درس‌هایی از کارِ برنامه‌نویسان

علیرضا خلیلیان، دانشجوی دکتری نرم‌افزار

گروه مهندسی نرم‌افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان

khalilian@eng.ui.ac.ir

چکیده: اگر تاکنون فرصت نکرده‌اید کتاب «کارِ برنامه‌نویسان» را بخوانید، اشکالی ندارد؛ به شما توصیه می‌کنم چند دقیقه‌ای را با ما در این مقاله بگذرانید تا جان کلامِ این کتاب ۵۳۶ صفحه‌ای را تقدیم شما کنیم. کسی چه می‌داند؛ شاید شما هم راغب شدید کل کتاب را بخوانید. دونالد کنوث در شاهکار چند جلدی‌اش، مقاله‌های قوی را خوانده‌است، آن‌ها را در ذهن خودش هضم کرده است و حاصل کار را در چندین خط یا پاراگراف خلاصه بیان نموده است. ما هم همین‌کار را با کتاب «کار برنامه‌نویسان» کردیم! چه مقایسه‌ای! با استاد ممتاز دانشگاه استنفورد و نابغه الگوریتم. بگذریم. با ما در این مقاله همراه باشید تا نکات برجسته را برای شما گزارش کنیم.

۱-۱- مقدمه

کتاب «کار برنامه‌نویسان» [1] در سال ۲۰۰۹ توسط پیتر سیبِل نوشته شده است. این کتاب توسط آقای مهندس ابراهیم نقیب‌زاده مشایخ ترجمه شده و متن آن به‌تدریج در شماره‌هایی از مجله «گزارش کامپیوتر» به‌چاپ رسیده است. همچنین در سال ۱۳۹۱ به سفارش انجمن انفورماتیک ایران[1] چاپ شده است. کتاب ضمن اینکه ترجمه بسیار روانی دارد، **خود شمول** است؛ به این معنی که مترجم هر جا اصطلاح خاصی ذکر شده توضیح اضافه‌ای برای درک آن درون متن درج کرده است که خواننده را از جست‌وجو به‌دنبال معنی و مفهوم آن بی‌نیاز سازد.

این کتاب گزارش مصاحبه‌هایی است که پیتر سیبل با ۱۵ نفر از برنامه‌نویسان مشهور انجام داده است. نویسنده کتاب سعی کرده مصاحبه‌هایش را به‌طور **نظام‌یافته** ترتیب دهد و تقریباً سؤال‌های مشابهی از همه پرسیده است. این اقدام خوب، زمینه‌ای را فراهم کرده که بتوانیم با خواندن کتاب و برجسته کردن نکات مهم، درس‌هایی از سال‌ها فعالیت و تجربه برنامه‌نویسان بیاموزیم. بسیاری از درس‌ها و نکات ممکن است در کتاب‌های آکادمیک یافت شوند ولی وقتی در خلال شرح حالِ کار و زندگی فردی مشهور بیان می‌شوند، بهتر به دل می‌نشینند و درک می‌شوند. این نکات به‌خصوص مهم هستند چون افراد مصاحبه شونده اغلب از زمانی که فقط زبان‌های اولیه‌ای همچون اسمبلی[2] و فرترن و رایانه‌های اولیه‌ای که تنها چند صد بایت حافظه داشته‌اند شروع به‌کار کرده‌اند و امروز

---

[1] http://isi.org.ir/index.asp

[2] زبان اسمبلی را در فارسی زبان «همگذاری» ترجمه کرده‌اند اما چون اسمبلی اسم است، به‌نظر نباید ترجمه شود و بنابراین از همان نام رایج استفاده شده است.

که رایانه‌های چند هسته‌ای با ده‌ها ترابایت حافظه و صدها زبان[1] برنامه‌سازی داریم برای ما حرف می‌زنند.

