تست برنامه‌هایPHP، کیفیت کد منبع و روند توسعه آنها

(امیر یوسفی، amiryousefi.it@gmail.com)

چکیده: تست یکی از جنبه‌های ضروری توسعه در هر زبان برنامه‌نویسی است. بدون تست کد منبع[1] امکان بررسی صحت عملکرد آن همانطوری که انتظار می‌رود عملا وجود ندارد. تست‌های دستی می‌توانند فقط به صورت نامنظم و به صورت محدود اجرا شوند. برای تست منظم و در عمق کدهای منبع، نیاز به نوشتن تست‌های خودکار هست تا به صورت متناوب اجرا شوند. آزمون‌های واحد[2] معمولا یک کار اسرارآمیز و زمان‌بر به نظر می‌رسد و ممکن است همین‌طور هم باشد. اما صرف زمان برای نوشتن تست‌ها باعث خواهد شد تا کیفیت کدهای منبع افزایش پیدا کند و در مجموع باگ‌های کمتری داشته باشد، بسیاری از خطاها زودتر تشخیص داده شوند، یک فرایند تست دائمی برای جلوگیری از تغییر رفتار کدهای قبلی توسط تغییرات جدید ایجاد شود، و اعتماد کافی برای تکیه به کد به وجود بیاید.

در این مقاله تلاش می‌شود ضرورت تست و اطیمنان از کیفیت کد منبع برنامه‌های PHP بررسی شده و برخی از ابزارهای مورد نیاز برای تست و اطمینان از کیفیت برنامه‌های PHP معرفی شود. همچنین نقش آزمون‌های واحد در «توسعه مبتنی بر تست[3]» بررسی و مزایای توسعه برنامه‌های تست‌پذیر و اطمینان از کیفیت کد منبع ذکر می‌شود.

کلمات کلیدی: تست‌پذیری، کیفیت سورس کد، آزمون واحد، PHP، T.D.D

[1] Source code
[2] Unit Tests
[3] Test Driven Development(T.D.D)

مقدمه. تست یکی از جنبه‌های ضروری توسعه در هر زبان برنامه‌نویسی است. بدون تست کد منبع امکان بررسی صحت عملکرد آن همانطوری که انتظار می‌رود عملا وجود ندارد. تست‌های دستی می‌توانند فقط به صورت نامنظم و به صورت محدود اجرا شوند. برای تست منظم و در عمق کدهای منبع، نیاز به نوشتن تست‌های خودکار هست تا به صورت متناوب اجرا شوند. برای نوشتن همچون تست‌هایی در PHP معمولا از یک چارچوب آزمون واحد استفاده می‌شود. چارچوبی که به کد منبع هر اپلیکیشن یا کتابخانه‌ای اجازه می‌دهد تا به صورت واحدهای عملکرد ایزوله مثل یک کلاس واحد یا یک متد تست شوند. با رایج شدن آزمون واحد، نیاز به پوشش آزمون واحد در کد منبع به عنوان یک روش استاندارد در کتابخانه‌ها و فریم‌ورک‌هایی مثل Zend Framework،Swiftmailer و Symfony در زبان PHP تبدیل شده است.

آزمون‌های واحد معمولا یک کار اسرارآمیز و زمان‌بر به نظر می‌رسد و ممکن است همین‌طور هم باشد. اما صرف زمان برای نوشتن تست‌ها باعث خواهد شد تا کیفیت کدهای منبع افزایش پیدا کند و در مجموع باگ‌های کمتری داشته باشد، بسیاری از خطاها زودتر تشخیص داده شوند، یک فرایند تست دائمی برای جلوگیری از تغییر رفتار کدهای قبلی توسط تغییرات جدید ایجاد شود، و اعتماد کافی برای تکیه به کد به وجود بیاید. مزایای دیگر آزمون واحد را در ادامه با دقت بیشتری بررسی خواهیم کرد.

۱. مغالطه‌های تست

آزمون واحد، و در واقع همه انواع تست، گرفتار چهار بهانه معمول زیر هستند که توسعه‌دهنده‌ها را از پذیرش آن منع می‌کنند:

۱. زمان‌بر هستند.
۲. کدهای پیچیده قابل تست نیستند.
۳. تا زمانی که کار می‌کند، نیازی به نوشتن تست نیست.
۴. تست خسته‌کننده است.

