شماره ۵/ اردیبهشت ماه ۱۳۹۶ – ۳۰ صفحه(رایگان)

تیستن

دوماهنامه الکترونیکی تخصصی تست نرم‌افزار

Containerها و محیط‌های شبیه‌سازی تست

TISTeN

The Iranian Software Test eNgineers

به نام یگانه عالِمِ عالَم‌آفرین

کیفیت،
ثروت و اعتبار است

محتوای تخصصی و کاربردی سایت تیستن در زمینه‌های:

- آخرین اخبار ایران و جهان در زمینه تست نرم‌افزار
- مقالات کاربردی تست نرم‌افزار
- آموزش در حوزه تست نرم‌افزار

مشاوره مهندسی تست در زمینه‌های:

- فرآیند توسعه نرم‌افزار و بسترسازی تست نرم‌افزار
- طراحی و آماده‌سازی فرآیند تست و جزییات آن
- آماده‌سازی و اجرای Test Plan
- استفاده از ابزارهای اتوماسیون تست
- راه‌اندازی Test Lab

آموزش و ساخت تیم تست نرم‌افزار:

- بررسی استعدادها، نیازمندی‌ها و منابع شرکت جهت آموزش متناسب
- آموزش مباحث مقدماتی و پیشرفته بر اساس بررسی‌های صورت گرفته
- ایجاد و به روزرسانی فرآیند توسعه و ایجاد فرآیندهای مرتبط با تست
- بررسی امکان مکانیزاسیون تست در حوزه‌های مورد نیاز
- آماده‌سازی و آموزش تیم برای تست مکانیزه

اجرای پروژه تست با نیروی مقیم در سازمان:

- در اینجا تمام موارد مطروحه در «مشاوره» و «آموزش و ساخت تیم تست نرم‌افزار» ارائه می‌شود، با این تفاوت که در اینجا کار به صورت یک پروژه تصور می‌شود، و سرپرستی و همراهی تیم مقیم در سازمان تا زمان خودکفایی تیم یا نیاز سازمان ادامه خواهد یافت

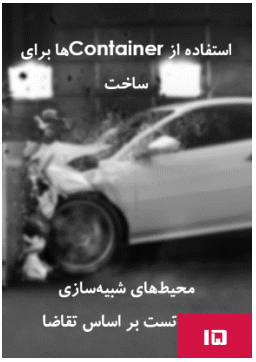
کیفیت، را با تیستن تجربه کنید

www.TISTeN.ir
Info@TISTeN.ir

تیسترِ | دوماهنامه الکترونیکی تخصصی تست نرم‌افزار ایران | اردیبهشت‌ماه ۱۳۹۵ خورشیدی

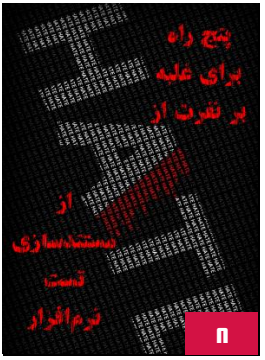
آخرین شمارهٔ تیستر را از اینجا بخواهید

به قلم تحقیق و خبر



به قلم ابوالفضل خواجه دیزجی

به قلم ابوالفضل خواجه دیزجی



به قلم مِلیک وارطانیان

به قلم امیرعلی یاسینی

به قلم هدی رضوی

به قلم بهاره اصغری

آخرین اخبار
از دنیای
تست نرم‌افزار

GENROCKET و HCL مشارکتی برای تنظیم یک استاندارد جدید در آزمایش شتاب(Testing Accelerated) با کیفیت بی‌سابقه(Uncompromised Quality) را تشکیل دادند

GenRocket، یک شرکت فن‌آوری تست در حال ظهور در اوهای(Ojai) کالیفرنیا در تاریخ سوم مارچ ۲۰۱۷ مصادف با سیزدهم اسفند ماه ۱۳۹۵ یک کسب و کار و فن‌آوری با همکاری HCL را اعلام کرد.

GenRocket یک راهکار نرم‌افزاری ابداع کرده است که سرعت و دقت تست نرم‌افزارها را بوسیله خودکارسازی استفاده از کدِ تستِ White Box و تولید دادهٔ تست مبتنی بر مدل(Model Based Test) به صورت بلادرنگ(Real Time) و بر مبنای تقاضا، بهبود می‌بخشد.

با استفاده از GenRocket، تسترها می‌توانند برای بکار انداختن تراکنش‌های Database و Business از هر اپلیکیشنی، به سرعت سناریوهای جامع تست را ایجاد و اجرا کرده و فرآیند تست را به مقدار بیش از ۱۰۰۰٪(ده برابر) سریعتر کنند. از آنجا که تست White Box به صورت سیستماتیک تمام Source Code اپلیکیشن را امتحان می‌کند، پوشش کامل تست(Full Test Coverage) را فراهم کرده و نواقص را در ابتدای چرخه توسعه نرم‌افزار شناسایی می‌کند، آنچنانکه Unit Testing و Complex Integration Testing را با سرعت بالایی انجام می‌دهد.

GenRocket یک تکنولوژی ضروری برای برآوردن نیازمندی‌های همزمان تست مداوم(Continuous Testing) و کیفیت بی‌سابقه(Uncompromised Quality) و یک جزء جدایی‌ناپذیر از چارچوب تست در چرخه حیات نرم‌افزار TM HCL ALMSmartبوده و مولفه‌ای از متدولوژی منعطف Agile می‌باشد که توسط HCL برای سرویس‌های ساخت اپلیکیشن‌های مدرن و پلتفرم دیجیتال مورد استفاده قرار می‌گیرد.

گارث رُز(Garth Rose) به عنوان مدیر عامل GenRocket می‌گوید: "ما به دلیل همتراز شدن با تکنولوژی‌های HCL به عنوان یک پیشرو در میان ارائه‌دهندگان سرویس‌های جهانی IT هیجان‌زده هستیم". وی می‌افزاید: ما به طور کامل به چارچوب استراتژیک ALMSmart و به اشتراک گذاردن آن در چشم‌انداز خود، برای آیندهٔ چرخه حیات تست نرم‌افزار متعهد هستیم".

تحویل مداوم(Continuous Delivery) یک پارادایم جدید برای توسعه نرم‌افزار در عصر دیجیتال است. با پیشرفت سریع اپلیکیشن‌ها برای بازار و بهبود مداوم روی تجربه مشتریِ دیجیتال با نرم‌افزار بدون خطا، استفاده از نرم‌افزار توسط سازمان‌ها به منظور تعامل با مشتریان، به یک ضرورت برای سبقت گرفتن از رقبا تبدیل شده است.

تحویل مداوم(CD-Continuous Delivery)، سرعت توسعه نرم‌افزار را برای نقطه‌ای که درخواست‌های بی‌سابقه در QA و تست نرم‌افزار قرار داده شده است را افزایش می‌دهد. به عنوان یک نتیجه، چرخه تست نرم‌افزار باید به روش‌های زیر تکامل یابد:

- ✓ سرعت تست نرم افزار باید بیشتر باشد
- ✓ حجم عملیات‌های تست باید بالاتر باشد
- ✓ کد باید از قبل در چرخه توسعه تست شود
- ✓ Complex Integration Testing باید ساده باشد
- ✓ داده تست(Test Data) با یکپارچگی ارجاعی(Referential Integrity) باید بر اساس تقاضا در دسترس باشد

HCL یک تکنولوژی بسیار سطح بالا و عملکردهای پیشرفته تست را برای برآوردن این دست از درخواست‌ها توسعه داده است، و شراکت GenRocket، برای تنظیم یک استاندارد جدید برای شتاب دادن به سرعت تست، آن هم بدون افت کیفیت نرم‌افزارِ منتشر شده در محیط بهره‌برداری، نمونه‌ای از نوآوریِ تکنولوژیک است که این شرکت درآن جای گرفته است.

درباره GenRocket

GenRocket یک رهبر تکنولوژی در حال

PARASOFT®

ظهور در تکنولوژی تست نرم افزار است، که به شرکت‌های خدمات IT و مشتریان Enterpriseای که خواهان کیفیت و بهره‌وری برتر در توسعه نرم‌افزار خود هستند سرویس می‌دهد. GenRocket Software، با پلتفرمی که تولید Test Codeهای White Box را اتومات می‌نماید، نقش Complex Integration Testing را مجددا تعریف می‌کند. GenRocket Software، بر مبنای تقاضا، از داده‌های تست(Test Data) بلادرنگ(Real Time) که به محیط‌های Databaseای بزرگ و پیچیده (در عین اینکه Referential Integrity را حفظ می‌کنند) مقیاسبندی می‌شوند، استفاده می‌نماید.

مقر اصلی این شرکت در اوهای کالیفرنیاست. این شرکت از طریق شبکه شرکای تکنولوژی خود، در تعداد زیادی از بازارهای بین‌المللی فعالیت می‌کند. برای کسب اطلاعات بیشتر، لطفا به www.GenRocket.com مراجعه کنید.

CrowdSprint سرویس‌های تست مبتنی بر Crowd خود را برای زیر ۱۰۰۰ دلار، راه‌اندازی کرد

ملبورن، ۸ مارچ ۲۰۱۷ مصادف با ۱۸ اسفندماه ۱۳۹۵، CrowdSprint، به عنوان یک شرکتی فعال در زمینه CrowdSource Testing، اعلام کرد که کلیه نوآوران در سرتاسر جهان که تمایل به تحویل بهترین اپلیکیشن‌ها و وبسایت‌ها دارند، اکنون می‌توانند یک بسته آنلاین تست، که شروع آن با قیمت زیر ۱۰۰۰ دلار است، کار را آغاز نمایند. این پلتفرم جدید برای Crowd Testing قدرت Crowd را برای تست نرم‌افزار به ارمغان می‌آورد، و آنرا برای هر سایز از Business دسترس‌پذیر و مقرون به صرفه نموده، و ایضا در هر نقطه از جهان امکان بهره‌برداری از آن را ممکن می‌سازد.

جِیمی دافیلد(Jamie Duffield) مدیرعامل CrowdSprint می‌گوید: "افرادیکه که اپلیکیشن و وبسایت را در سطح جهانی ایجاد می‌کنند، نیازمند این هستند که آنها را با سرعت و با حداقل هزینه تست نمایند. ما این کار را در راستای یافتن باگ‌ها و نواقص از طریق Crowd، برای انجام تست Functionality، کاربرد پذیری(Usability) و امنیت(Security) ساده کرده‌ایم. چه اپلیکیشن‌های خود را بسازید، و نیازمند یک بررسی QA روی مشتریان اپلیکیشن/وبسایت باشید، چه به صورت داخلی و قبل از لانچ محصول، منابع و زمان را برای تست اجرایی کنید، و چه اینکه بخواهید به سادگی برای شناسایی مشکلات، تست را اجرا کنید(به شکلی که آنها را بهبود دهید)، می‌توانید از هم اکنون یک بسته آنلاین مناسب برای تست را خریداری کرده و اجازه دهید CrowdSprint باقی کارها را انجام دهد."

وی ادامه می‌دهد: "برنامه‌ها و وبسایت‌های دارای ایراد می‌توانند حدفاصل میان موفقیت و شکست برای یک Business باشند، اما اغلب، مردم بر این باورند که می‌توان "تست" را توسط کارکنان، دوستان و خانواده با موفقیت به انجام رساند. در حالیکه اینجا محلی عالی برای آغاز است، تجربه به ما نشان می‌دهد که، با یک بسته در سطح Starter، می‌توانید انتظار عملکرد کشف حداکثر ۴۰ نقص معتبر را از آن داشته باشید. لانچ کردن یک اپلیکیشن با ۴۰ نقص می‌تواند برای هر Businessای فاجعه‌آفرین باشد.