به‌همین دلیل نویسنده مقاله حاضر موقع خواندن هر یک از ۱۵ فصل، نکات مهم را یادداشت‌برداری کرد تا بعد از خلاصه‌سازی و پالایش نکات، آن‌ها را در قالب مقاله حاضر ارائه نماید. هنگام نکته‌برداری سعی شده نکاتی برجسته شوند که حتی‌الامکان همه مصاحبه‌شوندگان به‌نوعی به آن پاسخ داده‌اند که امکان مقایسه فراهم گردد. همچنین نکاتی برداشت شده‌اند که در حال حاضر معتبر و قابل استفاده باشند یا درس مهمی برای فعالیت‌های امروز و آینده برنامه‌نویسان داشته باشند. علاوه بر نقل نکته‌ها، هر جا که لازم بوده، مطالب بیشتری هم برگرفته از مقاله‌ها یا کتاب‌های معتبر امروزی اضافه شده‌اند که سودمندی مطالب نقل شده را بیشتر نمایند.

۱-۲- مصاحبه‌شوندگان

اغلب مصاحبه‌شوندگان در دانشگاه‌های معتبری چون هاروارد، استنفورد و ام‌آی‌تی در رشته‌هایی مثل ریاضی، فیزیک و مهندسی برق در سطوح مختلف از کارشناسی تا دکتری تحصیل کرده‌اند. این افراد عموماً در شرکت‌های معتبری مثل گوگل، مایکروسافت و یاهو کار کرده‌اند یا مشغول به‌کارند. همچنین اغلب آن‌ها از اعضای دائمی یا برجسته IEEE، ACM یا فرهنگستان ملی مهندسی هستند. افرادی که پیتر سیبل برای مصاحبه انتخاب کرده است عبارتند از:

أ. **جیمی زاوینسکی**[2]: نویسنده نِت‌اِسکیپ

ب. **براد فیتز پاتریک**[3]: پدید آورنده لایو ژورنال

ت. **داگلاس کراکفورد**[4]: معمار ارشد جاوا اِسکریپت

ث. **برندن اِیک**[5]: خالق جاوا اِسکریپت

ج. **جاشوا بلاک**[6]: معمار ارشد جاوا

ح. **جو آرمسترانگ**[7]: خالق زبان اِرلانگ

---

[1] به وب‌گاه **http://www.99-bottles-of-beer.net/** مراجعه کنید تا کدی با عملکرد یکسان را به ۱۵۰۰ زبان برنامه‌سازی مختلف ببینید.

[2] Jamie Zawinski

[3] Brad Fitzpatrick

[4] Douglas Crockford

[5] Brendan Eich

[6] Joshua Bloch

[7] Joe Armstrong

خ. **سایمون پیتون جونز**[1]: معمار زبان هاسکِل

د. **پیتر نورویگ**[2]: مدیر بخش پژوهش شرکت گوگل

ذ. **گای استیل**[3]: طراح زبان اِسکیم

ر. **دن اینگالس**[4]: پیاده‌ساز اِسمال‌تاک

ز. **پیتر دویچ**[5]: نویسنده گوست اِسکریپت

س. **کن تامپسون**[6]: خالق سیستم عامل یونیکس

ش. **فران آلن**[7]: پیشگام مترجم‌های بهینه‌ساز

ص. **برنی کاسل**[8]: استاد اشکال‌زدایی

ض. **دونالد کنوث**[9]: نویسنده کتاب هنر برنامه‌نویسی رایانه

از بین این ۱۵ نفر تنها یک‌نفر یعنی فران آلن خانم است که نخستین خانمی بود که عنوان «همکار IBM[10]» که بالاترین عنوان فنی افتخاری در IBM است را به‌دست آورد. اغلب مصاحبه-شوندگان یک یا چند جایزه مورد از جوایز معتبر زیر را به پاس نوآوری‌ها یا سال‌ها فعالیت و تجربه دریافت کرده‌اند:

جایزه تورینگ[11] **ACM**: سالانه به اشخاصی که سهم به‌سزایی در زمینه رایانه دارند، اعطا می‌شود. شرکت‌های گوگل و اینتل حامیان مالی این جایزه هستند و معادل نوبل در نظر گرفته می‌شود.

جایزه جولت[12]: جایزه‌ایست که هر سال به محصولاتی داده می‌شود که اهمیت فوق‌العاده آن‌ها باعث تکانی در صنعت و منجر به تولید سریعتر، ساده‌تر و کارامدتر نرم‌افزار شوند.