این‌ها مغالطه‌های تست هستند، بهانه‌هایی که به نظر معقول می‌آیند ولی در واقع به صورت زیرکانه‌ای با اطلاعات غلط نتیجه‌گیری شده‌اند.

تست زمان می‌گیرد. سوال این است که چرا این زمان باید ارزشمند تلقی شود و جواب این است که تست زمانی که در آینده برای اصلاح،نگه‌داری، بهبود و رفع باگ‌های کشف‌نشده صرف خواهید کرد را کاهش می‌دهد. و همه ما می‌دانیم اگر تست را به صورت جامع انجام ندهیم هزاران باگ کشف‌نشده وجود خواهد داشت.

همچنان که کد می‌نویسید، می‌توانید مستقیما از آزمون‌های واحد برای تست متدها/کلاس‌های ایزوله یا گروهی از کلاس‌های کاربردی استفاده کنید. یک مشکل این است که باگ‌ها را غالبا پیدا می‌کنید و به سرعت اصلاح می‌کنید، طوری که به سختی متوجه زمانی که گرفته می‌شوید. وقتی شما تغییری ایجاد می‌کنید، و تستی شکست می‌خورد، می‌توانید به همان سرعت مشکل یکپارچگی را برطرف کنید و زمان بیشتری ذخیره کنید که باز به سختی متوجه آن می‌شوید. مزایای تست ممکن است به اندازه‌ای خوب مستتر شده باشد که کاملا

فکر کنید وجود ندارند و فقط زمانی که صرف نوشتن کدهای تست کرده‌اید را ببینید، نه باگ‌هایی که در عرض ده ثانیه حل شده‌اند که ممکن بود دقیقه‌ها و یا حتی ساعت‌های متمادی در طول چند ماه صرف کشف آنها کنید.

در مورد کدهای پیچیده باید بگوییم وجود آنها غیر قابل انکار است اما کدهای پیچیده و کدهای پیچیده‌ای که از قسمت‌های کوچک‌تر کاربردی ساخته شده‌اند دو مقوله کاملا متفاوت هستند. در برنامه‌نویسی شی گرا یک آبجکتیو[۴] ساده معمولا قابل تست است. اگر کدهای شما به راحتی قابل تست نیست این آزمون واحد نیست که شکست خورده است بلکه این شمایید که در نوشتن کدهای کاربردی که هم منعطف هستند و هم بخش‌های مختلف آن به خوبی از هم جدا شده‌اند شکست خورده‌اید. اگر همزمان با نوشتن کد تست را هم انجام بدهید ناخواسته مجبور خواهید شد تا کلاس‌ها را ازهم جدا کنید. این حقیقت که کدها برای تست بسیار پیچیده به نظر می‌رسند معمولا حاصل انتظار طولانی برای شروع تست است.

در مورد سوم، توجه کنید کدی که کار می‌کند با کد تست‌شده‌ای که کار می‌کند دو موضوع کاملا متفاوت هستند. تست‌ها یک تور حفاظتی ارائه می‌کنند تا با شناسایی تقریبا بلافاصله مشکلاتِ یکپارچگی امکان تغییر، بهبود و بهینه‌سازی و افزودن ویژگی‌های جدید به راحتی فراهم شود. علاوه بر این، کارآیی برنامه‌نویس‌های جدید تیم که با کدها ناآشنا هستند را نیز از طریق مشاهده بلافاصله اشتباهاتشان در یک چرخه کوتاه‌تر بازخورد و کسب تجربه از طریق آن، بهبود می‌بخشد.

و در آخر، تست همیشه هم خسته‌کننده نیست. تست‌ها معمولا به این دلیل خسته‌کننده هستند که نوشتن تست‌ها را به آخر پروسه توسعه موکول می‌کنیم. اگر به صورت پیوسته از کد به تست و برعکس سویچ کنید بدون خمیازه کشیدن کار بیشتری به مرحله انجام می‌رسانید. بنابراین درمان خسته‌کننده بودن تست به سادگی تعویض پیوسته بین کد و تست و نوشتن تست در طول زمان است. در دام نوشتن تست و فقط نوشتن تست برای چند روز متوالی نیفتید.