با استفاده از تست روی دستگاه‌ها و سیستم عامل‌های واقعی با کاربران واقعی، Business قادر است به شکل فوق‌العاده‌ای نرخ تشخیص نقص را بهبود داده، هزینه‌های تست را بین ۵۰ تا ۷۰ درصد کاهش داده، زمان رسیدن به بازار را تسریع کرده، و باعث شود بتوانید به صورت بی‌وقفه محصولات جدید دیجیتال را راه‌اندازی کنید."

PARASOFT سرویس مجازی‌سازی و API Testing را برای مشترکین Microsoft Visual Studio Enterprise منتشر کرد

۷ مارچ ۲۰۱۷، مصادف با ۱۷ اسفندماه ۱۳۹۵، Parasoft Virtualize/SOAtest Desktop اجازه داد که Microsoft Visual Studio Enterprise 2017 به مدت ۶ ماه به صورت رایگان به راهکار API Testing با Service Virtualization دسترسی داشته باشد.

Parasoft، به عنوان یکی از پیشروان در راهکارهای تست نرم‌افزار، انتشار Parasoft Virtualize/SOAtest Professional Desktop را برای مشترکان Visual Studio 2017 Enterprise در سطح جهان اعلام نمود. Parasoft Virtualize/SOAtest توسعه‌دهندگان و تسترهای نرم‌افزار را قادر می‌سازد تا به سرعت هر دو موضوع API Testing و مجازی‌سازی(Virtualization) را در سرتاسر طیف گسترده‌ای از پروتوکل‌های استاندارد و Message Format انجام دهند. این بسته‌بندی جدید Microsoft Developers Testing Toolbag را بزرگتر کرده، آنرا با روش‌های کارآمد و موثر برای انجام تست توانمند ساخته، میزان سربار را به حداقل رسانده و در آخر ریسک نواقص را نیز کاهش داده است. برای آغاز و مطالعه بیشتر به این آدرس مراجعه کنید:

http://software.parasoft.com/virtualize/microsoft

مارک لَمبِرت(Mark Lambert) به عنوان معاونت ریاست در امور محصولات(VP Of Products) در Parasoft می‌گوید: "ما از ارائه API Testing و Service Virtualization به مشترکان Microsoft Visual Studio Enterprise بسیار هیجان‌زده هستیم".

وی می‌افزاید: "امروز اپلیکیشن‌های Highly-Connected(قویا همبند) نیازمند تست وسیع در حوزه‌های API، Performance، و Security در محدوده وسیعی از محیط‌ها هستند. Virtualize/SOAtest Desktop به توسعه‌دهندگان اجازه اعتبارسنجیِ سریع و دقیق نیازمندی‌های Functional و Non-Functional را می‌دهد."

- ✓ Parasoft Virtualize/SOAtest Professional Desktop مشترکان Visual Studio Enterprise 2017 را با لیست ذیل مهیا می‌کند:
- ✓ Complete API Testing در سرتاسر طیف گسترده‌ای از پروتوکل‌های استاندارد صنعتی.
- ✓ توانایی برای شبیه‌سازیِ سریعِ Messaging و رفتار سیستم Multi-Tiered وابسته.
- ✓ تست API و پروتوکل به صورت بسیار کارآمد بدون ایجاد سربار در راستای نوشتن استاب‌ها(Stub) یا انتظار کشیدن برای آزاد شدن منابع.
- ✓ صرفه جویی در هزینه با به اشتراک گذاشتن سرویس‌های مجازی کامپونت‌های سیستمی.
- ✓ شش ماه دسترسی آزاد، که در ادامه ۲۵٪ تخفیف برای اشتراک یک ساله ارائه می‌شود.

کریگ کیترمَن(Craig Kitterman) به عنوان مدیر بازاریابی محصول برای ابزارهای توسعه‌دهندگان و سرویس‌های ابری(Visual Studio Product Marketing for Developer Tools & Cloud Services) می‌گوید: "ما به دلیل ارائه دسترسی مستقیم به API Testing و مجازی‌سازی سرویس در محیط توسعه، بسیار از شراکت با Parasoft هیجان‌زده و خوشحالیم". وی می‌افزاید: "استقبال تیم‌ها از DevOps به صورت یک وظیفه دلهره‌آور از اتومات کردن API Testing و تست در مقابل وابستگی‌ها رخ می‌نمایاند. با استفاده از Parasoft SOAtest و Parasoft Virtualize، Visual Studio Enterprise مشترکان می‌توانند به سادگی Test Scenarioهای پیشرفته End-To-End(بی‌واسطه) را ساخته، و تعاملات سیستم‌های مجتمع(Complex System) را در مقابل وابستگی‌های ناقص یا غیرقابل دسترس شبیه‌سازی کنند."

برای اطلاعات بیشتر در موارد ذیل به آدرس‌های مذکور مراجعه کنید:

- ✓ پورتال مزایای Visual Studio:

http://my.visualstudio.com?wt.mc_id=AID605451_QSG_141866

- ✓ صفحه بررسی اجمالیِ مشترکان Visual Studio:

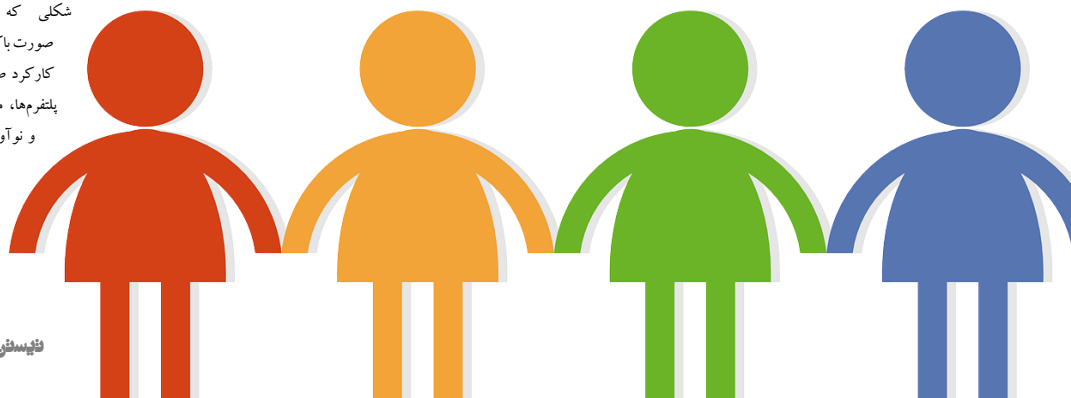
https://www.visualstudio.com/subscriptions?wt.mc_id=AID605451_QSG_141871

- ✓ Parasoft SOAtest-Virtualize Desktop برای مشترکان Microsoft Visual Studio Enterprise:

http://software.parasoft.com/virtualize/microsoft

تقاضا، پیشبینی‌ها و فرصت‌های بازار Cloud Testing و ASQ Software برای سال ۲۰۱۷ تا ۲۰۲۷

با افزایش پیشرفت‌های تکنولوژیکی، نیاز به تست مداوم(Continuous Testing) برای نرم‌افزارها همواره وجود دارد. در این زمان حالت مرسوم تست بسیار وقت‌گیر بوده و هزینه مرتبط با این نوع راهکارِ تست بالاست. به همین دلیل تقاضا برای راهکاری جهت تست اپلیکیشن روی Cloud و برای Cloud به شکلی که نرم‌افزار به صورت باکیفیت و دارای کارکرد صحیح در کلیه پلتفرم‌ها، موقعیت رقابتی و نوآوری تجاری را به ارمغان آورد، به



طور قابل توجهی در حال افزایش است.

فروشندگان نرم‌افزار در سراسر جهان مقدار زیادی از پول خود را در تحقیق و توسعه نرم‌افزار سرمایه‌گذاری کرده‌اند که به این ترتیب می‌تواند زیرساخت IT بیشتری با محوریت نرم‌افزار به مشتریان خود ارائه دهند.

این فروشندگان به دنبال تست اتوماتیک نرم‌افزار(Automated Software Testing-ASQ)، نرم‌افزار به عنوان یک سرویس (Software as a Service-SaaS) و زیرساخت‌های انطباقی هستند که در Cloud پشتیبانی گردد.

Cloud Testing و ASQ Software دسترسی سریع به راهکار برتر و زیرساخت پشتیبانی را برای حفظ تخصیص منبع نرم‌افزارِ مجتمع و توسعه پویا، تسهیل می‌کند. راهکارهای Cloud Testing نیازمند سرمایه‌گذاری کمتری در منابع و زیرساخت به نسبت راهکارهای ASQ هستند.

درخواست نمونه گزارش در:

http://www.futuremarketinsights.com/reports/sample/rep-gb-3088

بازار Cloud Testing و ASQ Software: پیشران‌ها و چالش‌ها

توسعه مداوم(Continuous Development) در Cloud Computing(رایانش ابری) محرک رشد بازار جهانی ASQ Software و Cloud Testing است. Cloud Computing تغییری جدید در مدل IT ایجاد کرده است. Cloud Computing فضای سازمان‌ها را جهت اتخاذ SaaS با یک هزینه بسیار پایین، تسهیل می‌کند. SaaS، سازمان تجاری را برای یک Framework چابکتر مهیا ساخته و بهره‌وری را در همان زمان افزایش می‌دهد. SaaS یک پدیده پیچیده است و نیازمند نظارت مستمر و مداوم می‌باشد. همانطور که سازمان‌ها بیشتر، راهکارهای همراه و اپلیکیشن موبایل را مستقر کرده و از آن استفاده می‌کنند، فروشندگان Cloud Testing و ASQ Software نیز شاهد فرصت بزرگی در بازار هستند.

با این حال نرم‌افزار سازمان‌های تجاری، در حال تغییر هستند و چون مقابله با این پیشرفت‌های نرم‌افزاریِ سریع‌التغییر برای فروشندگان Cloud Testing و ASQ Software بسیار سخت است، این موضوع به بزرگترین چالش فعلیِ بازار ASQ Software و Cloud Testing تبدیل شده است.

بازار Cloud Testing و ASQ Software: بازیکنان کلیدی

تعداد کمی از شرکت‌ها در بازار Cloud Testing و ASQ Software حضور دارند که عبارتند از: Parasoft، SOASTA، Compuware، Hewlett Packard Enterprise، IBM Corporation، Microsoft Corporation، Skytap، Micros Focus، SamrtBear و Cast.

درخواست برای لیست موضوعات:

http://www.futuremarketinsights.com/toc/rep-gb-3088

بررسی اجمالیِ بازار Cloud Testing و ASQ Software:

آمریکای شمالی بزرگترین بازار در حوزه Cloud Computing است، و از این رو فرصتی بزرگ برای فروشندگان Cloud Testing و ASQ Software به شمار می‌رود. پایه بسیاری از فروشندگان Cloud Testing و ASQ Software مانند IBM Corporation و Microsoft Corporation نیز در ایالات متحده است.

بازار رایانش ابری(Cloud Computing) در منطقه آسیا-اقیانوسیه به نسبت آمریکا شمالی و اروپا با نسبت آرام‌تری در حال رشد است، چرا که بسیاری از این کشورها در این منطقه هنوز در مورد خدمات رایانش ابری دچار تردید هستند.

تحقیق و خبر


پنج راه
برای غلبه
بر نفرت از
مستندسازی
تست
نرم‌افزار

نوشتن اسناد تست کار خوبیست، که باعث می‌شود یک فرآیند فکری منظم اجرا شود، و توضیح می‌دهد که شما چه چیزی را برنامه‌ریزی می‌کنید و علاوه بر این استراتژی تست را نیز بهبود می‌بخشد. اما دانستن فواید این کار باعث لذت بخش شدن آن نمی‌شود. من به رازهای این موضوع پی برده‌ام. بنابراین ۵ نکته در این مقاله ارائه می‌دهم که به ایده مستندسازیِ ساده‌تر در تست کمک می‌کند، و یا حداقل باعث می‌شود که نفرت از این کار کمی مشکل‌تر شود.

تا جاییکه من می‌دانم هنوز یک گروه پشتیبان برای تسترها وجود ندارد. ولی اگر وجود داشت من معتقد بودم که خیلی از ما اعتراف می‌کردیم که بیزاری شدیدی نسبت به نوشتن مستندا ت تست داریم.