---

[1] Simon Peyton Jones
[2] Peter Norvig
[3] Guy Steele
[4] Dan Ingalls
[5] Peter Deutsch
[6] Ken Thompson
[7] Fran Allen
[8] Bernie Cosell
[9] Donald Knuth
[10] IBM Fellow
[11] Turing
[12] Jolt

جایزه دستاوردهای استثنائی ناسا[1]: جایزه‌ایست که به افراد غیرنظامی ناسا و فضانوردان اعطا می‌شود. فرد دریافت کننده جایزه، کارمندی از ناساست که به نوآوری مهمی دست‌یافته است که باعث بهبود چشمگیر عملیات، کارایی، خدمات، صرفه‌جویی اقتصادی، علم یا فناوری شده است که مستقیماً سهمی در مأموریت ناسا داشته باشد.

جایزه گریس هاپر ACM[2]: جایزه‌ایست که به متخصص رایانه‌ای داده می‌شود که نوآوری مهم فنی یا خدماتی تا ۳۵ سالگی داشته باشد.

جایزه تعالی در برنامه‌نویسی دکتر داب[3]: جایزه‌ایست که هر ساله به افرادی که از نظر ویراستاران ژورنال دکتر داب، سهم به‌سزایی در پیشرفت توسعه نرم‌افزار داشته باشد اعطا می‌گردد.

جایزه سیستم نرم‌افزاری ACM[4]: این جایزه سالانه به افراد یا سازمانی اعطا می‌شود که سیستم نرم‌افزاری توسعه داده‌اند که تأثیر طولانی مدتی داشته است که این تأثیر در پذیرش تجاری یا کمک به مفاهیم یا هر دو منعکس شده باشد.

جایزه تیسوتومو کانائی IEEE[5]: این جایزه به فردی که نوآوری‌های اساسی در سیستم‌های رایانش توزیعی و کاربردهای آن داشته باشد اعطا می‌گردد.

هوش بالا، تجربه طولانی، شهرت و دریافت جوایز معتبر بین‌المللی باعث می‌شوند بتوانیم حرف‌های مصاحبه‌شوندگان را تعمیم دهیم و آن‌ها را برای زمان‌ها، مکان‌ها و موقعیت‌های گوناگون معتبر بدانیم. اغلب مصاحبه‌شوندگان با کامپیوتر در مدرسه و دبیرستان در دهه‌های ۵۰ تا ۷۰ میلادی آشنا شده‌اند و اکثر آن‌ها به‌خصوص قدیمی‌ترها با انواعی از کامپیوترهای IBM و زبان فرترن سال-ها کار کرده‌اند. یکی از عوامل موفقیت زبان‌ها در کتاب‌های طراحی و پیاده‌سازی زبان‌ها [2, 3] حمایت مالی و نفوذ شرکت سازنده[6] ذکر شده است. اکنون بهتر روشن می‌شود که چرا زبانی مثل فرترن آن‌قدر مورد استفاده همگان بوده در حالی‌که زبانی مثل الگول که طراحی بسیار زیباتری داشته است دوام چندانی نداشته است. فرترن از دل IBM غول محاسبات بیرون آمد و همه سازمان‌ها و دانشگاه‌ها کم و بیش رایانه‌های IBM را خریداری می‌کردند که روی آن‌ها فرترن نصب

---

[1] NASA Exceptional Achievement Medal

[2] ACM's Grace Murry Hopper

[3] Dr, Dobb's Excellence in Programming

[4] ACM Software System

[5] IEEE's Tsutomu Kanai

[6] Patronage, Inertia, and Sponsorship

بوده است. درحالیکه الگول زبانی است دانشگاهی و موقعیتی مشابه فرترن نداشته است. جو آرمسترانگ هم در مورد زبان ارلانگ بیان کرده که شاید بهتر باشد مایکروسافت برخی از ایده-هایش را بگیرد و در قالب محصولی عرضه کند که ارلانگ هم بازار وسیعی پیدا کند.

