شدن، تولید خواهند شد. برنامه نویسان دیگر مورد نیاز نخواهند بود زیرا تجار برنامه را از مشخصات[1] تولید خواهند کرد.

چرت است! ما هیچگاه از کدها خلاص نخواهیم شد، زیرا کد جزییات نیازمندی ها را نشان میدهد. در بعضی سطح ها آن جزییات نمی توانند نادیده گرفته شوند یا کلی نگری(انتزاع)[2] شوند. آنها باید مشخص باشند. و مشخص کردن نیازمندی ها با چنان جزییاتی که یک ماشین بتواند آنها را اجرا کند *برنامه نویسی* است. چنین مشخصه ای کد است.

من انتظار دارم که سطح تجرید زبان ما به گسترش خود ادامه دهد. من همچنین انتظار دارم که تعداد زبان های مختص دامنه[3] رشد پیدا کنند. این مسئله خوب است. اما این باعث حذف کد نمی شود. در واقع تمام مشخصات نوشته شده در سطوح

---

[1] specification
[2] abstract
[3] Domain-specifid

بالاتر و با زبان مختص دامنه، "کد " خواهند بود! این کد همچنان نیاز دارد که سختگیرانه، دقیق، و بسیار رسمی و جزیی باشد تا یک ماشین بتواند آن را بفهمد و اجرا کند.

گروهی که فکر میکنند که یک روز کد میتواند ناپدید شود شبیه به ریاضی دانانی هستند که امید دارند یک روز یک تعریف یا فرمول ریاضی را کشف کنند که نیاز نداشته باشد که رسمی باشد. آنها امید دارند که یک روز راهی برای ساختن ماشینهایی کشف کنند که میتواند چیزی را که ما میخواهیم به جای چیزی که میگوییم انجام دهند. این ماشینها باید قادر باشند که آنقدر خوب ما را بفهمند که نیازمندیهای خاص و مبهم را به برنامه های کاملا قابل اجرا که با دقت آن نیازمندی ها را مرتفع میکنند، ترجمه کنند.

این اتفاق هیچگاه نمی افتد. حتی انسان ها با تمام بینش و خلاقیتشان قادر به ساخت موفق سیستم ها از احساسات مبهم مشتریان نیستند. در واقع اگر نظم نیازمندیهای مشخصات چیزی به ما آموخته باشد، آن این است که نیازمندی های دقیق توصیف شده به اندازه کد رسمی هستند و می توانند به عنوان تست های قابل اجرایی از کد عمل کنند!

به خاطر داشته باشید که کد زبانی است که در نهایت با آن نیازمندیها را بیان میکنیم. ممکن است ما زبانهایی بسازیم که به نیازمندی ها نزدیکتر باشند. ممکن است ما ابزارهایی بسازیم که به ما در parse کردن و سرهمبندی این نیازمندیها با ساختارهای رسمی کمک کنند. اما ما هرگز دقت لازم را حذف نمیکنیم. بنابراین همیشه کد وجود دارد.

کد بد

اخیرا یک پیشگفتار از کتاب
نوشته Kent Beck خواند
مبنای یک فرضیه ضعیف بنا
است ..." یک فرضیه ضعیف
فرضیه یکی از بیشمار فرض
و بار زیاد در حرفه ماست
ما می دانیم که کد خوب م
بودیم با نبود آن کنار بیاییم

من یک شرکت را میشناس
killer نوشته است. این بر
از افراد حرفه ای آن را خ
چرخه انتشار شروع به کش
بعدی باگها درست نشدند.
crash افزایش یافت. من
محصول را با ناامیدی پاک
نکردم. مدت کمی پس از ان، ان شرکت از صحنه تجارت خارج شد.

دو دهه بعد من یکی از اولین کارکنان آن شرکت را دیدم و از او پرسیدم که چه اتفاقی افتاد. آن پاسخ، ترس های من را تایید کرد. آنها در عرضه محصول به بازار عجله کرده بودند و حجم عظیمی از کد را درست کرده بودند. هرچه آنها ویژگی های بیشتر و بیشتری اضافه کردند، کد بدتر و بدتر شد تا زمانی که دیگر آنها نتوانستند آن را مدیریت کنند. *این یک کد بد بود که آن شرکت را نابود کرد.*

آیا تا کنون توسط یک کد بد به مشکل خورده اید؟ اگر شما یک برنامه نویس با هر سطح از تجربه باشید آنگاه شما در مواقع زیادی این مانع را احساس کردید. در واقع ما یک نام برای آن

داریم. ما آن را به سختی راه رفتن[4] صدا میکنیم- ما در طول یک کد بد به سختی راه میرویم. ما در باتلاقی از خاربن های در هم تنیده و تله های مخفی تقلا میکنیم- ما برای یافتن مسیر، به امید چند اشاره، چند سر نخ از اینکه چه اتفاقی در شرف وقوع است تقلا میکنیم اما تمام چیزی که میبینیم کدهای بی معنی بیشتر و بیشتر است.

قطعا شما با کدهای بد به مشکل برخورده اید. خب چرا آنها را نوشتید؟

می خواستید سریع پیش بروید؟ عجله داشتید؟احتمالا بله. احتمالا شما احساس کردید که زمان کافی برای انجام یک کار خوب ندارید؛ رئیس شما از شما عصبانی میشود اگر زمانی را برای تمیز کردن کدتان هدر دهید. احتمالا شما از کار کردن بر روی این برنامه خسته شده بودید و میخواستید تمام شود. یا شاید شما به کارهای ناتمام بقیه کارمندانی که به آنها قول داده بودید که کار را انجام می دهید نگاه کردید و متوجه شدید که نیاز دارید تا ماژولها را به یکدیگر بچسبانید تا بتوانید کارها را به مرحله بعد ببرید. همه ما این کارها را انجام دادیم.

همه ما به ریخت و پاشی که درست کرده بودیم نگاه کردیم و سپس انجام آن را به روز دیگر موکول کردیم. همه ما آسودگی ناشی از کار کردن برنامه نامرتب خودمان را حس کردیم و تصمیم گرفتیم که انجام کار شلخته بهتر از هیچ کاری است.