قسمتی از تعریف شغل من این است که فردی آزاردهنده هستم که مردم را به نوشتن مستندات مجبور می‌کند. من الگوهای سند را تعریف کرده و اکثر مستنداتی که تیم من ایجاد می‌کنند را مرور می‌کنم. می‌توانم به شما بگویم این تنها چیزیست که شما تصور می‌کنید: کاری بدون هیجان و با کمترین جذابیت نسبت به اجرای تست و پیدا کردن یک باگ جالب.

موضوع از این هم بدتر است زیرا من در مورد مستندات تست موعظه هم می‌کنم. من مجبورم که یک نمونه خوب باشم و آنچه را که از دیگران می‌خواهم خودم هم انجام دهم. من نمی‌توانم نوشتن مستندات تست پروژه‌ای که خودم مسئولش هستم را از قلم بیندازم حتی با اینکه واقعاً از آن لذت نمی‌برم. در واقع وقتی مستندات تست را می‌نویسم مجبورم که بلند شوم و در راهرو قدم بزنم تا بیدار بمانم.

هرچند نوشتن مستندات تست یک وظیفه جالب نیست، اما کار خوبیست که باید آنرا انجام داد. این کار یک فرآیند فکری منظم را در مورد چالش تستِ پیش رو ایجاد می کند. در این مرحله در مورد آنچه برنامه‌ریزی می‌کنیم و اینکه چرا کارها را باید به این روش انجام دهیم و اینکه چرا به روش دیگر انجام نمی‌دهیم تشریح می‌شود. علاوه بر این برنامه عملیاتی(Action Plan)، استراتژی تست، برآورد تلاش (Effort Estimation) و Task Partition را بهبود می‌بخشد.

همچنین برنامه‌های(Plan) مستند شده به ذینفعان اجازه می‌دهد تا متوجه شوند چه انتظاراتی باید از تیم اعتبارسنجی(Validation) داشته باشند. این کار به ما اجازه فیدبک(بازخورد) می‌دهد. در صورتیکه ما در برنامه‌هایمان اشتباهاتی داشته باشیم، تعداد سورپرایزهایی که به علت سوءتفاهم یا انتظارات ناجور ایجاد می‌شوند، بر این اساس کاهش خواهند یافت.

هیچکدام از اینها اخبار جدیدی برای شما نیست، و کماکان این فرآیند برایتان جذاب نیست. در هر حال شاید من بتوانم تصور مستندسازی تست را کمی آسان‌تر کنم یا حداقل کمی خوش طعم‌تر. شما به هرحال مجبورید که این کار را انجام دهید، بنابراین چرا تلاش نکنید که بر نفرت خودتان از مستندسازی تست غلبه کنید و بعلاوه همت خود را به کار بندید تا در وضعیت روحی بهتری در مورد آن قرار گیرید.

نکات پنجگانه

۱. سند تست را برای خودتان بنویسید

به این واقعیت واقف باشید که افراد بسیار کم(اگر صفر نباشد) سند تست را خواهند خواند. این خوب است. بهره‌بردار اصلی سند تست، نویسنده آن سند است. نوشتن به

شما کمک می‌کند که تفکرات خود را نظم دهید و برنامه‌های تست را به شکل گام‌های واضحی سازماندهی کنید، طوری که خودتان(یا هر فرد دیگری) بتواند آن را بفهمد.

پ۹/وانگهی، اگر شما انتظار نداشته باشید که کسی سند را بخواند، وقتی چنین اتفاقی بیفتد، نومید نخواهید شد. کتاب "کاوش نیازمندی‌ها: کیفیت قبل از طراحی"(Exploring Requirements: Quality Before Design) نوشته دونالد سی. گاس(Donald C. Gauss) و جرالد وینبرگ(Gerald Weinberg) به این ناامیدی مرتبط است. آنها می‌نویسند که سند هیچ چیز نیست، بلکه مستندسازی همه چیز است.

نوشتن سند هدف است. اگر توسط کسی خوانده شود، یک پاداش برای مستندساز است.

۲. از فرآیند فکریِ انسانی خود استفاده کنید

در جهان توییتری، امروز ما احساس می‌کنیم که هر ایده‌ای می‌تواند در یک یا دو جمله منتقل شود، و اگر به بیشتر از این نیاز داشته باشیم کسی باید یک ترفند پیدا کند یا یک Application را توسعه(Develop) دهد که چنین کاری را برای ما انجام دهد. ما معتقدیم که تنها اگر به اندازه کافی و سخت فکر کنیم، یک راه حل فنی پیدا خواهیم کرد که برای ما مستندات را بنویسد، مانند Doxygen برای کد کردن مستندات.

وقتی که تمام وقت حس می‌کنید می‌توان کار را اتوماتیک نمود و آنرا به شکلی کاراتر انجام داد، صَرف زمان خیلی زیاد برای انجام یک کار به صورت دستی خسته کننده است. ولی نوشتن یک سند یک فعالیت فکری است و خوشبختانه هنوز هیچ کس نتوانسته بفهمد چطور می توان آن را اتوماتیک کرد. لذا دست از شکنجه دادن خود برای هدر دادن زمان بردارید و از این واقعیت که این کار یک کار انسانی است و جایگزینی ندارد لذت ببرید.

۳. تایپ لمسی را یاد بگیرید

قسمتی از بی‌میلی برای نوشتن مستندات، از این واقعیت سرچشمه می‌گیرد که انجام آن زمان زیادی را به خود اختصاص می‌دهد. این کار حتی زمان بیشتری را صرف می‌کند اگر شما در تایپ فقط از دو انگشت استفاده کنید یا برای نگاه کردن به کلیدها(صفحه کلید) ، مجبور به توقف شوید.

تایپ لمسی، مستندات تست شما را پیشرفت نخواهد داد ولی وضوح گزارش‌های باگ شما، اسناد کد شما و ایضا ایمل شما را بهبود می‌بخشد. وقتی تایپ کردن سخت باشد شما رها کردن نوشتن تمایل پیدا خواهید کرد یا مطالب خود را در کلمات تک سیلابی بیان خواهید نمود.

وقتی تایپ کردن ساده باشد نوشته شما برای خواندن، روان، واضح و آسان خواهد شد.

۴. تسلط بر زبان

برای افراد غیر انگلیسی زبان که نیاز دارند مستندات تست را به زبان انگلیسی بنویسند(یا هر زبان دیگری که در سطح زبان مادری به آن مسلط نیستند)، موضوع زبان برای نوشتن خوب و کارا یک مانع است. اگر شما به خاطر اینکه زبان مستند خارجیست، برای تنظیم یک جمله تقلا کنید، نوشتن زمان زیادتری نیاز دارد و می‌تواند به شدت خسته کننده یا نومیدکننده باشد.

برای کمک به اصلاح روند (فرآیند) ابتدا بپرسید که سند باید به زبان خارجی باشد یا نه. اگر به زبان خارجی بود بررسی کنید که آیا می‌توانید به زبان مادری خود بنویسید و یک مترجم سند داشته باشید. برای شرکت شما احتمالا زمان‌تان خیلی مهمتر و با ارزش تر از هزینه یک مترجم است.

در صورتیکه می‌دانید محیط تجارت در شرکت شما به گونه‌ایست که همیشه نیازمندید تا با زبان خارجی کار کنید، تلاشتان را روی بهبود تسلط خود بر زبان سرمایه‌گزاری کنید. ممکن است در اینجا شرکت شما موافق پوشش هزینه‌هایتان خواهد بود.

یکی از راههای یادگیری یک زبان جدید که برای من خیلی خوب نتیجه داد خواندن زیاد مطالب به آن زبان است. در دبیرستان من مجبور بودم زبان انگلیسی خود را ارتقا دهم در همان زمان نیز داستان‌های علمی تخیلی را کشف کردم. برای مدت سه سال منحصراً داستانهای علمی تخیلی را به زبان انگلیسی خواندم. نه تنها من قوانین سه‌گانه رباتیک و Grok و اینکه Grok چه معنی می‌دهد را می‌دانستم بلکه در نوشتن انگلیسی هم خیلی بهتر شده بودم.

کتاب‌هایی با موضوعاتی که دوست دارید در زمینه زبانی که نیاز دارید یاد بگیرید، پیدا کنید و در زمان یاد گیری از آنها لذت ببرید.

۵. یادبگیرید چطور بنویسید

من فقط در مورد اینکه چطور مستندات را بنویسید صحبت نمی‌کنم. بلکه یاد بگیرید هرچیزی را چطور بنویسید. به عنوان مثال نوشتن خلاقانه. هرچه نوشتن عمومی روان باشد، نوشتن مربوط به اسناد کاری نیز آسان‌تر خواهد بود.

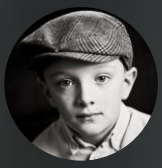
جوئل اسپالسکی(Joel Spolsky) به عنوان یک مهندس نرم‌افزار در مقاله خودش(مقاله‌ای که حتما باید بخوانید) در مورد اهمیت مستندات نیازمندی‌ها، می‌گوید: نوشتن ماهیچه است. هرچه بیشتر بنویسید بیشتر قادر به نوشتن خواهید بود. اسپالسکی بر اساس تجارب خودش تصدیق می‌کند که برای غلبه بر ترس خود از نوشتن، در کالج یک کلاس انتخاب کرد که در آن ملزم بود هر هفته یک انشاء بنویسد.

اگر شما در هنگام نوشتن مستندات خود، به مانع نویسنده گیر کردید، سعی کنید ابتدا هر چیزی را بنویسید! هرچیزی! پس از آن شما کلمات را خواهید یافت که به سادگی جاری می شوند.

فقط آن را انجام دهید

چیزهای زیادی در زندگی وجود دارند که شما انجام می‌دهید، چون باید آنها را انجام دهید نه به خاطر اینکه تمایلی به انجام دادن آنها دارید. مانند غذا خوردن، کار کردن، جلسات انجمن اولیاء، نخ دندان و ... . بنابراین مستندکردن تست را به این دسته بندی اضافه کنید. و تعویق را متوقف نمایید. تجربه من این است که وقتی نوشتن مستندات تست را به موقع انجام می‌دهید(در شروع پروژه نه وقتی رو به پایان است)، در اغلب مواقع شما احساس می‌کنید که تلاشتان با ارزش بوده و مزایایی به دست آورده اید.

امیدوارم این راهنمایی ها توانسته باشد به شما کمک کند.

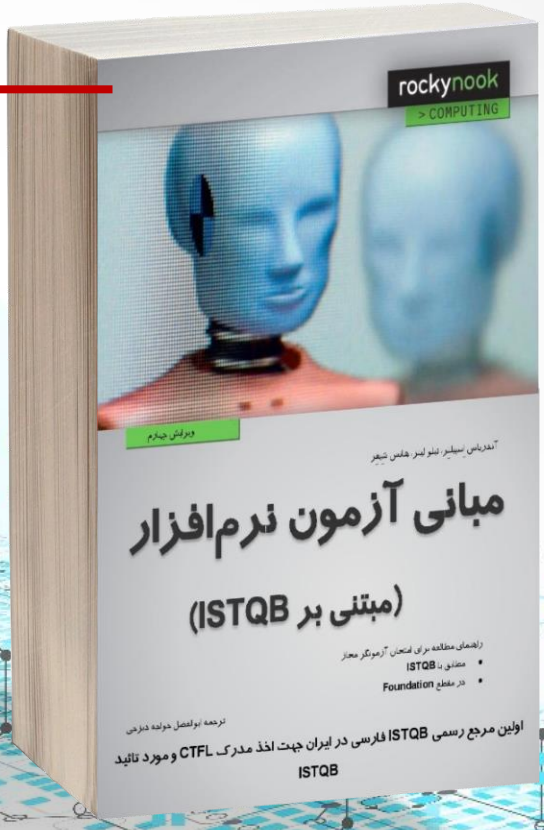


مِلیک وارطانیان

شابک: ۰-۹۸-۷۹۸۳-۶۰۰-۹۷۸

۴۰۲ صفحه

ISTQB®
International Software Testing Qualifications Board



برای اولین بار در ایران، منبع رسمی ISTQB با ترجمه فارسی

ترجمه: ابوالفضل خواجه دیزجی
(مدیر و مشاور آزمون نرم‌افزار)

برای اطلاعات بیشتر به www.TISTeN.ir مراجعه کنید

DevOps

همکاری

برای یک

هدف

خلاصه:

اهمیت همزیستی میان توسعه، بهره‌برداری(Operation) و تضمین کیفیت(QA)، مدت‌های مدیدیست که شناخته شده است. ولی هنوز هم آنها رابطه ضعیف یا نامتعادلی دارند. DevOps برهمکاری و رد طرز فکر «ما علیه آنها» تاکید دارد. هر واحد نیازمند اطلاعات، بازخورد و پشتیبانی از سوی سایر واحدهاست. این کار به همه کمک می‌کند تا ببینند چگونه یکدیگر را توانمند می‌کنند.