۱-۳- خوانایی کد

تقریباً همه افراد بر اهمیت خوانایی کد تأکید دارند. خوانایی کد تأثیر اساسی بر سهولت اشکال-زدایی، کم کردن هزینه‌ها و فعالیت‌های مرحله نگهداری کد و افزایش قابلیت اطمینان نرم‌افزار دارد. استفاده از شناسه‌های طولانی برای خوانایی توسط اکثر افراد همچون زاوینسکی و بلاک توصیه شده است. آرمسترانگ سعی کرده از اسامی بسیار نامتشابه استفاده کند تا ابهام از بین رود. گذاشتن توضیحات در متن برنامه توسط زاوینسکی لازم دانسته شده ولی بعضی توصیه نمی‌کنند. آرمسترانگ مستندسازی کد و نوشتن مشخصات کد را در حد معقول لازم می‌داند. بِلاک خوانا بودن برنامه و هنر نویسندگی و کار ادبیاتی در کد نویسی را خیلی مهم می‌داند. اکثریت بر این باورند که کد خوانا مربوط به برنامه‌نویسی است که نویسنده خوبی هم باشد.

۱-۴- زبان‌های برنامه‌سازی

اکثر مصاحبه‌شوندگان با زبان‌های معدودی کار کرده‌اند ولی مواردی هم هست که زبان‌های بسیاری مورد استفاده قرار گرفته‌اند. **اسمبلی، فرترن** و **لیسپ** در زمره زبان‌هایی هستند که مورد استفاده اکثریت بوده‌اند. **جاوا، پایتون، پرل** و **جاوا اسکریپت** هم سایر زبان‌های مورد استفاده بوده‌اند. تعدادی هم از **C** و از **C++** استفاده کرده‌اند با علم به اینکه این دو زبان مشکلات امنیتی بسیاری دارند. کاسل اظهار داشته که علی‌رغم مشکلات امنیتی بسیار، C نعمت بزرگی برای برنامه‌نویسی سیستم بوده است. اغلب افراد مثل آرمسترانگ رابطه خوبی با C++ ندارند زیرا آن‌را زبانی فوق‌العاده بزرگ و پیچیده می‌دانند. باید توجه کرد که C و C++ زبان‌هایی هستند که اساساً برای کارایی و برنامه‌سازی سیستم طراحی شده‌اند. دو ویژگی مذکور طراح را مجبور می‌کند که هر نوع محدودیتی را بردارد و قابلیت‌های متعدد را در زبان قرار دهد. در این‌صورت همه چیز به‌عهده برنامه‌ساز است و زبان به هیچ وجه مناسب افراد غیر حرفه‌ای نیست. به گفته بلاک، برنامه‌سازانی که از C++ استفاده می‌کنند، معمولاً زیرمجموعه‌ای از آن‌را انتخاب می‌کنند. این موضوع و پیچیدگی زیاد، باعث می‌شود که تامپسون اظهار دارد از C++ خوشش نمی‌آید و علی‌رغم اینکه C++ یکی از چهار زبان گوگل است ولی اصلاً برای انتقال الگوریتم مناسب نیست.

افرادی همچون کراکفورد و اِیک بر مشکلات امنیتی جاوا اسکریپت اذعان دارند و در عین حال تأکید دارند که این زبان نوآوری‌های قابل توجهی هم دارد از جمله سازگاری در مرورگرهای مختلف و پویایی. پیتون جونز معتقد است پژوهش درباره زبان‌های برنامه‌سازی ضعیف است چون سؤال‌های مشکلی برای پاسخ دادن وجود دارد. گای استیل اعتقاد دارد که در گذشته زبان‌های برنامه‌سازی همچون پاسکال به‌طور کامل طراحی می‌شده ولی امروزه به‌دلیل بزرگی، زبان‌ها را نمی‌شود یک‌باره طراحی کرد و زبان دوره تکامل را می‌گذراند. طراحی زبان برنامه‌سازی مجموعه-ایست از موازنه‌ها که طراح باید با توجه به نیاز برنامه‌نویسان و کاربرد زبان انتخاب در مورد ویژگی-های زبان تصمیم‌گیری نماید.