---

[4] wading

همه ما گفتیم بعدا که برمیگردیم، آن را تمیز میکنیم. قطعا در آن روزها ما قانون LeBlanc را نمیشناختیم : *بعدا یعنی هیچوقت.*

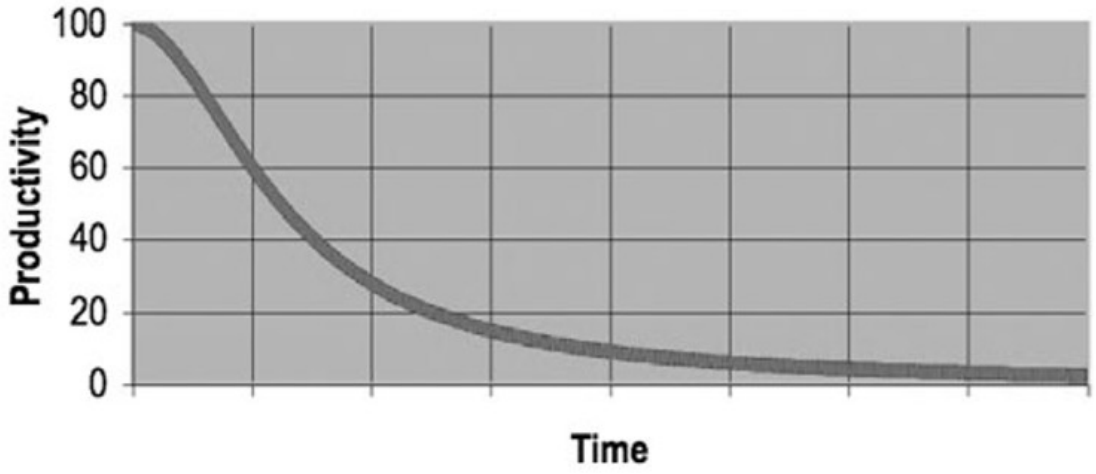
هزینه کلی مالک یک شلختگی بودن

اگر بیش از دو یا سه سال است که برنامه نویس بوده اید ، احتمالاً با کد کثیف شخص دیگری معطل مانده اید. میزان معطل بودن می تواند قابل توجه باشد. طی یک بازه زمانی یک یا دو ساله، تیم هایی که در ابتدای یک پروژه بسیار سریع در حال حرکت بودند ، خود را در حال حرکت با سرعت حلزونی می بینند. هر تغییری که در کد ایجاد کنند ، دو یا سه قسمت دیگر کد را خراب می کند. تغییر ندادن مهم است. هرگونه افزودن یا تغییر در سیستم مستلزم این است که گره خوردگی ها، پیچ و تاب ها و مشکلات "درک شوند". تا بتوانید تعداد بیشتری گره و پیچ و تاب اضافه کنید. با گذشت زمان شلختگی بسیار بزرگ و عمیق و بسیار بلند می شود ، به گونه ای که نمی توان آن ها را تمیز کرد. اصلاً راهی نیست.

با ساخته شدن شلختگی، بهره‌وری تیم همچنان رو به کاهش می رود، و در نهایت به صفر نزدیک می شود. با کاهش بهره وری ، مدیریت تنها کاری که می تواند را انجام می دهد؛ آنها به امید افزایش بهره وری کارکنان بیشتری را به این پروژه اضافه می کنند. اما آن کارکنان جدید سیستم را نمی شناسند. آنها تفاوت بین تغییری را که منطبق با هدف طراحی انجام میشود و تغییری که هدف طراحی را

ناکارآمد می کند، نمی دانند. علاوه بر این ، آنها و هر کس دیگری که در تیم حضور دارند ، تحت فشارهای هولناکی برای افزایش بهره وری قرار دارند. بنابراین همه آنها بیشتر و بیشتر باعث شلختگی می شوند و باعث می شوند که بازدهی هر چه بیشتر به سمت صفر برسد.

(شکل 1-1 را ببینید.)



تصویر 1-1 بهره وری در برابر زمان

طراحی مجدد بزرگ در SKY

سرانجام تیم شورش می کند. آنها به مدیریت اطلاع می دهند که نمی توانند بر مبنای این کد نفرت انگیز محصولی را توسعه دهند. آنها خواستار طراحی مجدد هستند. مدیریت نمی خواهد که منابع را صرف طراحی مجدد پروژه کند ، اما آنها نمی توانند انکار کنند که بهره وری وحشتناک است. سرانجام آنها به خواسته های توسعه دهندگان تن می دهند و اجازه انجام طراحی مجدد بزرگ در sky را می دهند.

یک تیم جدید قوی انتخاب شده است. همه دوست دارند در این تیم حضور داشته باشند زیرا این یک پروژه Greenfield است. آنها دوباره شروع به کار می کنند و چیزی زیبا را ایجاد می کنند. اما فقط بهترین ها و درخشان ترین ها برای تیم قوی انتخاب می شوند. همه افراد دیگر باید به حفظ سیستم فعلی ادامه دهند.

اکنون دو تیم در رقابت هستند. تیم قوی باید سیستم جدیدی بسازد که هر آنچه را که سیستم قدیمی انجام می دهد انجام دهد. نه تنها این ، آنها باید با تغییراتی که به طور مداوم در سیستم قدیمی انجام می شود ، به روز باشند. مدیریت تا زمانی که سیستم جدید نتواند کارهایی را که سیستم قدیمی انجام می دهد، انجام دهد، سیستم جدید را جایگزین سیستم قدیمی نخواهد کرد.

این رقابت می تواند برای مدت زمان طولانی ادامه یابد. من دیده ام که 10 سال طول کشید. و زمانی که این کار انجام شد، اعضای اصلی تیم قوی مدتها بود که از تیم رفته بودند ، و اعضای فعلی خواستار طراحی مجدد سیستم جدید هستند زیرا این سیستم بسیار شلخته است.

اگر حتی یک بخش کوچک از داستانی را که تعریف کردم تجربه کرده اید ، پس می دانید که صرف وقت برای تمیز کردن کد خود صرفاً مقرون به صرفه نیست. این یک کار حرفه ای برای بقا است.