سه گروه با مهارت‌های مختلف را تصور کنید که باید با استفاده از ترکیب مهارت‌هایشان کاری را انجام دهند. آیا آنها باید الف. به طور مستقل کار کنند در حالیکه دانش خود را جداگانه نگه می‌دارند. ب. دانش خود را به اشتراک بگذارند ولی مهارت‌هایشان را برای خودشان نگه دارند. پ. به سوی یک هدف مشترک در حالیکه دانش و مهارت هایشان را برای کمک به یکدیگر به اشتراک می گذارند، با هم کار کنند تا بتوانند فورا با استفاده از بهترین عملکردها(Best Practice) با هر مشکلی برخورد کنند؟ خیلی سخت نیست که گزینه پ را پاسخ ایده آل ببینیم.

فرض کنیم گرایش سوال به سمت DevOps و سه گروه مشتمل بر توسعه، بهره‌برداری، و QA می‌باشد. اکنون اجازه دهید نگاهی دقیقتر به آنچه که در این موضوع بدان پرداخته می‌شود، داشته باشیم.

چرخه حیات توسعه نرم افزار Agile

بگذارید در حالیکه این سه واحد و مسوولیت‌هایشان را تحلیل می‌کنیم، ابتدا آنها را در چارچوب پدر DevOps در نظر بگیریم: یعنی Agile.

تیم توسعه (Development) حافظ کد است. آنها مالک نرم‌افزار، توسعه و اتوماسیون بوده و مسئولیتهای‌شان شامل کدنویسی، ایجاد Buildها، استقرار و تست است. مدیر ساخت مسئول ایجاد Buildها و استقرار است. بازبینی کد به صورت Pre-Build(قبل از Build) و بواسطه برنامه‌نویسی دونفره(Pair Programming) انجام می‌شود، و هر روز تا قبل از روز آخرِ اسپرینت به صورت Post-Build(پس از Build) به وسیله Unit Testing از سوی QA پیگیری می‌شود، که اغلب آزمون رگرسیون را تقویت می‌کند.

تیم بهره‌برداری(Operations) حافظ زمانبندیست. آنها بینش خیلی خوبی در این مورد دارند که چطور هر پروژه با زمانبندی سازگارشود. این موضوع در حالیست که تیم توسعه و تیم تضمین کیفیت تمایل دارند بتوانند در هر زمان روی هر نقطه‌ای از پروژه دید عمیقی داشته باشند. تیم Operations در نظارت(Monitoring)، تحلیل و مدیریت مهارت داشته و مسئول تغییرات، استقرار و باثبات‌سازی است. معمولاً تضمین کیفیت(QA) در روز آخر اسپرینت Build را به Operations منتقل می‌کند.

تضمین کیفیت(QA) حافظ تصدیق(Verification) و اعتبارسنجی(Validation) سیستم تحت تست(SUT-

System Under Test) است که بر اساس آن باید سیستم کاری را انجام دهد که از آن انتظار می‌رود. این موضوع شامل این موارد می‌باشد: ایجاد موارد مرزی(Edge Cases)، ایجاد انتظارات شناخته شده از آنها، ابراز هر نوع نگرانی در مورد کاربردپذیری(Usability) و ایضا حصول اطمینان از اینکه مراقبت درستی از نیازهای کسب و کار(Business) و مسائل مرتبط با انطباق(Compliance-در ایزو ۹۱۲۶ شش شاخص برای کیفیت نرم‌افزار تعریف شده است که هر یک دارای زیرشاخص‌هایی می‌باشند. در تمامی این شش شاخص یک زیرشاخص به نام Compliance وجود دارد، که تطابق آن شاخص با استانداردهای Regulatory در زمینه همان شاخص را می‌سنجد. برای اطلاعات بیشتر به مستندات ایزو ۹۱۲۶ مراجعه فرمایید) به عمل آمده است یا خیر. علاوه بر این، آنها مسئول امضا و تائید کسب و کار، تست پذیرش کاربر(User Acceptance Test)، Performance و امنیت نیز هستند.

امکان بسیار زیادی وجود دارد که این نوع از محیط‌های Agile اشتباه پیش بروند. چرخه Build روزانه کوتاه و اسپرینت ده روزه به طور جدی کیفیت، کار توسعه جمعی، تست Non-Functional و پشتیبانی بعد از توسعه را محدود می‌کند. این شعار تبلیغاتی را تصور کنید: "ما به سرعت آن را انجام می دهیم اما ...". علاوه بر این، توسعه و بهره‌برداری بدون اینکه کنترلی نیاز داشته باشند به هم وابسته هستند، که این موضوع می‌تواند منجر به بی‌اعتمادی شود. بین جلسات و برنامه‌نویسی دو نفره، ممکن است تعداد زیادی از روزهای تیم توسعه از بین بروند.

بهبود فرآیند از طریق DevOps

DevOps یک تغییر فرهنگی را پیاده‌سازی می‌کند که بر این موانع غلبه کرده و همه چیز را تا جاییکه ممکن است با تضمین کیفیتی، که توانمندیِ عمومی را به ارمغان می‌آورد، ساده می‌کند.

DevOps را نمی‌توان مستقلا با مطالعه آن بدست آورد. این کار نیازمند کسب تجربه از طریق غوطه ور شدن در آن است. این کار فقط در مورد قرار دادن بلوک‌ها روی یکدیگر به منظور رسیدن به یک راهکار نیست، بلکه درباره ملاتیست که این بلوک‌ها را در کنار یکدیگر نگاه می‌دارد. تعامل و همکاری، قلبِ چگونگی کارکردِ DevOps است.

بسیار خوب. تصور کنید که شما تیم توسعه هستید. شاید شما چندین بار در یک روز در حال تحویل قسمتی از یک کد جدید باشید. ممکن است شما قطعه بزرگی ایجاد کنید که تحویل آن چندین روز طول بکشد. در اینصورت شما نیازمند تست مداومِ سریع با بازخورد سریع هستید.

با DevOps، تست قادر خواهد بود با سرعت چندین بار در یک روز انجام شود، حتی اگر تیم تضمین کیفیت بیرون در حال خوردن ناهار باشد. در DevOps، تیم QA مهارت‌های فنی‌ای بدست می‌آورد که بواسطه آن خواهد توانست Automation Test Scriptهای بسازد که قادر هستند از قبل در فرآیند استقرار و Build جاسازی شوند. تست Nonfunctional(امنیت، Performance و...) به شکلی که برنامه‌نویسان در زمان کدنویسی فکر کردن را یاد بگیرند، از قبل در Test Scriptها به مقدار زیادی جاسازی می‌شود. توسعه، Buildها و محیط‌ها را اتوماتیک می‌کند. به این ترتیب زمانی هدر نخواهد رفت. بسیار خوب، حالا تصور کنید که شما در تیم بهره‌برداری هستید. چه چیزی نیاز دارید؟ متریک. شما می‌خواهید آخرین باگ ثبت شده، باگ‌های اصلاح شده و Test Caseهای اجرا شده را بدانید. بعلاوه عادت دارید که به صورت ۲۴/۷ به Bug Tracker و ابزارهای مدیریت تست Login نمایید، تا بدین ترتیب مرتب آنها را در اکثر اوقات روز استخراج نمایید.

QA یک بار دیگر نیز فعال می‌شود، و پس از اینکه مجموعهٔ تست‌های اتومات به سرعت اجرا شد، این متریک‌ها را به صورت خودکار ارائه می‌دهد. تیم بهره‌برداری پاسخگویی بیشتری را بدست می‌آورد که البته آنرا دوست دارد. با رشد اتوماسیون مشکلات کمتری در زمینه Production و بهره‌برداری ظهور می‌کند، که منجر به ثبات بیشتر خواهد شد. چیزی که آنها باز هم آنرا دوست دارند. به این ترتیب زمان بیشتری برای تمرکز بر توان عملیاتی به دست خواهد آمد.

برای تضمین کیفیت این به معنی حجم بیشتر در بهبود مهارت و تغییرات نقش، مجاز دانستن کار بیشتر در صف مقابل اتوماسیون، تمرکز بیشتر روی فرآیند تست، و زمان پیگیری بیشتر روی آموزش، بهبود مداوم(Continuous Improvement) و توسعه مداوم(Continuous Development) است. QA، تست آنها را به شکلی که توسعه مانعی برای آن نباشد اتوماتیک کرده، متریک‌ها را به شکلی که Ops منتظر مطلع شدن از آنها بماند اتومات کرده، و درباره تست رفتاری(Behavioral Test) به طوری که Dev و Ops یک زبان مشترک را ارائه دهند، صحبت می‌کند. قسمت جالب قضیه جاییست که هر وقت توسعه، QA، یا Ops، رؤسایی پرغرور دارند(که چرا واحد و دپارتمان آنها یک دپارتمان خاص است)، باز هم هرگز حسِّ "ما در مقابل آنها" وجود خارجی ندارد. همه ما می‌دانیم که نیازمند همزیستی هستیم اما حالا به وضوح یکدیگر را توانمند می‌کنیم، بخصوص در تضمین کیفیت. DevOps ما را مجبور می‌کند که بازخورد نیازهای هر واحد را از مجموع اطلاعاتی که جمع‌آوری کرده‌ایم در سریعترین زمان ممکن گزارش دهیم.

این همکاری بهبود یافته واقعاً چیزی بیشتر از یک تغییر طرز تفکر، استفاده از Shift-Left، تحویل مداوم(Continuous Delivery) و بهبود مداوم(Continuous Improvement) است. صادقانه بگوییم، نمی‌دانم چطور یک متدولوژی پیگیری می‌تواند بر روی این مدل بهبود یابد؟ آیا راهی برای الزام در راستای نوآوری مداوم(Continuous Innovation) وجود دارد؟ آیا یک سلاح سِرّی که همکاری و همبازی شدن با دیگران را آسانتر کند، وجود دارد؟ فکر می‌کنید چطور می‌توانیم فرآیندها را در آینده بهبود بخشیم؟

.

هدی رضوی


استفاده از Containerها برای
ساخت
محیط‌های شبیه‌سازی
شده تست بر اساس تقاضا

خلاصه: اتخاذ مجازی‌سازی سرویس می‌تواند به سازمان‌ها اجازه دهد که با استفاده از حذف گلوگاه‌های سنتیِ محیط تست، به توسعه و تست موثرتر در زمینه نرم‌افزار دست یابند. ادغام مجازی‌سازیِ سرویس در تحویل مداوم(Continuous Delivery) با استفاده از Containerization، به تیم‌ها کمک می‌کند که به سطحی از انعطاف‌پذیری برسند که بازار رقابتی امروز به آن نیازمند است.

در نبرد شدید برای جلب توجه و رضایت مشتری، سازمان‌ها به طور مداوم در جستجوی افزایش انعطاف پذیری خود در زمان ورود به تولید نرم‌افزار به منظور واکنش به تقاضای بازار و یا حتی فراتر از تقاضای بازار هستند. یک رویکرد معروف برای بدست آوردن چنین چیزی اتخاذ مدل تحویل مداوم(Continuous Delivery-CD) به عنوان مدل توسعه و تحویل نرم‌افزار است.