۱-۵- استانداردسازی زبان

برخی همچون کراکفورد معتقدند که زبان نباید خیلی زود استاندارد شود و باید کمی صبر کرد تا تجربه‌هایی از کاربردهای واقعی زبان به‌دست آید. زمان‌شناسی، پیروی از استاندارد و کهنگی سه مسئله مهم در استانداردسازی زبان هستند.

۱-۶- دانش ریاضی

تقریباً اکثریت اذعان دارند که داشتن دانش عمیق ریاضی برای برنامه‌نویسان ضروری نیست؛ ولی آگاهی نسبی از ریاضی گسسته، آمار و احتمال و منطق ضروری و مفید است. نیاز به بنیاد قوی ریاضی مانع می‌شود که اکثر برنامه‌سازان بتوانند کتاب هنر برنامه‌نویسی رایانه کنوث را کامل بخوانند و درک نمایند.

۱-۷- نظام مهندسی، الگوها و چارچوب‌ها

بعضی همچون زاوینسکی به‌خاطر مسایل رقابت در بازار، اعتقادی به استفاده از الگوها و چارچوب-های طراحی ندارند. اِیک الگوهای طراحی را چندان سودمند نمی‌داند. ولی بلاک آگاهی از الگوهای طراحی را بسیار مفید می‌داند؛ شاید چون مدرک دکتری او در مهندسی کامپیوتر، نگاه آکادمیک بیشتری به او بخشیده است. امروزه در تولید بسیاری از کتاب‌خانه‌ها و چارچوب‌های نرم‌افزاری از الگوهای طراحی استفاده شده است که آگاهی از آن‌ها به درک سریعتر کمک می‌کند. شاید یکی از هدف‌های الگوهای طراحی، ساختاردهی به کد برنامه‌سازان (مبتدی) و افزایش خوانایی و قابلیت بازبه‌کارگیری است.

۱-۸- آموزش و یادگیری زبان و برنامه‌نویسی

برخی همچون زاوینسکی اعتقاد دارند به‌جای قوانین گرامری زبان، باید طرز عمومی برنامه‌نویسی را آموزش داد. تقریباً همه مصاحبه‌شوندگان تأکید دارند که یکی از بهترین روش‌های یادگیری و مهارت یافتن در برنامه‌نویسی، خواندن کد دیگران است، البته کد خوب از برنامه‌نویسان خوب. بلاک معتقد است برای یادگیری برنامه‌نویسی نباید از زبانی مثل C شروع کرد. آرمسترانگ معتقد است که برای درک کامل یک زبان بهتر است مترجمی برای آن زبان بنویسید. گای استیل خواندن کد TeX نوشته کنوث را بسیار مفید می‌داند چون خوش نوشتار است و خوب اشکال‌زدایی شده است. آلن هم خواندن برنامه‌های خوب زبان جدید را برای یادگیری آن مؤثر می‌داند. یکی از دلایل طراحی زبان‌های جدید بر مبنای زبان‌های گذشته، انتقال تجربه‌ها و سهولت آموزش زبان است.

۱-۹- یادگیری جزئیات سیستم‌ها

از صحبت‌های افراد می‌توان نتیجه گرفت که حتی امروزه آگاهی از جزئیات سطح پایین سخت-افزاری و نرم‌افزاری برای برنامه‌نویسان مفید است؛ هر چند که دیگر ۹۹ درصد افراد درگیر چنین جزئیاتی نمی‌شوند. امروزه برنامه‌نویسی روی لایه‌های بالایی از انتزاع انجام می‌شود و کسی مجبور نیست سیستم عامل، ویراستار یا مترجم بنویسد.