نگرش

آیا تا به حال آنقدر با شلختگی ها کلنجار رفته اید که کاری را که در طی چند ساعت می توانستید انجام دهید، هفته ها طول بکشد؟ آیا دیده اید که چیزی که

میتواست با تغییر یک خط ایجاد شود ، در عوض در تغییر در صدها ماژول مختلف انجام شد؟ این علائم بسیار شایع است.

چرا این اتفاق برای کد می افتد؟ چرا کد خوب خیلی سریع به کد بد تبدیل میشود؟ توضیحات زیادی برای آن داریم. ما شاکی هستیم که نیازمندی ها به گونه ای تغییر یافته که طرح اصلی را خنثی می کند. ما ناراحتیم که برنامه ها برای انجام درست کارها بسیار فشرده بود. ما در مورد مدیران احمق و مشتریان عجول و انواع بازاریابی های بی فایده و ضد عفونی کننده های تلفنی صحبت می کنیم. اما Dilbert عزیز ، مشکل در سرنوشت ما نیست ، بلکه در خودمان است.

ما غیر حرفه ای هستیم.

ممکن است پذیرش این حقیقت تلخ، مشکل باشد. چطور ممکن است این شلختگی تقصیر ما باشد؟در مورد نیازمندی ها چطور؟ در مورد برنامه چطور؟ در مورد مدیران احمق و انواع بی فایده بازاریابی چطور؟ آیا آنها مقصر نیستند؟

خیر. مدیران و بازاریابان برای دریافت اطلاعاتی که برای وعده ها و تعهدات لازم دارند ، به ما نگاه می کنند. و حتی وقتی آنها به ما نگاه نمی کنند ، ما نباید از گفتن آنچه که فکر می کنیم شرمنده باشیم. کاربران به ما نگاه می کنند تا تصدیق کنند که ما چگونه سیستم را با نیازمندی ها پر میکنیم. مدیران پروژه از ما انتظار دارند که به پیش برد برنامه کمک کنیم.

ما عمیقا درگیر برنامه ریزی پروژه هستیم و مسئولیت هرگونه خرابی را به عهده میگیریم. به خصوص اگر این شکستها مربوط به کد بد باشند!

"اما صبر کنید!" شما بگویید. "اگر آنچه را که مدیر من می گوید ، انجام ندهم ، اخراج می شوم." احتمالا نه. اکثر مدیران حقیقت را می خواهند ، حتی اگر چنین رفتار نکنند. اکثر مدیران کدهای خوبی می خواهند ، حتی وقتی در مورد برنامه وسواس دارند. آنها ممکن است با شور و شوق از برنامه ها و نیازمندی ها دفاع کنند؛ اما این کار آنهاست *این وظیفه شماست* که با اشتیاق برابر، از کد دفاع کنید.

برای اینکه این مسئله برای شما جا بیفتد ، چه می شود اگر شما یک پزشک باشید و یک بیمار داشته باشید که به این دلیل که شستن دستها قبل از عمل جراحی زمان زیادی میبرد، خواستار جلوگیری از این کار احمقانه باشد ؟[5] به وضوح بیمار رئیس است. و با این وجود پزشک کاملاً باید از انجام عمل خودداری کند. چرا؟ زیرا پزشک بیشتر از بیمار از خطرات بیماری و عفونت اطلاع دارد. این که پزشک مطابق با خواست بیمار عمل کند (حتی اگر مجرمانه نباشد) غیرحرفه ای است .

به همین ترتیب غیر حرفه ای است که برنامه نویسان به خواسته مدیرانی که ریسک ایجاد شلختگی ها را نمی فهمند تن در دهند.

مسئله بغرنج اصلی

برنامه نویسان با مجموعه ای از مسائل پایه ای روبرو هستند. همه توسعه دهندگان با بیش از چند سال تجربه می دانند که شلختگی های قبلی آنها را کند می کند. و با این وجود همه توسعه دهندگان فشار ناشی از ایجاد شلختگی را احساس می کنند زیرا باید به deadline برسند. به طور خلاصه ، آنها وقت لازم را برای حرکت سریع ندارند!

حرفه ای ها می دانند که قسمت دوم این مسئله اشتباه است. شما با ایجاد شلختگی به deadline نمیرسید. در واقع ، شلختگی فورا شما را کند می کند و شما را مجبور می کند که deadline را از دست بدهید. *تنها* راه برای رسیدن به deadline- تنها راه سریع پیشبرد کار- این است که همیشه کد را تا حد ممکن تمیز نگه دارید.

هنر کد تمیز؟

---

[5] هنگامی که برای اولین بار در سال 1847 توسط Ignaz Semmelweis به پزشکان توصیه شد دستهای خود را بشویند ، این توصیه به دلیل اینکه سر پزشکان خیلی شلوغ بود و وقت نداشتند میان ویزیت بیماران دستهای خود را بشویند رد شد.

بیایید بگوییم شما معتقد هستید که کد کثیف مانع قابل توجهی است. بیایید بگوییم که شما می پذیرید که تنها راه پیشبرد سریع، تمیز نگه داشتن کدهایتان است. سپس باید از خود بپرسید: "چگونه می توانم کد تمیز بنویسم؟" اگر نمی دانید تمیز بودن کد چیست ، تلاشتان برای نوشتن کد تمیز خوب نیست!

خبر بد این است که نوشتن کد تمیز بسیار شبیه به نقاشی کشیدن است. بیشتر ما می دانیم چه زمانی یک تصویر خوب یا بد نقاشی شده است. اما قدرت تشخیص هنر خوب از بد به معنای این نیست که بلدیم چگونه نقاشی کنیم. بنابراین قدرت تشخیص کد تمیز از کد کثیف به این معنی نیست که می دانیم چگونه می توان کد تمیز نوشت!

نوشتن کد تمیز مستلزم استفاده منظم از تکنیک های ظریف بی شماری است که این تکنیک ها از طریق به کارگیری حس"پاکیزگی" بکار برده می شود. این "حس کردن کد" مهم است. برخی از ما با آن متولد می شویم. برخی از ما باید برای به دست آوردن آن بجنگیم. نه تنها این امکان را به ما می دهد که ببینیم آیا کد خوب

است یا بد ، بلکه برای تبدیل کد بد به کد تمیز، استراتژی استفاده از قوانین مان را به ما نشان می دهد.