برای آن دسته از شما که با تحویل مداوم(CD) آشنا نیستند، یک جمله خلاصه از فلسفه پشت آن ممکن است چیزی شبیه به این باشد که:

تحویل مداوم(CD) یک رویکرد مهندسی نرم‌افزار است که در آن تیم‌های توسعه روی ایجاد نرم‌افزار در چرخه‌های(Cycle) کوتاه، و در نتیجه حصول اطمینان از اینکه نرم‌افزار می‌تواند به صورت ایمن در زمان مقرر در محیط بهره‌برداری منتشر گردد، تمرکز می‌کند.

پایه‌های فلسفه CD در کلمه ایمنی نهفته است. CD یعنی اطمینان از اینکه هر نسخه ارائه شده از نرم‌افزارِ تحت توسعه می‌تواند به صورت ایمن در بهره‌برداری منتشر شود. تیم توسعه باید قادر باشد که به روال‌های تست و دروازه‌های کیفیت که قسمتی از خط(Pipeline) تحویل مداوم(CD) هستند، اطمینان نماید. اغلب قسمت بزرگی از این معیارهای کیفی شامل چک کردن‌های اتوماتیک، اعم از Unit Test روی تمام مسیرها برای سطح End-To-End است.

به منظور فراهم کردن کردن‌های اتوماتیک را بتوان بر اساس تقاضا(که به عنوان تست مداوم شناخته می‌شود)، بدون مراقبت(یعنی هیچ مداخله دستی برای اجرا و تفسیر نتایج نیاز نباشد) و در هر زمان که نیاز باشد، اجرا نمود. به منظور حصول اطمینان از اینکه تست‌ها سریعا اجرا می‌شوند، و مقادیر منفی غلط(False Negative-که به صورت غیرضروری خط CD را متوقف خواهد کرد) یا مثبت غلط(False Positive-که یک حس نادرست از امنیت ایجاد می کند) تولید نمی‌کنند، نیازمند مهندسی دقیق روی Testing Suiteهای اتوماتیک هستیم تا بدین ترتیب مطمئن شویم که تست‌ها سریع اجرا شده و Test Resultها به شکلی شفاف و بدون ابهام برای موتور خط(Pipeline Engine) منتشر می‌شوند.

قسمتی که اغلب در ایجاد این بررسی‌های اتوماتیک نادیده گرفته می‌شود، توانمند کردن تیم‌ها به منظور اتخاذ رویکرد تحویل مداوم(CD) می‌باشد، در حالیکه این تلاش، قابلیتست برای تست مداوم مکان‌ها در Test Environmentها. بویژه با اپلیکیشن های مبتنی بر سرویس(میکروسرویس) و توزیع شدهٔ(Distributed) مدرن و امروزی، به سادگی نمی‌توانید یک اپلیکیشن یا یک کامپوننت را به صورت ایزوله تست کرده و آن را به صورت

تحویل مداوم(CD) به شکل درست، بسیار ضرورت دارد که این چک

ایمن در یک محیط بهره‌برداری منتشر کنید. شما به یک محیط تست(Test Environment) نیاز خواهید داشت، که با تمام وابستگی‌های(Dependency) لازم(به اندازه و بر اساس درخواستیست که Test Suite آنرا اِعمال می‌کند) تکمیل می‌شود. این بدین معنیست که همه وابستگی‌ها نیاز دارند که منسجم اجرا شده و تست‌های End-To-End (Unit Testها اغلب از Mock Objectها برای وابستگی‌های دور انتزاعی یاAbstract Away Dependency، استفاده می‌کنند) باید در دسترس بوده و در همه زمانها در حالت درست باشند.

هر کسی که تا به حال درگیر تست کردن Applicationهای توزیع شده(Distributed) بوده می‌داند که این کار یک کار بزرگ و ایضا ساده نیست. وابستگی‌ها(Dependency) که برای تکمیل یکپارچه‌سازی و تست‌های End-To-End ضروری هستند، غالبا به برخی دلایل ذیل یا به سختی در دسترس هستند یا به سادگی قابل حصول نیستند:

- ✓ خود وابستگی تحت توسعه است، که آنرا برای استفادهٔ تست دسترس‌ناپذیر یا نامناسب می‌کند.
- ✓ آماده‌سازیِ Test Dataهای مورد نیاز برای اجرای تست‌ها، بسیار سخت(یا حتی ناممکن) است.
- ✓ اشتراک وابستگی بین تیم‌ها بدین معنیست که وابستگی‌ها می‌توانند تنها در نقاط زمانی خاصی مورد استفاده قرار گیرند(مینفریم‌ها در محیط‌های تست اغلب از این پدیده رنج می‌برند).
- ✓ Dependency، یک کامپوننت ثالث است که نیازمند هزینه‌های دسترسیست که باید قبل از اینکه کسی اجازه داشته باشد از آنها استفاده کند، این هزینه‌ها را تقبل کرد.

رویکردی که در رابطه با محدودیت‌های محیط تست با موفقیت اثبات شده است، مجازی‌سازی سرویس (Service Virtualization) است. حتی با وجود SV، این رفتار مهم هنوز به سختی به وابستگی‌ها(Dependency) که در دارایی‌های مجازی(Virtual Asset) شبیه‌سازی شده‌اند، دسترسی دارد. این دارایی‌های مجازی اجازه می‌دهند که تیم‌های توسعه مجدداً کنترل محیط تست را بدست آورده و نتایج ذیل حاصل شود:

- ✓ تستِ زودتر: نیازی نیست منتظر وابستگی‌ها بمانیم تا بعداً در فرآیند توسعه در دسترس قرار گیرند.
- ✓ تست بیشتر: با دارایی‌های مجازی که کاملا تحت کنترل تیم توسعه است، سِتاپ کردن آنها با چنین روشی(به عنوان نمونه، استفاده از Test Data خاص یا ویژگی‌های Performance

استفاده از Containerها برای ساخت محیط‌های شبیه‌سازی شدهٔ تست بر اساس تقاضا

خاص) برای شبیه‌سازی کردن رفتار در موارد مرزی(Edge Case) ساده است. در غیر این حالت ستابِ برای تیمی که با وابستگی واقعی سرو کار دارد بسیار سخت یا غیرممکن می‌باشد.

- ✓ تست بیشتر موارد: دسترسی نامحدود به دارایی‌های مجازی و ایضا رفتاری که آنها اِعمال می‌کنند، اجازه می‌دهد که تیم‌های توسعه، آنها را بر اساس تقاضا ارائه و مجددا تنظیم کرده و اساسا یک محیط جدید تست را به سادگی برای هر اجرای تست ایجاد نمایند.

مجازی‌سازی سرویس(SV) در طول چند سال گذشته، شاهد رشد به کارگیری خود به عنوان راهکاریست که به سازمان‌ها اجازه می‌دهد تا تلاش‌های خود در راستای تست را بواسطه شبیه‌سازیِ رفتارِ وابستگی به صورت موثر و زیرکانه بهبود دهند. گام بعد در بنیان کردن توسعه، تست و استقرار(Deployment) منعطف‌تر و بر اساس تقاضا(اساساً مبتنی بر بلندهمتی سازمان‌ها برای اجرای CD است)، تلقی کردن محیط‌های شبیه‌سازی شده تست به عنوان یک فرآورده(Artifact) در فرآیند CD است، و این یعنی دقیقا مشابه با آنچه که پیش از این با Test Suiteهای اتومات به طور منظم انجام می‌شده است. این بدین معنیست که توسعه وابستگی‌های شبیه‌سازی شده، به عنوان یک وظیفه(Task) از توسعه تلقی شده و ایضا دارایی‌های مجازی به همراه دستورالعمل بهره‌برداری(Production Code) و تست‌های اتوماتیک از طریق خط تحویل مداوم(CD)، منتشر می‌شوند.

در حال حاضر، برای رسیدن به چنین چیزی، چند راهکار مجازی‌سازی سرویس(SV) شما را قادر خواهند ساخت تا محیط‌های شبیه‌سازی شده تست را مانند Container(نگهدارنده) حرکت دهید، درست مانند زمانیکه با اپلیکیشن تحت تست کار می‌کنید و احتمالاً در این زمان با Test Suiteهای اتومات مربوطه همراه می‌شوید. یک چرخه بالقوه در فرآیند تحویل مداوم (CD) به شرح ذیل است:

۱. یک توسعه‌دهنده تغییری را به مخزن مرکزیِ کد ارسال می‌کند، مانند Git.
۲. Build Server یک Build جدید را راه‌اندازی کرده و Unit Testها را اجرا می‌کند.
۳. بعد از اینکه Unit Testها پاس شد، Build با استفاده از Containerها در محیط تست مستقر(Deploy) می‌شود.
۴. به موازات گزینه ۳، وابستگی‌های مجازی‌سازی شده نیز مهیا و مستقر می‌شوند، همچنین با استفاده از Containerها نقاط پایانیِ مد نظر، Test Data Setها، و مشخصات Performance نیز مستقر می‌گردند.
۵. تست‌های Integration و End-To-End اجرا شده و اپلیکیشن تحت تست و وابستگی‌های شبیه‌سازی شده آن، آنچنانکه مد نظر است، به کار انداخته می‌شوند.
۶. بعد از اینکه تست‌ها پاس شدند، اپلیکیشن به صورت ایمن در محیط بهره‌برداری مستقر می‌گردد.
۷. محیط تست شبیه‌سازی شده تخریب شده و برای چرخه بعدی آماده می‌شود.

توجه داشته باشید که توالی اَعمال بالا به یک تغییر در اپلیکیشن تحت تست مربوط می‌شود. یک چرخه مشابه می‌تواند برای یک ورژن به روز شده از هر دارایی مجازیِ مورد استفاده، تکمیل شود.

به دلیل اینکه این موارد، قسمت‌های اصلی فرآیند توسعه هستند باید تحت تست قرار گیرند، همانطور که این کار باید با کد تولیدی شما انجام شود.

در اینجا برخی از بزرگترین مزایای استفاده از مجازی‌سازی سرویس به صورت کانتینریزه(Containerize) تشریح شده است:

- ✓ ایجاد همان محیط تست قبلی آنچنانکه در تست گذشته اجرا شده است.
- ✓ دارایی‌های مجازی می‌توانند به سادگی به عنوان یک فرآورده در فرآیند توسعه نرم‌افزار تلقی شوند، بدین معنی که آنها می‌توانند تحت کنترل نسخه(Version Control) قرار گرفته، توزیع شده و مورد استفاده مجدد قرار گیرند. درست همانطور که با دستورالعمل بهره‌برداری و تست‌های اتومات رفتار می‌شود.
- ✓ مدیریت کردن محیط‌های تست و تنظیم مجدد(ریست کردن) آنها برای اجرای تست بعدی، موضوعیست که به سادگی نمی‌توان از آن گذشت، و ارزش فوق‌العاده‌ای دارد.

طیفی گوناگون از ارائه دهندگان راهکار مجازی‌سازی سرویس(SV) در حال اتخاذ تکنیک‌های کانتینریزه کردن هستند تا بدین ترتیب راهکارهای خود را هر چه منعطف‌تر نمایند. برای مثال SV Platform Hoverfly که Open Source است(تولید شده توسط Spectolabs) به عنوان یک Container Docker روی DockerHub در دسترس است. موتور Virtualize SV از Parasoft نیز به عنوان یک Docker Container موجود است. بعلاوه این امکان نیز وجود دارد که یک ماشین مجازی روی Azure ایجاد کنیم و آن را با موتور مجازی‌سازی و بقیه ابزارها تجهیز نموده تا بدین ترتیب مستقیما عملیات استقرار محیط‌های تست مجازی آغاز گردد. باقی ارائه‌دهندگان راهکارها به دنبال Suite بوده و یا در حال حاضر در فرآیند انجام آن به سر می‌برند.