۱-۱۰- تضمین کیفیت، اشکال‌زدایی و وارسی صحت برنامه

تقریباً همه مصاحبه‌شوندگان در عمده فعالیت‌های اشکال‌زدایی خود از **دستور چاپ** استفاده کرده‌اند. برخی هم از **دستور اظهار**[1] برای بررسی مقادیر **نامتغیر**[2] در متن برنامه بهره برده‌اند. عده‌ای هم از امکانات محیط برنامه‌نویسی برای **ردیابی کد**[3] و **گام‌زنی**[4] استفاده کرده‌اند. برخی هم ابزارهایی همچون Valgrind و Strace و FindBugs را برای یافتن نوع خاصی از خطاها سودمند می‌دانند. برنامه‌نویسان جاوا اسکریپت مثل کراکفورد هم از ابزارهای اشکال‌زدایی مرورگرها مثل **فایرباگ** استفاده کرده‌اند.

تقریباً همگی معتقدند که وارسی صحت برنامه از طریق اثبات صوری عملی نیست و اگر هم بخواهیم استفاده کنیم، صرفاً باید در کدهای کوچک و نرم‌افزارهایی که در کاربردهای حساس به-

---

[1] Assertion

[2] Invariant

[3] Trace

[4] Step through code

کار گرفته می‌شوند از اثبات صوری استفاده شود. از نظر گای استیل و کنوث، اثبات صحت هم ممکن است خطا داشته باشد و با اثبات صحت نمی‌توان ثابت کرد که برنامه بی‌خطاست. اما چون اثبات صحت با روش دیگر و ابزار دیگری صورت می‌گیرد، احتمال وجود خطای آن کمتر از خطای کد است. دویچ نیز اظهار می‌دارد که صوری کردن بسیاری از خصوصیت‌های نرم‌افزار فوق‌العاده مشکل است.

افرادی چون اِیک بیان می‌کنند که برای اشکال‌زدایی پژوهش‌های کافی صورت نگرفته است. این حرف او را می‌توان کاملاً قابل قبول دانست وقتی که نتایج مقاله اُورسو [4] را مطالعه کنیم. این مقاله پس از مطالعه‌های موردی به این نتیجه رسیده است که علی‌رغم مطالعه‌های بسیار، فنون موجود هنوز در دست برنامه‌نویسان در موقعیت‌های عملی سودمند نیستند.

تقریباً همه مصاحبه‌شوندگان از **مرور کد**[1] به‌عنوان روشی برای تضمین کیفیت و صحت کد استفاده کرده‌اند. برخی چون بلاک عاشق **تحلیل‌های ایستا** هستند چون دسته‌ای خطاها به‌طور خودکار حذف می‌شوند. به‌نظر پیتون جونز «هر کجا نوع ایستا برازنده بود حتماً از آن استفاده کنید زیرا مزایای فوق‌العاده‌ای برای نگهداری نرم‌افزار دارد.» نورویگ و تامپسون استفاده از **آزمون واحد**[2] و **آزمون پس‌نمایی**[3] را برای تضمین کیفیت مهم می‌دانند. بنابر اظهارات نورویگ، در گوگل یکی از گام‌های مهم در تضمین کیفیت کد، مرور کد توسط فردی دیگر است.

۱-۱۱- کتاب‌های مورد مطالعه و توصیه شده

به‌جز معدودی همچون زاوینسکی اکثراً کتاب «**هنر برنامه‌سازی رایانه**[4]» اثر دونالد کنوث را به‌طور کلی یا جزئی خوانده‌اند و معتقدند کتابی است فاخر که حالت مرجع دارد و در موقعیت‌های لازم باید به آن رجوع کرد. کتاب «**الگوهای طراحی**[5]» را برخی توصیه می‌کنند و برخی مضر می‌دانند. عده‌ای هم همچون بلاک عقیده دارند کتاب‌های «**نفر ماه افسانه‌ای**[6]» و «**عناصر سبک (نویسندگی)**[7]» را باید خواند به‌خصوص که دومی کتاب برنامه‌نویسی نیست ولی بخشی از

---

[1] Code Review
[2] Unit Testing
[3] Regression Testing
[4] The Art of Computer Programming
[5] Design Patterns
[6] Mythical Man-Month
[7] Elements of Style