به طور خلاصه ، یک برنامه نویس که کد تمیز می نویسد ، هنرمندی است که می تواند یک صفحه خالی بگیرد و از طریق یک سری دگرگونی ها آن را به یک سیستم با کدگذاری زیبا تغییر دهد.

کد تمیز چیست؟

احتمالاً به اندازه برنامه نویسها تعریف وجود دارد. بنابراین از برخی از برنامه نویسان بسیار شناخته شده و با تجربه پرسیدم که چه فکر می کنند.

Bjarne Stroustrup مخترع زبان C++ و نویسنده *The C++ Programming Language*

من دوست دارم کد من زیبا و کارآمد باشد. منطق باید سر راست باشد تا پنهان کردن باگ ها دشوار باشد ، وابستگی های حداقلی برای سهولت در نگهداری ،کنترل کامل خطا مطابق با یک استراتژی تکه به تکه و عملکرد نزدیک به بهینه، بگونه ای که مردم را وسوسه نکند تا کد را با بهینه سازی های غیر اصولی شلخته کنند. کد تمیز یک کار را به خوبی انجام می دهد.

Bjarne از کلمه “elegan t”[6] استفاده میکند. این کلمه کامل است! دیکشنری موجود در MacBook من این تعریف را برای این کلمه ارائه میدهد : *از نظر ظاهری و رفتاری دلپذیر و برازنده و شیک است[7]. کاملاً مبتکرانه و ساده.* به کلمه دلپذیر دقت کنید. ظاهرا Bjarne فکر می کند که خواندن کد تمیز لذت بخش است. خواندن آن باید لبخند را به لبان شما بیاورد، همانگونه که یک جعبه موسیقی خوش ساخت و یا یک ماشین با طراحی زیبا باعث میشود لبخند بزنید.

---

[6] زیبا

[7] **Pleasingly graceful and stylish in appearance or manner; pleasingly ingenious and simple**

Bjarne همچنین دو بار از واژه بازده[8] استفاده می کند. شاید وقتی این کلمه از دهان مخترع C ++ خارج میشود، غافلگیر کننده نباشد. اما فکر می کنم چیزی فراتر از تمایل صرف برای سرعت وجود دارد. چرخه های تلف شده لذت بخش نیستند و ناخوشایند هستند.

و اکنون به کلماتی که Bjarne برای توصیف پیامد آن ناهنجاری استفاده می کند توجه داشته باشید. او از کلمه "وسوسه"[9] استفاده می کند. در اینجا یک حقیقت عمیق وجود دارد. کد بد شلختگی را *وسوسه* میکند تا رشد کند! وقتی دیگران کد بد را تغییر می دهند ، متمایل به بدتر کردن آن هستند.

Dave Thomas عملگرا و Andy Hunt این نکته را با یک روش متفاوت می گویند. آنها از استعاره پنجره های شکسته[10] استفاده کرده اند. وقتی بنایی پنجره

[8] efficiency
[9] tempt
[10] **http://www.pragmaticprogrammer.com/booksellers/2004-12.html**

های شکسته دارد اینظور به نظر می رسد که کسی به آن اهمیتی نمی دهد. بنابراین افراد دیگر هم از توجه به آن خودداری می کنند. آنها اجازه می دهند تا پنجره های بیشتری شکسته شود. سرانجام آنها به طور جدی آنها را می شکنند. آنها با گرافیتی نما را خراب می کنند و اجازه میدهند در آن جا زباله جمع شود. یک پنجره شکسته روند زوال را شروع می کند.

Bjarne همچنین خاطرنشان می کند که رسیدگی به خطا باید کامل باشد. این مربوط به قانون توجه به جزییات می شود. کنترل خطای مختصر فقط یکی از راه هایی است که برنامه نویسان با استفاده از آن به تفصیل جزئیات می پردازند. نشت حافظه یکی دیگر از راهها و شرایط مسابقه راه دیگر است. راه دیگر نامگذاری متناقض است. نتیجه اصلی این است که کد تمیز توجه زیادی به جزئیات دارد.

Bjarne با این ادعا که کد تمیز یک کار را به خوبی انجام می دهد ، بحث را می بندد. تصادفی نیست که بسیاری از اصول طراحی نرم افزار وجود دارند که می توانند دلیل اصلی این توصیه ساده باشند. نویسندگان یکی پس از دیگری سعی در برقراری ارتباط با این اندیشه داشتند. کد بد بیش از حد خرابکاری می کند، تمایلات را زشت کرده و اهداف را مبهم می کند. کد تمیز *متمرکز* است. هر تابع، هر کلاس، هر ماژول یک نگرش تک ذهنیتی را که کاملاً دست نخورده و آلوده نشده باقی مانده است با جزئیات پیرامون خود در معرض نمایش قرار می دهد.

Grady Booch ، نویسنده "Object Oriented analysis and Design with Applications"

کد تمیز ساده و سر راست است. کد تمیز مثل یک نثر خوب نوشته شده است. کد تمیز هرگز هدف طراح را مبهم نمی کند بلکه پر از انتزاعات واضح و خطوط کنترل سر راست است.

Grady برخی از نکاتی را که Bjarne بیان می کند، عنوان میکند، اما او یک جنبه خوانایی[11] را در نظر می گیرد. من خصوصاً این نظر او را که کد تمیز باید مانند نثر خوب نوشته شده باشد دوست دارم. دوباره به کتاب خوبی که مطالعه کرده اید فکر کنید. به یاد آورید که چگونه کلمات ناپدید شدند تا تصاویر جایگزین شوند!

[11] Readability

مثل تماشای فیلم بود ، اینطور نیست؟ بهتر! شخصیت ها را دیدید ، صداها را شنیدید ، ترحم و طنز را تجربه کردید.

خواندن کد تمیز هرگز شبیه به خواندن کتاب ارباب حلقه ها نخواهد بود. با این وجود استعاره ادبی بد نیست. مانند یک رمان خوب ، کد تمیز باید به وضوح تنش های موجود در مشکل را حل کند. باید آن تنش ها را به اوج خود برساند و سپس به خواننده بگوید که: "آها! اینه!" زیرا مشکلات و تنش ها در ظهور یک راه حل واضح برطرف می شود.