اتخاذ SV با حذف گلوگاه‌های محیط تست سنتی، می‌تواند سازمان‌ها را بوسیله دستیابی به توسعه و تست موثرتر در نرم‌افزار مهیا سازد. یکپارچه‌سازیِ SV در خط CD آن هم با استفاده از Containerization به تیم‌های توسعه کمک می‌کند تا بیشتر به سطح انعطاف مورد نیاز بازارهایی که در حال افزایش تقاضا هستند نزدیک شوند.

ابوالفضل خواجه دیزجی


نیستن
از ابتدای مسیر کیفیت با
شما خواهیم بود
START

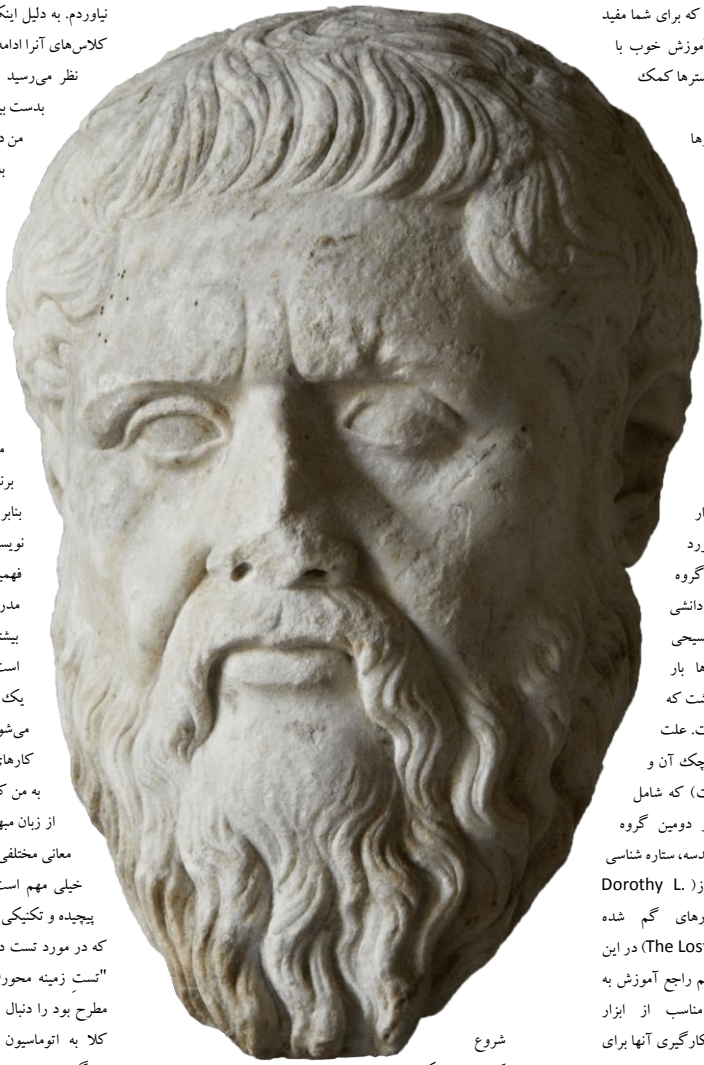
کشف دوباره ابزارهای یادگیری با علوم مقدماتی در تست نرم‌افزار

خلاصه: در حالیکه داشتن یک مدرک علوم کامپیوتر به طور قطع به حرفه شما در تست یا برنامه نویسی کمک می‌کند، اما داشتن یک پس‌زمینه در علوم مقدماتی نیز ممکن است به شما بیشتر از دانش‌تان کمک کند. علوم مقدماتی قادر هستند منطق، علم بیان و چگونگی دیدن یک تصویر بزرگ(وقتی شما در تلاش هستید تا ارتباطی در سیستم‌های پیچیده ایجاد کنید) را که برای شما مفید است به شما آموزش دهند. یک آموزش خوب با قابلیت‌های فراگیر به همه به ویژه به تسترها کمک می‌کند.

ما تسترها این روزها به سوی ابزارها جذب می‌شویم. به نظر می‌رسد اگر شما با ابزارهای جدید و معروف، در اطراف دوستان تستر خود ندرخشید، آنها شک می‌کنند که شما در حال از کار افتادن هستید. اما بهترین ابزارها چند پلاگین فانتزی نیستند که تماما از توئیتر الهام گرفته‌اند. مفیدترین این ابزارها که برای تست نرم‌افزار در اختیار شماست، ابزارهای یادگیری هستند.

هفت علم مقدماتی به عنوان ابزار یادگیری از زمان یونان باستان مورد احترام هستند. آنها به طور سنتی به دو گروه تقسیم می‌شوند: تریویم(Trivium دانشی اولیه در قرون وسطی در ممالک مسیحی بود. این دانش در سطحی صدها بار پیشرفته‌تر در جوامع اسلامی وجود داشت که امروزه بخشی از جامع المقدمات است. علت ذکر تریویم در اینجا دامنه بسیار کوچک آن و تناسب آن با دسته‌بندی مذکور است) که شامل گرامر، منطق و علم بیان است و دومین گروه کوئادریویم که شامل علم حساب، هندسه، ستاره شناسی و موسیقی است. دوروتی ال.سایرز(.Dorothy L Sayers) نویسنده کتاب "ابزارهای گم شده یادگیری"(The Lost Tools of Learning) در این کتاب می‌گوید: "در واقع تمام تریویم راجع آموزش به دانش‌آموز، در مورد استفاده مناسب از ابزار یادگیریست. البته قبل از اینکه وی بکارگیری آنها برای موضوعات را آغازکند". اگر تریویم در مورد استفاده مناسب از ابزارهای یادگیریست، کوئادریویم در مورد تمرین استفاده از این ابزارها و محیا کردن یک تحول برای مطالعات پیشرفته در ادبیات کهن است.

با این اوصاف چرا من در مورد چنین مطالب فاضلانه‌ای در چنین مقاله‌ای صحبت می‌کنم! در حالیکه به نظر می‌رسد باید تجربه‌ام را در زمینه تست نرم‌افزار منعکس نمایم؟ دلیل این امر این است که طی سال‌هایی که در مورد علوم مقدماتی و ادبیات کهن مطالعه می‌کردم، بدون اینکه متوجه شوم



شروع کردم به فکر کردن مانند یک تستر!

یادگیری برای دلیل، قبل از یادگیری برای تست

پس از تحصیل در زمینه مهندسی کامپیوتر، تصمیم گرفتم به سراغ علوم مقدماتی بروم. به صورت خلاصه انگیزه‌ای که در فکرم بود این بود که می‌توانستم کارهای بیشتری در مورد ریاضی، علوم پایه و برنامه‌نویسی کامپیوتر انجام دهم. حدود ۱۰ سال بعد، حقیقتاً در زمینه اشتغالِ مفید [بواسطه آن چیزی که آموخته بودم] خیلی چیزی به دست نیاوردم. به دلیل اینکه در زمینه برنامه‌نویسی خوب بودم کلاس‌های آنرا ادامه دادم و اگر چیز دیگری نیاز نبود، به نظر می‌رسید که بتوانم دستمزد بخور و نمیری بدست بیاورم.

من دوره‌های تست نرم‌افزار را هم بالاخره به پایان رساندم. البته برنامه‌نویسی قسمتی از شغل من بود، بنابراین یاد گرفتم انجام تست نرم‌افزار چقدر برای این کار مفید است. در بسیاری از موارد برای حل مشکلات مربوط به تلاش تست(کار صورت گرفته برای تست-Test Effort)، در قالب ابزارها و اسکریپت‌های موردی(Ad Hoc) از مهارت‌های برنامه‌نویسی خود استفاده می‌کردم، بنابراین ارزش کلاس‌های برنامه نویسی‌ام کاهش نیافت. در طول زمان فهمیدم که به عنوان یک تستر نرم‌افزار، مدرک علوم مقدماتی من به مراتب تاثیر بیشتری روی توانایی‌های من داشته است.

یک مثال که دو حیطه مطالعه را شامل می‌شود: دستور زبان و فلسفه است. بویژه کارهای ارسطو روی منطق صوری. هر دو به من کمک کرد که مفهوم ابهام یا استفاده از زبان مبهم را(به شکلی که خود این موضوع معانی مختلفی را باز می‌کند) درک نمایم.

خیلی مهم است که از ابهام مرتبط با موضوعات پیچیده و تکنیکی آگاه باشیم. برای نمونه، من بحثی را که در مورد تست در مقابل چک کردن در میان جامعهٔ "تستِ زمینه محور(Context-Driven Testing)" مطرح بود را دنبال می‌کردم. این مباحث در درجه اول کلا به اتوماسیون تست مرتبط بود که می‌توانست جایگزین تست دستی شود. مشکل اصلی در این بحث این بود که وقتی ما تست را چک کردن محض می‌نامیم، یعنی روی تست ابهام داریم. زبان مبهم معنی را گِل‌آلود و مبهم کرده و بحث را مشکل می‌سازد. این مساله افراد

را در مقابل همدیگر قرار می‌دهد و بدتر اینکه افراد را از بحث ناامید و خسته کرده و در نهایت علاقه آنها به موضوع از دست می‌رود. نتیجه‌ای که من آن را یک تراژدی می‌بینم.

ستاره‌شناسی به عنوان علم مقدماتی دیگر، مثال خوب دیگریست از اینکه چطور یک حوزه مطالعات کلاسیک می‌تواند روی تست تاثیرگذار باشد. در دوره مطالعاتم روی علوم مقدماتی چیزی خواندم که توسط بطلمیوس (ستاره‌شناس یونان باستان) نوشته شده بود. این مطالعه مرا به سمت کشف این موضوع که چطور آسمان می‌چرخد، سوق داد. در اینجا شما اول به این موضوع توجه می‌کنید که ستاره‌ها حرکت می‌کنند و علاوه بر این آنها مشترکاً نیز در حال چرخش به دور یک نقطه مرکزی هستند(حتی در دورترین نقطه‌ای که به نظر می‌رسد، همین نظم استوار است). این اجرام به طور کلاسیک به عنوان ستاره‌های ثابت اطلاق می‌شوند. سپس شما می‌فهمید که تعداد کمی از آنها نسبت به بقیه متفاوت حرکت می‌کنند که سیاره‌ها، قمرها و خورشید نامیده می‌شوند. سپس متوجه خواهید شد که حرکات سیاره‌ها لزوماً بر اساس نظم و قاعده‌ای مانند ستاره‌های ثابت نیست. برخی در واقع به صورت عقبگرد برای مدتی در مسیر خودشان حرکت می‌کنند. این یک مدل عالی برای این است که چطور بتوانیم سیستم‌های کوچک را از سیستم‌های بزرگ تشخیص دهیم. وقتی با موقعیتهای پیچیده برخورد می‌کنم، به طور غریزی خود را روی این مسیر فکری می اندازیم: اول سیستم بزرگتر را جستجو کن سپس کوچکترها را کشف رمز کن.

پروژه‌ای که من در حال حاضر روی آن کار می‌کنم، این فرآیند فکری را در کار نشان می‌دهد. این پروژه چندین سیستم یکپارچه، شامل OCR، یک سیستم یادگیرنده که شامل (AI- Artificial Intelligence)هوش مصنوعی و یک سیستم وارد کردن داده را در بر می‌گیرد. وقتی اول شروع کردم ترسیدم. چیزی را به خودم گفتم که همیشه وقتی یک پروژه جدید را شروع می‌کنم می‌گویم:"من هرگز این را نخواهم فهمید". در این زمان خود من در حال سقوط در الگوی تفکری‌ای بودم که برای اولین بار زمانی ایجاد شد که در حال مطالعه ستاره‌شناسی بودم.