۲. آزمون واحد در PHP

دو انتخاب معمول برای آزمون واحد در PHP یکی SimpleTest و دیگری PHPUnit است. همچنین می‌توانید از phpt که کمی قدیمی‌تر است استفاده کنید. همه این سه روش امکان نوشتن آزمون‌های واحد را فراهم می‌کند، همچنین کتابخانه‌های دو چارچوب اولی امکانات بیشتری نیز در اختیار شما قرار می‌دهد.

هر تست از یک کلاس با چند متد عمومی تشکیل شده است. همچنین از آنجایی که هر تست باید ایزوله باشد و هیچ اطلاعاتی را با تست‌های دیگر مشترک نشود این کلاس از فیکسچرهایی[۵] بهره می‌برد که کار ساختن یک وهله[۶] از شی مورد نظر برای ورود به تست و نابود کردن آن در انتهای تست را به عهده دارند. فیکسچرها باعث می‌شوند تا مطمئن شویم برای هر تست یک نسخه جدید از شی را برای آزمون در اختیار داریم.

---

[4] Objective
[5] Fixtures
[6] Instance

هر متد عمومی کلاس تست یک آزمون است. هر متد ارجاعاتی به تابع اظهار[7] دارد. هر اظهار[8] یک مقدار مورد انتظار را می‌گیرد و آن را با یک مقدار واقعی مقایسه می‌کند. دانستن چیزی که باید در هر تست اظهار شود ۹۰ درصد راه است. ۱۰ درصد بقیه اطمینان از ایزوله بودن تست از سایر تست‌ها است.

۳. اجرای تست‌ها

اجرای تست‌ها معمولا به سادگی اجرای یک دستور در خط فرمان و یا بارگذاری مجدد مرورگر است. البته چارچوب‌های آزمون واحد امکان دسته‌بندی تست‌ها و اجرای مجموعه‌ای از تست‌ها را به صورت یکجا می‌دهد تا دردسر اجرای تک به تک تست‌ها را نداشته باشید.

۴. آزمون واحد و توسعه مبتنی بر تست(TDD)

یک روال معمول در برنامه‌نویسی این است که کلاس‌ها را بنویسیم و بعد تست‌ها را انجام بدهیم. اما اجازه بدهید از نگاه دیگری به مساله تست نگاه کنیم.

یکی از کاربردهای معمول آزمون واحد تمرین توسعه مبتنی بر تست است. TDD یک جزء ضروری آزمون واحد نیست، بلکه روشی است که از آزمون واحد به عنوان استانداردی برای نوشتن مثال‌های قابل اجرا استفاده می‌کند. ایده پشت TDD این است که به وسیله شرح رفتار یک کلاس در کد قابل اجرا(به عنوان مثال تست) قبل از کدنویسی آن، ابزاری برای جهت‌دهی به چگونگی طراحی کلاس فراهم می‌شود.

در نوشتن یک تست برای یک کلاس، مجبور به طراحی API عمومی مربوط به آن هستیم. از آنجایی که بیشتر از چگونگی عملکرد کلاس به نتیجه مورد انتظار از کلاس علاقه داریم نیازی به نوشتن تست برای متدهای خصوصی[9] کلاس نیست. در بسیاری از موارد، نتیجه ثابت می‌ماند اما ممکن است کد در طول زمان به خاطر بهبود و بهینه‌سازی، عملکردهای جدید PHP، یا نیازمندی‌های سیستمی دستخوش تغییر شود. بنابراین تست کردن همه این اجزا نتیجه‌ای جز تغییر پیوسته کدهای تست نخواهد داشت.

تعریف API عمومی کلاس‌ها به منزله شروع زودهنگام طراحی کلاس‌ها است. در واقع هنوز تست‌ها را بر روی خود کلاس‌ها انجام نمی‌دهیم(آنها هنوز نوشته نشده‌اند)، اما انتظاری که از نحوه عملکرد کلاس داریم را مشخص می‌کنیم و سپس کدها را می‌نویسیم تا عملکرد مشخص‌شده را اجرا کنند.

زمانی که واقعا شروع به کدنویسی کردیم، بسیاری از کارهای طراحی را انجام داده‌ایم و علاوه بر آن مجموعه‌ای از تست‌ها به طور پیوسته صحت کدهایی که می‌نویسیم را بررسی می‌کنند.

با وجود اینکه TDD یک متدولوژی طراحی است اما همچنین مجموعه‌ای از تست‌ها را نیز فراهم می‌کند. حال می‌توانیم شروع به کدنویسی کنیم تا تست‌های نوشته‌شده پاس شوند. همچنین ممکن است در آینده تست‌های بیشتری نیز اضافه کنیم.