مهندسی نرم‌افزار نویسندگی است. اغلب توصیه می‌کنند باید کتابی در الگوریتم و ساختمان داده‌ها خوانده شود. نورویگ خواندن یک کتاب اشکال‌زدایی و یک کتاب در نکات و مهارت‌های کدنویسی همچون «**کد کامل**[1]» را ضروری می‌داند. پیتون جونز هم کتاب‌هایی همچون «**مرواریدهای برنامه‌نویسی**[2]» جان بنتلی که درباره زیبایی‌های درونی کد است، «**ساختمان داده‌های تابعی محض**[3]» از کریس اوکازاکی، «**ساختار و تفسیر برنامه‌های کامپیوتر**[4]» از ابلسون و سوزمان و «**ترجمه از طریق ادامه‌دهی**[5]» از اندرو اپل را توصیه می‌کند.

۱-۱۲- بهینه‌سازی کد

اغلب مصاحبه‌شوندگان اعتقاد دارند که بهینه‌سازی کد توسط برنامه‌نویس به‌خصوص امروزه اهمیت کمی دارد و خوانایی بسیار مهم‌تر است. اگر بخشی از کد قرار است میلیون‌ها بار اجرا شود و نرم-افزار در کاربردهای بلادرنگ استفاده می‌شود، آن‌گاه کارایی و بهینه‌سازی اهمیت پیدا می‌کند. تامپسون معتقد است بهینه‌سازی وقت‌گیر است و کد را پیچیده می‌سازد.

۱-۱۳- پاک‌سازی و بازسازی کد

افرادی همچون کراکفورد توصیه می‌کنند بعد از هر شش چرخه نرم‌افزار باید آن‌را مورد پاکسازی اساسی قرار داد. سال‌هاست که فنونی همچون **بازسازی کد**[6] در محیط‌های توسعه یکپارچه[7] ادغام شده‌اند تا به خوانایی کد و ساختاردهی بهتر آن کمک کنند.

۱-۱۴- استفاده از برنامه‌نویسی ادیبانه کنوث

اغلب مصاحبه‌شوندگان از برنامه‌نویسی ادیبانه[8] کنوث اطلاع دارند و عده‌ای ضمن تمجید آن، در کدنویسی هم استفاده کرده‌اند. معدودی همچون تامپسون هم ضمن تحسین آن معتقدند در عمل قابل استفاده نیست. گای استیل برنامه‌نویسی ادیبانه را در کار خودش مؤثر می‌داند.

---

[1] Code Complete
[2] Programming Pearls
[3] Purely Functional Data Structures
[4] Structure and Interpretation of Computer Programs
[5] Compilation with Continuation
[6] Refactoring
[7] Integrated Development Environment (IDE)
[8] Literate Programming

۱-۱۵- شناخت برنامه‌نویس خوب برای استخدام

تقریباً همه بیان کرده‌اند که موقع مصاحبه برای استخدام برنامه‌نویس طرح سوال‌ها و مسئله‌های معما گونه که در شرکت‌هایی مثل گوگل و مایکروسافت رایج است، سودمند نیست. آرمسترانگ اعتقاد دارد برخی از برنامه‌نویسان خوب در حل معماها کند هستند. بیشتر افراد علاقه‌مندند میزان تسلط فرد بر ساختمان داده‌ها و الگوریتم‌ها را بدانند و از او بخواهند از چگونگی حل مسئله، جزئیات کارهای عملی و بهترین پروژه‌ای که انجام داده صحبت کند. برنامه‌نویس خوب باید در موقعیت‌های برنامه‌نویسی بتواند تصمیم‌های منطقی بگیرد و بلد باشد خودش را با محیط و ابزار جدید وفق دهد و کار با آن‌را بیاموزد. تسلط بر یک زبان خاص دلالت بر خوبی برنامه‌نویس نیست. نورویگ معتقد است فردی که می‌خواهد جذب صنعت شود باید شیوه کارِ گروهی را بلد باشد. این نیازمندی و نیز اشکال‌زدایی را در زمره مهارت‌هایی می‌داند که در دانشگاه آموزش داده نمی‌شوند. کاسل برای استخدام به‌نبال فردی بوده است که جست‌وجوگر، دقیق و کنجکاو باشد و رویکرد منطقی به حل مسایل داشته باشد.