برای غلبه بر این موضوع به تدریج هر سیستم را به طور جداگانه از کل سیستم مشخص کرده و آنرا از بقیه جدا کردم. درست مثل زمانیکه با سیاره‌ها و حرکاتشان در میان ستاره‌های ثابت کار داشتم. با این کار زمانیکه درک من واضح‌تر شد، می‌توانستم ببینم چگونه OCR و Data Entry(ورود داده) مستقلا با همان فیلد از صفحه Data Entry تعامل می‌کنند، و چگونه سیستم یادگیرنده هوش مصنوعی اطلاعات را در یک زمان و سیستم OCR اطلاعات را در زمان دیگر تامین می‌کند و چرا.

مطالعات خود را ادامه دهید

برای جمع‌بندی، نمی‌گویم که شما باید به مطالعه علوم مقدماتی بپردازید، تا این علوم و اصولش را تمرین نمایید. اما به شما می‌گویم که اگر این کار را انجام دهید خیلی خوب و مفید است، حتی اگر این کار به مقدار کم و به هر اندازه‌ای باشد که استطاعت آن را دارید. به دنبال افرادی باشید که در زمینه این ایده با آنها به بحث و تبادل نظر بپردازید. به افراد فنی آویزان نشوید. افرادی از سایر رشته‌ها پیدا کنید. شما سورپرایز خواهید شد وقتی ببینید افق دیدتان چطور گسترده خواهد شد.

مهمترین چیزی که شما را در فکر کردن مشتاق نگه می‌دارد، جستجوی هر چیز و هر فردیست که به شما کمک می‌کند تا کارتان را به بهترین شیوه انجام دهید. شما می‌بینید که نه تنها به عنوان یک تستر نرم‌افزارِ بهتر ساخته شده‌اید، بلکه به عنوان یک آدم بهتر نیز شکل گرفته‌اید.

امیرعلی یاسینی

www.TISTeN.ir

با کلیک روی این لوگو آخرین شماره از

تیستن را دریافت کنید و به **قطب تلگرامیِ**

مهندسان تست نرم‌افزار ایران ملحق شوید

توسعه نرم افزار در DevOps را با یک بازی آموزش دهید

خلاصه: ایده اصلی DevOps عبارتند از نقش‌های متفاوتی که با همدیگر کار می‌کنند تا یک سیستم نرم‌افزاری پایدار ایجاد کنند. افراد می‌توانند در مورد این موضوع بشنوند یا در موردش بخوانند یا حتی آن را مشاهده کنند. ولی اغلب بهترین راه درک DevOps برای یک تیم که جدیداً با آن روبرو شده است، فقط اجرای آن است. زمانیکه شما تصمیم به آغاز می‌گیرید، آغاز شما می‌تواند منجر به شکست(Failure) روی یک سیستم واقعی شود، به همین دلیل شبیه‌سازی، یک ایدهٔ خوب است. بازی همیشه یکی از بهترین راه‌های آموزش است. بنابراین سعی کنید برای معرفی DevOps به تیمتان نیز از یک بازی کمک بگیرید.

وقتی با دیگران در مورد DevOps صحبت می‌کنم نظرات بسیاری را می‌شنوم که اغلب در مورد اتوماتیک کردن همه چیز و تعداد زیادی استقرار زیاد می‌باشد.

مشکل این است که اگر شما به تعداد زیادی استقرار داشته باشید، ولی همان شیوه‌های باگ‌دار گذشته را حفظ کنید، تیم شما همواره کدهای باگ‌دار را مستقر خواهد کرد. باگ‌ها سریعتر از تغییرات جدید تشدید می‌شوند. هر چند که اصلاحات نیز منجر به معرفی باگ‌های جدید می‌شوند. بالاخره یک نفر شکست را اعلام خواهد کرد و تیم به ورطه "سختگیری فرآیندی" یا "دیسیپلین" یا مورد دیگری خواهد افتاد. DevOps شهرت کارگاهی بدی دارد و [به همین دلیل] معرفی مفاهیم، نیازمند نامی جدید است، که بعدا ارائه می‌شود.

این نام نباید شبیه نام قبلی باشد، و الزامی هم به این شباهت وجود ندارد.

مشکل اصلی که من می‌بینم این است که افراد با ارزش‌ها شروع می‌کنند. یک ارزش کلاسیک برای برنامه‌نویسان این است که "کد خوب است". اگر شما با این ارزش شروع کنید، بنابراین شبکه‌های تعریف شدهٔ نرم‌فزاری، زیرساخت‌های تعریف شدهٔ نرم‌فزاری و اتوماسیونِ ابزاریِ تست، همگی ذاتاً خوب هستند، چرا که همه آنها کد هستند.

DevOps یک عنصر قوی از اعمال مفاهیم خوب برنامه‌نویسی به دیسیپلین‌های دیگر مانند بهره‌برداری، اجرا، ایجاد سرورهای جدید و ابزار و زیرساخت تست است. ایده اصلی DevOps در مورد نقش‌های مختلفیست که در حال کار با یکدیگر هستند تا یک سیستم تحویل(و نرم افزاری) که پایداری بیشتری دارد ایجاد کنند.

من این موضوع را به دیگران گفته بودم. برای آنها مقالاتی فرستاده بودم تا در موردش بخوانند. اما بهترین روشی که برای توضیح DevOps دیدم، اجرای آن یا انجام یک شبیه‌سازی بود. البته از نظر من شبیه‌سازی روش بهتری به جای اجرای آن است.

چند وقت پیش دیدم یک مهندس کامپیوتر به نام نوح ساسمَن(Noah Sussman) یک بازی برای انجام این کار توسعه داده است، به نام Abelian Sandpile Game.

ساخت نرم افزار در یک Abelian Sandpile

برای یک شخص کنجکاو، Abelian Sandpile یک شبیه‌سازی ریاضی از یک توده ماسه است. هرچه شما به توده، ماسه اضافه کنید بلندتر می‌شود، تا زمانیکه توده نتواند وزن را تحمل کند و سپس پایه فرو می‌ریزد. من فکر می‌کنم این یک نمونه مناسب برای شبیه سازی چیزیست که در پروژه‌های نرم‌افزاری رخ می‌دهد.

برای بازی، شما به یک ورق کاغذ با یک شبکه ۴ در ۴ که روی آن پرینت شده و ایضا تعدادی سکه، مهره بازی یا صفحه‌های کوچک دیگر که بتوانند روی هم به صورت توده جمع شوند، نیاز دارید. تصور کنید که این شبکه مدل نرم‌افزاری شماست که شما روی آن سه "سایت استقرار(Deploy Site)" قرار می‌دهید. وظیفه

شما به عنوان یک تیم، استقرار ۴۰ درخواست تغییر روی سایت‌های استقرار است.

البته یک Catch وجود دارد. نرم‌افزار تعدادی باگ خواهد داشت. وقتی یک خانه(یکی از سلول‌های شبکه ترسیم شده)، ۵ تغییر(که به وسیله سکه‌ها نمایش داده می‌شود) در خود داشته باشد، خرابی‌های نرم‌افزار تبدیل به یک سکه در خانه اصلی، یکی در خانه بالایی، یکی در خانه سمت راست، یکی در چپ و یکی در خانه پایین خواهد شد. شما می‌توانید همچنان استقرار ادامه دهید تا نهایتاً به یک خرابی دیگر برخورد کنید. خرابی، خرابی به همراه خواهد داشت. هنگامیکه نرم‌افزار شبکه(شبکه ۴ در ۴) را خراب کند، شما یک قطعی بزرگ ایجاد کرده‌اید.

این شبیه‌سازی، با آنچه که ما همیشه در نرم‌افزار می‌بینیم فرق زیادی ندارد. یعنی دقیقا همان اتفاقاتی که در زمان پیکربندی سیستم‌های عامل، ارتقاء پایگاه داده و وب‌سرور به اضافه همه کدهایی که می‌نویسیم، رخ می‌دهد. اغلب ما در جاییکه چند باگ کوچک به عنوان باعث و بانی یک مشکل بزرگتر با هم ترکیب می‌شوند، اثر آبشاری(تخریب ترتیبی) را رویت می‌کنیم.

یک درس در DevOps

در راند اول، بازیکنان با هدف کوچک کردن تعداد نارسایی‌های(Failure) جدی، نرم‌افزار را در شبکه مستقر می‌کنند(یعنی در تلاش هستند که با حداقلِ تعدادِ سکه‌ها که راهیِ خارجِ شبکه می‌شوند، مواجه باشند. تسهیل‌کننده بدون اعلام اینکه قصد زمانبندی دارد، راند را زمانبندی می‌کند. در راند دوم، تسهیل‌کننده می‌گوید تیم مکانیزمی برای تشخیص مشکل‌ها دارد و فورا Rollback(عقبگرد) می‌کند، بنابراین بازیکنان نباید نگران خروج سکه‌ها از شبکه باشند. در عوض آن، تیم فقط نیازمند تلاش برای بدست آوردن همه ۴۰ تغییر مستقر شده است.

اولین راند عموماً ۱۰ تا ۱۲ دقیقه برای بازی زمان می‌برد. اگر یک گروه بزرگتر که نیاز به اجماع دارند وارد بازی شوند، این زمان احتمالاً طولانی‌تر خواهد شد، و اگر یک بازیکن وجود داشته باشد، بازی سریعتر پیش خواهد رفت. راند دوم ممکن است ۴ دقیقه طول بشکد.

یک تفاوت بسیار بزرگ در توان عملیاتی(Throughput) وجود دارد که اشاره دارد به تفاوت بین دو رویکرد تست سنتی یعنی"گرفتن مشکلات قبل از انتشار(Release)" و رویکرد مدرن‌ترِ متمرکز بر DevOps یعنی "حرکت سریع و پایدار نگهداشتن سیستم". تیم‌هایی که بازی Abelian Sandpile را انجام می‌دهند تفاوت در عملکرد را تجربه می‌کنند.

بزرگترین مشکلی که من با رویکردهای DevOps و تحویل مداوم در زندگی واقعی دیده‌ام این است که این رویکردها قطعه پایداری را گم می‌کنند. چیزی که در چرخه مهندسی به آن حالت انعطاف‌پذیری گفته می‌شود. اگر به جای تمرکز بر کاهش زمان بین Failureها، بر ایجاد یک سیستم منعطف(چیزی که هزینه Failure را به حداقل می‌رساند، کاهش زمانِ ریکاوریست) تمرکز کنیم، می‌توانیم در زمان کمتر و با بحث و جدل کمتر ، کار بیشتری انجام دهیم.

لطفاً به ورژن بتای پیشرفته این بازی توجه کنید. چیزی که از آن مطمئن نیستم تعداد ایده آل سکه‌ها برای بازی با آنهاست. با تعداد خیلی کم، بازی خیلی ساده می‌شود، البته تعداد بسیار بالای آنها نیز باعث می‌شود که بازی به یک رژه مرگ خسته‌کننده تبدیل شود که شکست در آن اجتناب ناپذیر خواهد بود.

اگر از نظر شما این بازی ارزشمند بود لطفا آنرا در میان دیگران نیز نشر دهید.

بهاره اصغری

برای ارسال مقاله به تیستن و همکاری با این مجموعه با آدرس زیر در تلگرام تماس بگیرید

@adizaji

نشستِ بهره‌برداری،

شمشیر دو لبه

خلاصه:
تست کردن در زمان بهره‌برداری(Production) فرصت‌های واقع‌گرایانه‌تری به تست می‌دهد. این کار موجب افزایش شفافیت بین تیم اصلیِ محصول و کاربران شده، و از ایده توسعه مداوم(Continuous Development) از طریق تست مداوم(Continuous Test) حمایت می‌کند. این یک تکنیک خوب برای احاطه به فرآیند تست شماست. هر چند که نباید بدون آمادگی نباید وارد آن شد. اکنون در اینجا مزایا و معایب این کار را می‌آموزیم.