---

[7] Assert
[8] Assertion
[9] Private

۵. مزایای آزمون واحد

۵,۱. آزمون واحد سطح باگ‌های کد نهایی[10] را کاهش می‌دهد.

با اجرای متناوب مجموعه در حال رشد تست‌های نوشته شده باگ‌ها شناسایی خواهند شد. با شناسایی زودهنگام باگ‌ها، معمولا می‌توانید در مدت زمان کوتاهی آنها را برطرف کنید زیرا کدها هنوز در ذهنتان تازه اند. علاوه بر این، برای باگ‌های جدیدی که تست‌ها بلافاصله شناسایی نکرده‌اند، می‌توانید تست‌های جدیدی بنویسید تا در انتشارهای بعدی نیز شناسایی شوند. اجازه ندهید هیچ باگی دو بار اتفاق بیفتد.

۵,۲. آزمون واحد برای توسعه زمان ذخیره می‌کند.

برطرف کردن یک باگ نزدیک به زمان کشف آن، بسیار سریع‌تر از زمانی اتفاق می‌افتد که مجبور به پیدا کردن آن، دیباگ و حل آن در آینده هستید. به این زمان ذخیره‌شده، زمانی را که ایجاد تغییرات باعث ایجاد مشکل در کدهای قبلی می‌شود و شتاب پیوسته رو به جلو در پیشرفت پروژه را نیز اضافه کنید. البته این همیشه به معنی اتمام زودتر پروژه نیست اما قطعا زمان کمتری برای نگه‌داری[11] در طول دوره عمر پروژه صرف خواهید کرد. نگه‌داری معمولا از کارهای هزینه‌بری است که دوست داریم محدود کنیم.

۵,۳. تست‌های خودکار بسته به نیاز می‌توانند به طور متناوب اجرا شوند.

تست‌های دستی زجرآورند. با استفاده از تست‌های خودکار می‌توانید همه آنها را با یک دستور خط فرمان ساده یا بارگذاری مجدد مرورگر اجرا کنید. می‌توانید پیوسته از عملکرد کد منبع همانطور که از آن انتظار می‌رود مطمئن شوید و باگ‌ها را تقریبا همزمان با رخ دادن آنها شناسایی کنید. نوشتن تست‌ها هزینه‌ای است که یک بار پرداخت می‌کنید تا یک سطح از اطمینان همیشگی را به دست بیاورید.

۵,۴. آزمون واحد بهبود و تغییر کدها را آسان می‌کند

همه ما می‌دانیم تغییر کد ریسک‌های معمول خود را دارد. اجرا شدن کد همانند گذشته، امکان وجود باگ‌ها یا خطاهای منطقی اتفاقی، مشکل در تطابق با نسخه‌های قبلی API عمومی و یا PHP از جمله این ریسک‌ها هستند. مجموعه‌ای از تست‌ها می‌توانند جلوی هر کدام از این ریسک‌ها را بگیرند. تا جایی که تست‌های شما بر روی API عمومی و مبتنی بر نتیجه مورد انتظار از کلاس‌ها نوشته شده باشد، می‌توانید با تکیه بر تست‌های نوشته شده کدها را بهبود و یا تغییر دهید و تعدیل کنید. یک سیستم یکپارچه‌سازی[12] نیز می‌تواند وظیفه انجام تست‌ها بر اساس نسخه‌های مختلف PHP و یا سیستم‌عامل‌های مختلف را بر عهده بگیرد.

۵,۵. آزمون واحد طراحی کد را بهبود می‌بخشد، به خصوص زمانی که با TDD همراه شود.

---

[10] Production code
[11] Maintenance
[12] Integration

نوشتن تست‌ها برای مشخص کردن آنچه باید کلاس انجام بدهد کمک می‌کند تا بفهمیم کلاس‌ها چطور باید طراحی شود. همچنین با افزودن Mock Object ها می‌توان به طراحی چندین کلاس و ارتباط بین آنها نیز جهت داد. حتی بدون TDD، نوشتن تست‌ها همزمان که در حال کدنویسی هستید طراحی را بهبود می‌دهد زیرا کلاس‌های قابل تست قطعا نیاز دارند تا منعطف بوده و به خوبی از همدیگر جدا شده باشند.

۵,۶. آزمون واحد نوعی مستندسازی است.