به نظر من استفاده grady از عبارت "انتزاع خشک"[12] به عنوان یک کلمه ضد و نقیض جذاب است! در نهایت، کلمه "خشک"[13] تقریباً مترادف "واقعی"[14] است. فرهنگ لغت مک بوک من تعریف زیر از "خشک" را دارد: *قاطعانه و سرنوشت ساز ، بدون تردید و جزئیات غیر ضروری.* علیرغم این کنار هم گذاشتن معانی، کلمات دارای پیام قدرتمندی هستند. کد ما باید برخلاف حدس و گمان ها و واقعی باشد. کد باید فقط شامل چیزهای مهم و ضروری باشد. خوانندگان ما باید قاطعیت ما را درک کنند.

آقا Dave Thomas، موسس OTI، پدرخوانده استراتژی Eclipse

کد تمیز میتواند بجز نویسنده اصلی آن، توسط یک توسعه دهنده دیگر نیز خوانده شود و بهبود یابد. این کد Unit test و Acceptance test دارد. این کد اسامی معنی دار دارد. کد تمیز به جای اینکه راههای زیادی برای انجام یک

t

[12] Crisp abstraction
[13] crisp
[14] concrete

کار ارائه کند، یک راه برای انجام یک کار دارد. کد تمیز حداقل وابستگی ها ، که به طور واضح تعریف شده اند ، و یک API واضح و حداقلی را ارائه می دهد. کد باید دانا باشد زیرا بسته به زبان ، تمام اطلاعات لازم را نمی توان به طور مشخص و به تنهایی با کد بیان کرد.

Dave بزرگ، تمایل Grady برای خوانایی را با پیچیدگی مهمی به اشتراک می گذارد. Dave ادعا می کند که کد تمیز باعث میشود بهبود آن برای سایر افراد آسان باشد. این ممکن است واضح به نظر برسد ، اما نمی تواند بیش از حد مورد تأکید قرار بگیرد. از این گذشته ، میان کدی که خواندن آن آسان است و کدی که تغییر آن آسان است تفاوت وجود دارد.

Dave تمیز بودن کد را با تست ها پیوند می زند! ده سال پیش این امر باعث تعجب بسیار میشد. اما قوانین توسعه مبتنی بر تست[15] تأثیر عمیقی بر صنعت ما گذاشته و به یکی از اساسی ترین قوانین ما تبدیل شده است. حق با Dave است. کد ، بدون تست ، تمیز نیست. مهم نیست که چقدر زیبا باشد ، هر چقدر هم که قابل خواندن و در دسترس باشد ، اگر آزمایش نشده باشد ، کثیف است.

Dave دو بار از کلمه حداقل استفاده می کند. ظاهراً منظور او در این تعریف کدهای کوچک است. در واقع ، از زمان پیدایش ادبیات نرم افزار تا کنون، این یک ترجمه رایج بوده است. کوچکتر بهتر است.

Dave همچنین می گوید که کد باید دانا[16] باشد. این یک اشاره ریز به ادبیات برنامه نویسی Knuth[17] دارد. نتیجه کلی این است که کد باید به شکلی تهیه شود که برای انسانها خوانا باشد.

---

[15] Test Driven Development
[16] Liteate
[17] **[Knuth92]**

MIcheal Feathers، نویسنده Working Effectively with Legacy Code

من می توانم تمام خصوصیاتی را که در کد تمیز به آن توجه می کنم ذکر کنم ، اما کیفیت فرا معماری وجود دارد که بر تمام آنان ارجح است. همیشه به نظر می رسد که کد تمیز توسط کسی نوشته شده است که به آن اهمیت داده است. هیچ چیز واضحی وجود ندارد که بتوانید انجام دهید تا کد بهتر شود. به همه این موارد توسط نویسنده کد فکر شده است و اگر سعی دارید پیشرفت ها را تصور کنید ، به جایی که هستید برمیگردید، جایی که از کد شخصی که برای شما باقی گذاشته تشکر میکنید - کدی که توسط کسی که عمیقاً به این مهارت اهمیت می دهد به جا گذاشته شده است.

یک کلمه: اهمیت دادن[18]. این کلمه واقعاً موضوع این کتاب است. شاید یک عنوان مناسب این باشد که *چگونه به کد اهمیت دهیم.*

Micheal به مهمترین نکته اشاره کرد. کد تمیز کدی است که از آن مراقبت شده است. شخصی وقت خود را برای ساده و منظم نگه داشتن آن صرف کرده است. آنها توجه کافی به جزئیات داشته اند. آنها مراقبت کرده اند.

[18] care

Ron Jeffries ، نویسنده Extreme Programming Installed و Extreme Programming Adcenturs در C #

Ron حرفه برنامه نویسی خود را در Fortran در فرماندهی هوایی استراتژیک آغاز کرد و تقریباً به هر زبان و تقریباً بر روی هر دستگاهی ، کدی را نوشت. این امر باعث شد که در مورد صحبت کردنش بسیار مراقب باشد.

در سال های اخیر من قوانین کد ساده Beck را شروع و تقریباً به پایان رساندم. به ترتیب اولویت ، کد ساده:

- *تمام تست ها را اجرا می کند؛*
- *بدون Duplicate است.*
- *تمام ایده های طراحی موجود در سیستم را بیان می کند.*
- *تعداد موجودیت هایی چون کلاسها ، متدها ، توابع و موارد مشابه را به حداقل می رساند.*

از این میان ، بیشتر روی تکثیر[19] تمرکز می کنم. وقتی همین کار بارها و بارها انجام شد ، این نشانگر این است که ایده ای در ذهن ما وجود دارد که به خوبی در کد نمایش داده نشده است. سعی می کنم بفهمم که چیست. سپس سعی می کنم این ایده را با وضوح بیشتری بیان کنم.