۱-۱۶- ضرورت تحصیلات آکادمیک برای برنامه‌نویسی

نظر اِیک اینست که همه نیاز به دکتری رایانه ندارند و او از تجربه حضور در دره سیلیکان فکر می‌کند که معامله اقتصادی سودآوری نیست. به‌نظر می‌رسد امروزه که برنامه‌نویسی در سطوح انتزاعی گوناگونی صورت می‌گیرد، داشتن تحصیلات کارشناسی برای برنامه‌نویسی در سطح رابط گرافیکی کاربر و طراحی وب کافی باشد؛ اما برای برنامه‌نویسی در سیستم‌های مقیاس‌پذیر که محاسبات پیچیده‌ای دارند و ملاحظات فنی بسیاری لازم است، تحصیلات آکادمیک در مقاطع بالاتر می‌تواند مفید باشد.

۱-۱۷- خطاهای داخل کد

اکثر برنامه‌نویسان بیان کرده‌اند که خطاهای موجود در برنامه‌های هم‌روند، چندریسمانی و موازی در زمره دشوارترین خطاها برای اشکال‌زدایی و نوشتن آزمایه هستند. همچنین اِیک از خطاهای محل مشتری بیان می‌کند که مدت زمان زیادی باقی نمی‌مانند ولی یافتن آن‌ها دشوار است. برخی از خطاها هم در مواقع نادر رخ می‌دهند که کشف آن‌ها معمولاً ساده نیست. به‌نظر گای استیل، زبان از نوع خاصی از خطاها جلوگیری می‌کند.

۱-۱۸- طرز طراحی و کدنویسی

بلاک توصیه می‌کند کدی که از **واسط برنامه کاربردی**[1] استفاده می‌کند را پیش از خود واسط بنویسید. افرادی همچون بلاک با روش **برنامه‌نویسی دونفره**[2] بسیار کار کرده‌اند و معتقدند ضمن حفظ استقلال، خوب است برنامه‌نویسان یک تیم از کد همدیگر اطلاع داشته باشند. در مقابل اینگالس برنامه‌نویسی دونفره را نمی‌پسندد و تجربه هم نکرده است. برخی همچون نورویگ ابزارهای مدل‌سازی همچون UML را چندان دوست ندارند. تامپسون و دویچ بر طراحی نرم‌افزار و طراحی ساختمان داده‌های مناسب پیش از شروع کدنویسی تأکید دارند و دویچ اندازه‌گیری و مقیاس‌بندی را به‌خصوص در مواجهه با نرم‌افزارهای بزرگ مهم می‌داند. کاسل بر طراحی برنامه به‌طور آزمایش‌پذیر تأکید دارد.

۱-۱۹- کلام آخر با چند جمله کوتاه و خواندنی

پیتر دویچ: «سریع، ارزان، خوب: دوتایش را انتخاب کن.»

دونالد کنوث: «برنامه‌نویسی شبیه اعتقادات مذهبی است.»

پیتون جونز: «ویژگی هور: به‌جای نداشتن خطاهای واضح، واضح است که خطایی ندارد.»

جاشوا بلاک: «بهینه‌سازی کد صحیح آسان‌تر از تصحیح کد بهینه‌سازی شده است.»

داگلاس کراکفورد: «هر چه زبان موفق‌تر باشد، هزینه تغییر در آن بیشتر است.»

ادگار دایکسترا: «اگر نمی‌توانید به زبان مادریتان بنویسید، برنامه‌نویسی را رها کنید.»

مراجع

[1] Seibel, Peter. *Coders at work: Reflections on the craft of programming*. Apress, 2009.

[2] Sebesta, Robert W. *Concepts of programming languages*, 11th Ed., Pearson , 2016.

[3] Scott, Michael L. *Programming language pragmatics*, Morgan Kaufmann, 2009.

[4] Parnin, Chris, and Alessandro Orso. "Are automated debugging techniques actually helping programmers?." In *Proceedings of the 2011 international symposium on software testing and analysis*, pp. 199-209. ACM, 2011.

[1] Application Program Interface (API)

[2] Pair programming