در ادوار گذشته، یعنی زمانیکه بیشتر شرکت‌ها از الگوی آبشاری(Waterfall) برای توسعه استفاده می‌کردند، گوگل در این صنعت یک جوک محسوب می‌شد، چرا که محصولاتش همیشه دارای حالت بتا بود. به ضرس قاطع می‌توان گفت، گوگل یک پیشگام در ساخت نمونهٔ تستی برای بهره‌برداری است. به طور سنتی، قبل از اینکه Build به مرحله عملیاتی برسد، یک تستر در محیط تست یا عملیاتی مسئول تست همه سناریوها اعم از تعریف شده و فی‌البداهه است. اما امروزه، این قضیه در چند جبهه در حال تغییر است.

به عنوان نمونه، تستر در تست دیگر تنها نیست. توسعه‌دهندگان، طراحان، مهندسین ساخت(Build)، سایر ذینفعان و کاربران نهایی، در خارج و داخل تیم تولید، اپلیکیشن را برای ارائه بازخورد تحت تست قرار می‌دهند.

امروزه محیط تست(Test Environment)، محصول تحت تست، فناوری‌های زیربنایی، ترکیبات تست(دستگاه‌ها، پلتفرم‌ها، مرورگرها، و غیره)، همگی بسیار پیچیده شده‌اند. طرز تفکر در چنین خدماتی بسیار سطح بالاست، و به همین دلیل محلی برای تست محصول در نظر می‌گیرند. محیط Cloud مقیاس‌پذیری و سهولت مورد نیاز در اینترفیس‌های پیچیدهٔ تست را ارائه می‌کند، و محیط بهره‌برداری فرصت‌های تست منحصر به فردی ارائه می‌دهد که امکان پیاده‌سازی کامل آنها قبل از انتشار(Release) وجود ندارد.

تیم تست با محدودیت‌هایی دست به گریبان است از جمله: زمان، هزینه، و در دسترس بودن تسترها و بسیاری مشکلات دیگر که از قبل نیز وجود داشته است.

با وجود تمام این عوامل در بازی، تست بهره‌برداری اجتناب‌ناپذیر است. با این حال، به جای نگاه کردن به تست بهره‌برداری به عنو ان یک گزینه که روی تیم‌ها فشار وارد می‌کند، اگر از نزدیک نگاهی به مزایای واقعیِ موجود در آن بیندازیم، کمک بزرگی در افزایش کیفیت محصول به ما خواهد کرد.

با این اوصاف، تست بهره‌برداری واقعا چه چیزی را بیان می‌کند؟ برخی از راه‌هایی که می‌توانید آنرا انجام دهید چیست و چگونه این موضوع به شما کمک می‌کند؟

تست بهره‌برداری به چه معنیست

تست بهره‌برداری، تمرینیست که در آن اعتبارسنجی(Validating) و تصدیق(Verifying) یک اپلیکیشن، در محیط زنده، بعد از انتشار(Release)، بوسیله تستر یا کاربر نهایی(End User) اخذ می‌شود. البته افراد دیگری نیز می‌توانند در این موضوع دخیل باشند، مانند واسطه‌ها، تیم‌های بازاریابی، یا تحلیلگرانی که بازخوردها را با تیم محصول به اشتراک می‌گذارند.

تست بهره‌برداری فرصت‌های واقعی بیشتری برای تست ایجاد می‌کند، که منجر به افزایش شفافیت بین تیم اصلی محصول و کاربران شده، و ایده توسعه مداوم از طریق تست مداوم را حمایت می‌کند. در دنیای Mobile-First امروز، تست بهره‌برداری یک تکنیک اصلیست که باید در فرآیند تست شما جای گیرد.

البته یک بار معنایی منفی در رابطه با مشکلات گزارش شده در این قالب وجود دارد: این موضوع اساسا به این معناست که تست انجام شده روی محصول توسط تستر به اندازه کافی جامع و مؤثر نبوده است. مشکلی که با این قالب گزارش می‌شود فورا با اولویت بالا و در نزدیک‌ترین زمان ممکن مورد رسیدگی قرار می‌گیرد. در حالیکه آن دست از مشکلاتی که در بهره‌برداری نشان داده می‌شوند هنوز هم می‌توانند نماینده یک لکه سیاه در نشان دادن تلاش‌های تستر باشند، اما باید گفت که تست بهره‌برداری، امروزه دامنه‌ای گسترده دارد. این موضع نه تنها به معنی اقدامات واکنشی در پاسخ به مشکلات گزارش شده از سوی کاربر است، بلکه تلاش‌های برنامه‌ریزی شده کنشی‌ای است قبل از اینکه محصول به صورت رسمی اجرایی شود، بروز می‌کند.

تکنیک‌هایی برای تست بهره‌برداری

تکنیک‌های اصلی برای تست بهره‌برداری عبارتند از: نظارت فعال(Active Monitoring) و منفعل(Passive Monitoring) که شامل داده‌های واقعی و تراکنش‌های مصنوعی

هستند، آزمایش یا Experimentation (کنترل شده و کنترل نشده) با کاربران واقعی، و تست تنش(Stress) جهت مانیتور کردن پاسخ سیستم. در حالی که ممکن است چنین چیزی ساده به نظر برسد، یک نفر باید در تمام این وظایف به شدت حساس باشد، چون در این برهه داده‌های زندهٔ کاربران اغلب درگیر شده و باید مورد حفاظت قرار گیرد(حساسیت فقط روی از دست رفتن داده‌ها یا Data Loss نیست، بلکه حفظ حریم خصوصی یا Privacy و امنیت یا Security را نیز شامل می‌شود). بعلاوه، حجم تراکنش‌ها در محیط بهره‌برداری آنقدر بالاست که هر گونه تلاشی در اینجا باید به میزان کافی تحت نظارت و پیگیری باشد. تکنیک‌هایی مانند تله‌متری یا تشخیص در حدود استفاده از نرم‌افزار، خوانایی نتایج تست بهره‌برداری را به ارمغان می‌آورد. در حالی که نظارت فعال(Active Monitoring) بیشتر روی خروجی‌های تولید شده توسط کاربر متمرکز می‌شود، نظارت منفعل(Passive Monitoring) با تلاش تست(Test Effort) صورت گرفته از سوی تیم‌های تست با داده‌های مصنوعی می‌باشد. به عنوان نمونه، نظارت فعال خواهد بود اگر شما یک تیم از کاربران برای تست بتا استخدام کنید تا به شما بازخورد ارائه دهند، در حالی که نظارت در صورتی منفعل است که تیم تست نظارت را یا خود و یا از طریق سایر تیم‌های محصول آغاز کند. این روش همچنین می‌تواند شامل اجرای مجموعه‌ای از تست‌های اتومات سلامت(Sanity Test) به صورت مداوم توسط تیم بهره‌برداری و پشتیبانی برای سالم نگاه داشتن اپلیکیشن در محیط زنده و عملیاتی باشد.

تست با یک قالب کنترل شده، اغلب برای ارزیابی بازخورد کاربر در سناریوهای خاص شامل تکنیک‌هایی همچون A/B Testing می‌شود، در حالیکه تست کنترل نشده با تست بتا و تست جمع‌سپاری‌شده(Crowdsourced Testing) در ارتباط است. تست تنش(Stress Test) در محیط زنده و عملیاتی باید به دقت تحت نظارت قرار گیرد. به خصوص آنهایی که در فصول اوج بار(Load) برای اپلیکیشن انجام می‌شوند. به عنوان مثال، فصل اوج خرید اپلیکیشن(مثلا فصول تعطیل یا فروش ویژه) می‌تواند باعث افزایش چشمگیر بار روی اپلیکیشن شود. به جای اینکه منتظر شنیدن این مشکلات در خلال چنین فصولی باشیم، تسترها باید به صورت فعالانه در چنین زمان‌هایی Loadهای حاصله را مانیتور کرده و یک تیم آماده برای مشخص کردن مشکلات داشته باشند.

در ارتباط با مسائل مطروحه در بالا، امروزه محصولات در رسانه‌های اجتماعی نیز حضور دارند. اپلیکیشن‌های فروشگاه‌های خرده‌فروشی(Retail Stores) و حتی دسک‌تاپ‌ها، صفحات اختصاصی خود را در فیس بوک، توییتر و لینکدین دارند. تستر باید به صورت فعالانه مباحث موجود روی این فروم‌ها را دنبال کند تا چیزهایی که کاربران در مورد آن صحبت می‌کنند(مانند کاربردپذیری یا Usability، Performance، کلیات Functionality و UI برخی از مطالب مهم برای دریافت بازخورد است) را متوجه شود. مطالعات میدانی روی کاربران، بازدیدها برای Enterprise Deploymentها(در مورد Enterprise Aplicationها) و غرفه‌ها در رویدادها(مانند نمایشگاه‌ها و کنفرانس‌ها) همگی مکان‌هایی بزرگ

برای حصول بازخوردهای زنده هستند، که در نهایت به سمت تست بهره‌برداری هدایت می‌شوند.

اخطار

در حالی که تست بهره‌برداری پتانسیل و دامنه بزرگی دارد، به معنی دعوتنامه‌ای برای تسترها به منظور تاخیر در مسئولیت‌هایشان تا زمان انتشار (Release) نیست.

علاوه بر این ممکن است اتخاذ تست بهره‌برداری در سطح سازمانی برای رسیدن سریعتر به بازار و البته صرف هزینه پایینتر وسوسه انگیز باشد، اما این یک استراتژی تکمیلیست و نباید مطلقا به استراتژی اصلی کیفیت تبدیل شود.

تست بهره‌برداری شمشیر دولبه‌ایست که اگر به درستی استفاده شود، بسیار موثر خواهد بود. اما اگر برای این کار نگرش درست یا آمادگی علمی نداشته باشید استفاده از آن نگران کننده است. باید این موضوع را به درستی برای خود تعریف کرده و ارزش آنرا به درستی تخمین بزنید. تست بهره‌برداری پیشنهاد زیادی برای سال ۲۰۱۷ دارد. بنابراین روی آن سرمایه‌گذاری کنید.

ابوالفضل خواجه دیزجی

کیفیت،
ثروت و اعتبار است

محتوای تخصصی و کاربردی سایت تیستن در زمینه‌های:

- آخرین اخبار ایران و جهان در زمینه تست نرم‌افزار
- مقالات کاربردی تست نرم‌افزار
- آموزش در حوزه تست نرم‌افزار

مشاوره مهندسی تست در زمینه‌های:

- فرآیند توسعه نرم‌افزار و بسترسازی تست نرم‌افزار
- طراحی و آماده‌سازی فرآیند تست و جزییات آن
- آماده‌سازی و اجرای Test Plan
- استفاده از ابزارهای اتوماسیون تست
- راه‌اندازی Test Lab

آموزش و ساخت تیم تست نرم‌افزار:

- بررسی استعدادها، نیازمندی‌ها و منابع شرکت جهت آموزش متناسب
- آموزش مباحث مقدماتی و پیشرفته بر اساس بررسی‌های صورت گرفته
- ایجاد و به روزرسانی فرآیند توسعه و ایجاد فرآیندهای مرتبط با تست
- بررسی امکان مکانیزاسیون تست در حوزه‌های مورد نیاز
- آماده‌سازی و آموزش تیم برای تست مکانیزه

اجرای پروژه تست با نیروی مقیم در سازمان:

- در اینجا تمام موارد مطروحه در «مشاوره» و «آموزش و ساخت تیم تست نرم‌افزار» ارائه می‌شود، با این تفاوت که در اینجا کار به صورت یک پروژه تصور می‌شود، و سرپرستی و همراهی تیم مقیم در سازمان تا زمان خودکفایی تیم یا نیاز سازمان ادامه خواهد یافت

کیفیت، را با تیستن تجربه کنید

www.TISTeN.ir
Info@TISTeN.ir

ویرایش چهارم

آندریاس اِسپیلنر، تیلو لینز، هانس شِیفر

مبانی آزمون نرم‌افزار

(مبتنی بر ISTQB)

راهنمای مطالعه برای امتحان آزمونگر مجاز

- مطابق با **ISTQB**
- در مقطع **Foundation**

ترجمه ابوالفضل خواجه دیزجی

اولین مرجع رسمی ISTQB فارسی در ایران جهت اخذ مدرک CTFL و مورد تائید ISTQB