برای من، قابلیت بیان[20] شامل اسامی معنادار است ، و من احتمالاً چندین بار اسامی چیزها را قبل از تثبیت آنها، تغییر می دهم. با ابزار مدرن کدنویسی مانند Eclipse ، تغییر نام کاملا بدون هزینه است ، بنابراین تغییر دادن برای من مشکل ساز نخواهد بود. با این وجود بیان کد فراتر از نامها است.

[19] duplication
[20] expressiveness

من همچنین به این موضوع نگاه می کنم که آیا یک شی یا متد بیش از یک کار را انجام می دهد یا نه. اگر یک شیء باشد ، احتمالاً باید به دو یا چند شیء تقسیم شود. اگر یک متد باشد ، من همیشه از refactoring Extract Method روی آن استفاده می کنم ، نتیجه اجرای این روش بر روی یک متد این است که چیزی که متد انجام میدهد را واضح تر بیان میکند، و برخی از زیرمتدها چگونگی انجام این کار را بیان میکنند.

تکثیر[21] و صراحت[22] ، برای رسیدن به چیزی که من آن را کد تمیز تلقی کنم، راه بسیار طولانی ای را طی می کنند و بهبود کد کثیف فقط با توجه به این دو مورد می تواند تفاوت بزرگی ایجاد کند. با این حال ، یک چیز دیگر وجود دارد که من از انجام آن آگاه هستم ، که توضیح آن کمی سخت تر است.

بعد از سالها انجام این کار ، به نظر من همه برنامه ها با عناصر بسیار مشابهی ساخته شده اند. یک مثال "یافتن چیزها در یک مجموعه" است. چه بانک اطلاعاتی از سوابق کارمندان ، یا نقشه درهم ساز[23] کلیدها و مقادیر ، و یا آرایه ای از بعضی از اقلام داشته باشیم ، ما اغلب خودمان را خواهان مورد خاصی از آن مجموعه میبینیم. در پی آگاهی از وقوع این اتفاق ، من اغلب پیاده سازی خاصی را در یک متد یا کلاس انتزاعی تر می پیچم. این به من چند مزیت جالب می دهد.

من اکنون می توانم آن عملکرد را با چیزی ساده پیاده سازی کنم ، مثلا با یک هش مپ، اما از آنجایی که اکنون همه ارجاعات مربوط به آن جستجو را توسط انتزاع کوچکم تحت پوشش قرار دادم ، می توانم هر زمان که بخواهم ، پیاده سازی را تغییر دهم. من بعدا می توانم با حفظ توانایی خود برای تغییر، به سرعت پیش بروم.

---

[21] duplication
[22] expressiveness
[23] Hash map

علاوه بر این، وقتی با چند روش نسبتا ساده می توانم همه چیزی که می خواهم را بیابم، مجموعه انتزاع اغلب توجه مرا به آنچه که "واقعاً" در جریان است ، جلب می کند و مرا از مسیر پیاده سازی مجموعه رفتارهای دلخواه منصرف می کند.

تکثیر کاهش یافته ، بیان واضح و ساخت انتزاعات ساده از ابتدا. این چیزی است که برای من کد تمیز می سازد.

در اینجا ، در چند پاراگراف کوتاه ، Ron مطالب این کتاب را خلاصه کرده است. بدون تکثیر ، یک چیز ، بیان ، تجریدهای کوچک. همه چیز آنجاست.

Ward Cunningham مخترع ویکی، مخترع Fit، هم اختراع کننده eXtreme Programming. نیروی انگیزشی در پشت Design Pattern. رهبر فکری شی گرایی و Smalltalk. پدرخوانده همه کسانی که به کد اهمیت میدهند.

وقتی هر روالی[24] که میخوانید دقیقا همان چیزی است که انتظار دارید، شما می دانید که دارید روی کد تمیز کار می کنید.
همچنین هنگامی که کد شبیه به زبانی که برای مشکل درست شده است ، می توانید آن را یک کد زیبا بنامید.

جمله هایی مانند این ، خصوصیات Ward است. شما آن را خوانده اید ، سر خود را تکان داده اید ، و سپس به موضوع بعدی رفته اید. منطقی به نظر می رسد، بطور واضح این مسئله به سختی به عنوان یک مسئله عمیق و ژرف درنظر گرفته میشود. ممکن است فکر کنید تقریباً همان چیزی بود که انتظار داشتید. اما بیایید نگاه دقیق تری داشته باشیم.

[24] Routine

" . . تقریباً آنچه انتظار داشتید. " آخرین باری که ماژولی را دیدید که تقریباً همان چیزی بود که انتظار داشتید کی بود؟ آیا به احتمال زیاد ماژول هایی که به آنها نگاه می کنید گیج کننده ، پیچیده و درهم و برهم نیستند؟ این قانون اشتباه نیست؟ آیا شما عادت نکردید که تلاش برای گرفتن و نگه داشتن تارهای استدلال که از کل سیستم به دست می آیند و راه خود را در ماژولی که می خوانید درست میکنند، به هیچ انگارید؟ آخرین باری که یک کد را خواندید و سر خود را به شکلی که Ward گفته است تکان دادید، کی بود؟

Ward انتظار دارد که وقتی کد تمیز را می خوانید به هیچ وجه تعجب نکنید. در واقع ، شما حتی تلاش زیادی هم نمی کنید. شما آن را خواهید خواند ، و تقریباً همان چیزی است که انتظار دارید. این امر آشکار ، ساده و قانع کننده خواهد بود. هر ماژول مقدمات را برای مرحله بعد تنظیم می کند. هر کدام به شما می گوید که بعدی چگونه نوشته خواهد شد. برنامه هایی که آن چنان تمیز هستند، به گونه ای عمقی و خوب نوشته شده اند که حتی متوجه آن نمی شوید. طراح باعث می شود مانند همه طرح های استثنایی ، این مسئله به طرز مسخره ای ساده به نظر برسد.

تفکر Ward در مورد زیبایی چطور؟ همه ما در برابر این واقعیت که زبانهای ما برای مشکلات ما طراحی نشده اند جبهه می گیریم. اما جمله Ward باری را بر دوش ما می گذارد. او می گوید که *کد زیبا باعث می شود اینطور به نظر برسد که زبان برای این مشکل ایجاد شده است*[25]*!* بنابراین این مسئولیت ماست که زبان را ساده جلوه دهیم! طرفداران زبانها در همه جا هستند، هشیار باشید! این زبان نیست که برنامه ها را ساده جلوه دهد. این برنامه نویس است که باعث می شود زبان ساده به نظر برسد!