به چیزی که یک تست انجام می‌دهد فکر کنید: شرح یک مورد استفاده[۱۳] یا یک مثال. یک مثال در مستندات به اندازه هزاران کلمه برای توسعه‌دهنده ارزش دارد. همچنین بسیاری از چارچوب‌های آزمون واحد ابزاری به نام «testdox» برای خلاصه‌سازی مستندات ارائه می‌کنند که اساسا لیستی از اظهارات هر مثال است. در اصل این مستندات خلاصه‌ای از مشخصات کلاس‌ها فراهم می‌کند که حتی توسط غیربرنامه‌نویس‌ها نیز قابل فهم است.(مانند مدیر پایگاه داده)

۵,۷. آزمون واحد شما را مجبور می‌کند تا مستقیما با مشکل روبرو شوید.

یکی از مشکلاتی که معمولا رخ می‌دهد این است که توسعه‌دهنده بدون اینکه ساختار یا برنامه‌ریزی‌ای داشته باشد تلاش می‌کند تا با نوشتن کد مشکلی را حل کند. با نوشتن تست شما جز روبرو شدن با نحوه حل مشکل با تمرکز بر روی API عمومی و نوشتن تست‌های موثر راهی ندارید. در واقع آزمون واحد مشوقی برای راه‌حل‌های ساده‌ی کاربردی است. در مجموع حرکت پیوسته رو به جلوی شما حفظ می‌شود و بازگشت به عقب به سختی اتفاق می‌افتد.

۵,۸. آزمون واحد باعث اعتماد به نفس می‌شود.

وقتی به کد منبع خود می‌توانید اطمینان کنید، باگ‌های فعلی و آینده را شناسایی کنید، تغییر دیگر یک پیشنهاد پرمخاطره نیست، زمان کمتری برای اصلاح صرف می‌کنید و زمان بیشتر صرف توسعه می‌شود، باعث تعجب نیست که تست و اختصاصا آزمون واحد اعتماد به نفس شما را افزایش دهد. شما به طور مستمر به جلو نگاه می‌کنید. به همه این‌ها ترجیح توسعه‌دهندگان به استفاده از کتابخانه‌ها، چارچوب‌ها و برنامه‌هایی که به خوبی تست شده‌اند را نیز اضافه کنید که مزیت بسیار ارزشمندی است.

۵,۹. آزمون واحد معیاری برای اتمام است.

اگر تست‌های مورد نیاز برای پوشش همه ویژگی‌ها و عملکردهای کلاس‌ها را نوشته باشید و همه آنها پاس شده باشند، منطقا می‌توانیم فرض کنیم شما آن بخش از کد را تمام کرده‌اید. ممکن است در آینده ویژگی جدیدی به کد اضافه شود اما تا آن روز کار کدنویسی می‌تواند متوقف شود. به یک معنا تست به خوبی با متدلوژی چابک چفت می‌شود تا نه تنها به طراحی جهت دهد بلکه نیازمندی‌های عملیاتی را نیز شناسایی و در بر بگیرد.

---

[13] Use case

۵,۱۰. آزمون واحد سرگرم‌کننده است.

با اعتماد به کدها، هزینه کم تغییرات، حل مشکلات با راه‌حل‌های ساده‌تر، امکان بهبود و بهینه‌سازی پروسه توسعه و تست سرگرم‌کننده و کاربردی خواهد بود. دیگر تست یک فرایند خسته‌کننده که توسعه‌دهندگان از آن فرار می‌کنند نیست.

نتیجه‌گیری

امروزه تست و آزمون واحد ضرورتی در برنامه‌نویسی است که به سختی قابل چشم‌پوشی است. با در نظر گرفتن مزایای تست، توجه کنید برای نوشتن تست‌ها نیازمند زمان و تجربه هستید. همانطور که یک توسعه‌دهنده خوب یک شبه ساخته نمی‌شود برای اینکه بتوانید از مزایای تست بهره‌مند شوید نیازمند یادگیری و تمرین آن هستید. به خاطر داشته باشید تست، آزمون واحد و یا توسعه مبتنی بر تست بیشتر از آنکه مستندات و کدها و چارچوب‌هایی برای مطالعه و استفاده باشند، شامل به‌روش‌ها و متدولوژی‌هایی هستند که نیازمند یادگیری عملی و کسب تجربه هستند.