مکتب فکری!

در مورد من(عمو Bob) چی؟ من در مورد کد تمیز چه فکر میکنم؟ این کتاب با جزییات زیاد آنچه من و همفکرانم درباره کد تمیز فکر می کنیم را به شما خواهد گفت. ما آنچه که فکر میکنیم یک نام متغیر تمیز ، یک تابع تمیز ، یک کلاس تمیز و غیره را ایجاد می کند به شما

age look like it was made for the problem!

خواهیم گفت. ما این عقاید را مطلق ارائه خواهیم کرد و از سختگیری خود عذرخواهی نمی کنیم. برای ما ، در این مرحله از حرفه مان ، آنها مطلق هستند. آنها مکتب فکری ما در مورد کد تمیز هستند.

هنرمندان رزمی همه با بهترین هنر رزمی یا بهترین تکنیک در یک هنر رزمی موافق نیستند. اغلب استادان هنرهای رزمی مکتب خود را تشکیل می دهند و دانش آموزان را برای یادگیری دور خود جمع می کنند. بنابراین ما Gracie Jiu Jistu را می بینیم ، که توسط خانواده Gracie در برزیل تأسیس و تدریس شده است. ما Hakkoryu Jiu Jistu را می بینیم که توسط Okuyama Ryuho در توکیو تاسیس و تدریس شده است. ما Jeet Kune Do را می بینیم ، که توسط بروس لی در ایالات متحده تاسیس و تدریس شده است.

هنرجویان این رویکردها، خود را در آموزه های بنیانگذار غرق می کنند. آنها خود را وقف این می کنند که آنچه آن استاد خاص تدریس میکند را فارغ از چیزی که استاد

دیگر تدریس میکند، بیاموزند. بعداً با رشد هنرجویان در هنر خود ، ممکن است دانش آموز استاد دیگری شوند تا بتوانند دانش و تمرین خود را گسترش دهند. عده ای سرانجام برای کشف مهارت های خود ، به کشف تکنیک های جدید و تأسیس مکتب خود می روند.

هیچ یک از این مکاتب مختلف کاملاً درست نیستند. با وجود این ، در درون یک مکتب خاص به نظر می رسد که آموزه ها و فنون صحیح هستند. از این گذشته ، یک روش درست برای تمرین Hakkoryu Jiu Jitsu یا Jeet Kune Do وجود دارد. اما این حق در یک مکتب ، آموزه های یک مکتب متفاوت را باطل نمی کند.

این کتاب را در مورد توصیفات مکتب اشیا آموزشی در کد تمیز[26] در نظر بگیرید. تکنیک ها و آموزه های موجود روشی است *که* هنر خود را تمرین می کنیم. ما مایل هستیم ادعا کنیم که اگر این آموزه ها را رعایت کنید ، از مزایایی که ما از آنها لذت بردیم لذت خواهید برد و یاد می گیرید که کدی بنویسید که تمیز و حرفه ای باشد. اما این اشتباه را نکنید که فکر کنید که "حق"به طور مطلق با ما است. مکاتب و اساتید دیگری نیز وجود دارند که به همان اندازه که ما ادعا داریم، حرفه ای هستند. شایسته است که شما از آنها نیز بیاموزید.

در واقع ، بسیاری از توصیه های این کتاب جنجال برانگیز است. احتمالاً با همه آنها موافق نخواهید بود. ممکن است با بعضی از آنها به شدت مخالف باشید. خوب است. ما نمی توانیم ادعای کمال اعتبار کنیم. از طرف دیگر ، توصیه های موجود در این کتاب مواردی است که ما طولانی و سخت در مورد آنها فکر کرده ایم. ما آنها را طی چندین دهه تجربه و آزمایش و خطای مکرر آموخته ایم. بنابراین چه موافق باشید یا مخالف باشید ، اگر به نقطه نظر ما احترام نگذارید و آن را نبینید شرم آور خواهد بود.

ما نویسنده ایم

فیلد @author در یک Javadoc به ما می گوید که ما کی هستیم. ما نویسنده هستیم و یک چیز در مورد نویسندگان وجود دارد و آن این است که آنها خواننده دارند. در واقع ، نویسندگان *مسئول* برقراری ارتباط خوب با خوانندگان خود هستند. دفعه بعد که شما یک خط از یک کد را نوشتید ، به یاد داشته باشید که شما نویسنده ای هستید که برای خوانندگانی که تلاش شما را قضاوت می کنند ، می نویسید.

ممکن است بپرسید : یک کد واقعا چه مقدار خوانده میشود؟ تمام تلاش ما صرف نوشتن آن نمیشود؟

[26] Object Mentor School of Clean Code

آیا تاکنون به یک جلسه ویرایش[27] دوباره باز گشته اید؟ در دهه 80 و 90 ویرایشگرانی مانندEmacs داشتیم که هرگونه فشار به صفحه کلید را ردیابی می کردند. می توانید یک ساعت کار کنید و بعد از آن کلیه ویرایش های خود را مانند یک فیلم پر سرعت پخش کنید. وقتی این کار را کردم ، نتایج جالب توجه بود.

اکثریت قریب به اتفاق تجدید نظرها در مورد بخش پیمایش و هدایت به سایر ماژول ها بود!

باب وارد ماژول می شود.

او به تابعی که نیاز به تغییر دارد می رود

او با توجه به گزینه های خود مکث می کند.

اوه ، او در حال برگشتن به بالای ماژول برای بررسی مقدار اولیه داده شده به یک متغیر است.

اکنون او دوباره به پایین برمیگردد و شروع به تایپ می کند.

اوه ، او دارد آنچه را که تایپ کرده است پاک میکند!

او دوباره آن را تایپ می کند.

[27] Edit session

او دوباره آن را پاک می کند!

او نیمی از چیز دیگری را تایپ می کند اما بعد آن را پاک می کند!

او به تابع دیگری می رود که تابعی را که دارد تغییر می دهد را صدا میزند تا ببیند چگونه آن تابع صدا زده شده است.

او دوباره به بالا برمیگردد و همان کدی را که تازه پاک کرده است تایپ می کند.

مکث می کند.

او دوباره آن کد را پاک می کند!

وی پنجره دیگری را باز می کند و به یک زیر کلاس نگاه می کند. آیا این تابع دوبار نوشته شده است؟

. . .

شما بی اراده کار می کنید. در واقع ، نسبت زمانی که صرف خواندن میکنید در مقابل زمانی که صرف نوشتن میکنید بیش از 10: 1 است. *همیشه* بخشی از تلاشمان برای نوشتن کد جدید ، برای خواندن کد قدیمی صرف میشود.

از آنجا که این نسبت بسیار زیاد است ، می خواهیم خواندن کد آسان باشد ، حتی اگر این کار نوشتن را سخت تر کند. البته هیچ راهی برای نوشتن کد بدون خواندن آن وجود ندارد ، بنابراین *آسان تر کردن خواندن در واقع نوشتن آن را آسان تر می کند.*

از این منطق گریزی وجود ندارد. اگر نمی توانید کدهای اطراف را بخوانید ، نمی توانید کد بنویسید. نوشتن کدی که می خواهید امروز بنویسید بسته به اینکه چقدر خواندن کد اطراف آن سخت یا آسان باشد، دشوار یا آسان خواهد بود. بنابراین اگر می خواهید سریع کار کنید ، اگر می خواهید به سرعت کار خود را به اتمام برسانید ، اگر می خواهید کد شما به راحتی نوشته شود ، خواندن آن را آسان کنید

قانون پیشاهنگان پسر[28]

اینکه کد به خوبی نوشته شود کافی نیست. کد باید در طول زمان تمیز نگه داشته شود. با گذشت زمان ، همه ما شاهد پوسیدن و کاهش درجه ارزش کد هستیم. بنابراین ما باید نقش فعالی در جلوگیری از این تخریب داشته باشیم.

پیشاهنگان پسر امریکا یک قانون ساده دارند که ما می توانیم از آن در حرفه خود استفاده کنیم.

[28] The boy scout rule

محل اردوگاه را تمیزتر از آنچه که به آن وارد شدید، ترک کنید[29]

اگر همه ما هنگام ورود به کد، کد خود را کمي تميزتر از زماني که آن را رها کرده بوديم کنيم ، کد به سادگي نمي تواند پوسيده شود. پاکسازی لازم نیست چیز بزرگی باشد. بهتر کردن نام یک متغیر، شکستن یک تابع نسبتا بزرگ به توابع کوچکتر، از بین بردن یک تکثیر ، پاک کردن یک عبارت شرطی ترکیبی باعث تمیزتر شدن کد میشود.

آیا می توانید کار کردن روی پروژه ای که کدش با گذشت زمان *به آسانی بهتر شده است* را تصور کنید؟آیا معتقدید که گزینه ای غیر از این حرفه ای است؟ در واقع ، آیا پیشرفت مداوم جز ذاتی حرفه ای بودن نیست؟

مقدمه و اصول

از بسیاری جهات ، این کتاب "مقدمه" کتابی است که من در سال 2002 با عنوان توسعه نرم افزار چابک: اصول ، الگوهای و عملکردها[30] نوشتم. کتابPPP خود با اصول طراحی شی گرا و بسیاری از شیوه هایی که توسط توسعه دهندگان حرفه ای استفاده می شود درگیر استاگر PPP را نخوانده اید ، ممکن است در آینده متوجه شوید که آن کتاب، داستانی که توسط این کتاب گفته شده را ادامه

[29] این از پیام وداع Robert Stephenson Smyth Baden-Powell با پیشاهنگان اقتباس شده است: "سعی کنید این دنیا را کمی بهتر از آنچه به آن وارد شدید ترک کنید. . "

[30] **Agile Software Development: Principles, Patterns, and Practices** (PPP)

می دهد. اگر قبلاً آن را خوانده اید ، می توانید تکرار بسیاری از تفکرات آن کتاب در سطح کد را در این کتاب ببینید.

در این کتاب اشاراتی پراکنده به اصول مختلف طراحی خواهید یافت. در میان این اصول اصل تک مسئولیت[31] ، اصل بسته باز [32] و اصل وارونگی وابستگی[33]وجود دارد. این اصول به تفصیل در PPP شرح داده شده است

نتیجه گیری

کتابهای مربوط به هنر قول نمی دهند شما را به یک هنرمند تبدیل کنند. تمام کاری که آنها می توانند انجام دهند این است که برخی از ابزارها ، تکنیک ها و فرآیندهای فکری که سایر هنرمندان استفاده کرده اند را به شما ارائه می دهند. بنابراین این کتاب نیز نمی تواند قول دهد شما را به یک برنامه نویس خوب تبدیل کند. نمی تواند قول بدهد که به شما "درک کد" را بدهد. تمام کاری که می تواند انجام دهد این است که فرآیندهای فکری برنامه نویسان خوب و ترفندها ، تکنیکها و ابزارهایی را که از آنها استفاده می کنند به شما نشان دهد.

درست مانند یک کتاب در زمینه هنر ، این کتاب پر از جزئیات خواهد بود. تعداد زیادی کد وجود دارد. هم کد خوب خواهید دید و هم کد بد. میبینید که کد بد را به کد خوب تبدیل می کنید. لیست های از اکتشاف[34] ، قوانین و تکنیک ها را مشاهده خواهید کرد. مثال پشت مثال خواهید دید. پس از آن ، به شما بستگی دارد.

شوخی قدیمی درباره ویولنیست کنسرت را که در راه رسیدن به یک اجرا گم شده بود را به یاد می آورید؟ او پیرمردی را در گوشه ای متوقف کرد و از او پرسید که چگونه به Carnegie Hall برسد. پیرمرد به ویولنیست و ویولون زیر بازویش نگاه کرد و گفت: "تمرین کن پسرم. تمرین کن!"

---

[31] **Single Responsibility Principle (SRP)**
[32] **Open Closed Principle (OCP)**
[33] **Dependency Inversion Principle (DIP)**
[34] heuristic