فصل ۱۸ راهبردهای آزمون نرم افزار

مفاهیم کلیدی (مرتب بر حروف الفبا)

آزمون اعتبارسنجی ، آزمون الف و ب (آلفا و بتا) ، آزمون جامعیت ، آزمون دود ، آزمون رگرسیون ، آزمون سیستم ، آزمون واحد ، اشکال زدائی ، اعتبارسنجی و تعیین صحت (V&V) ، راهبردهای افزایشی(تکمیلی) ، گروه آزمون کنندگان مستقل(ITG) ؛ معیارهای تکاملی

KEY CONCEPTS

alpha and beta testing , criteria for completion , debugging , incremental strategies , ITG , integration testing , regression testing , smoke testing , system testing , unit testing , validation testing , V&V

نگاه اجمالی

راهبرد آزمون نرم‌افزاری چیست؟ طراحی موارد آزمونی مناسب و مؤثر (فصل ۷) مهم است، اما راهبرد مورد استفادهٔ شما برای اجرای این موارد آزمون مهم‌تر می‌باشد. آیا شما باید یک طرح رسمی برای آزمون‌های خود به‌وجود آورید؟ آیا شما باید کار آزمون را درکل برنامه، به‌عنوان یک آزمون کلی انجام دهید و یا آن را فقط در بخش کوچکی از برنامه اعمال کنید؟ آیا شما با اضافه شدن جزءهای جدید به یک سیستم بزرگ باید کار آزمون قبلی خود را مجدداً انجام دهید؟ شما چه زمانی باید مشتری را داخل کارهای خود کنید؟ این سؤالات و بسیاری از سؤالات دیگر هنگامی پاسخ داده خواهند شد که شما یک راهبرد امتحان‌کنندهٔ نرم‌افزاری را به‌وجود آورده و تکمیل کرده باشید.

چه کسی راهبرد آزمون نرم‌افزاری را برعهده دارد؟ راهبرد لازم برای آزمون کردن نرم‌افزار، توسط مدیر پروژه، مهندسین نرم‌افزار و متخصصان آزمون نرم‌افزاری به‌وجود آمده و تکمیل می گردد.

چرا راهبرد آزمون نرم‌افزاری از اهمیت برخوردار می‌باشد؟ آزمودن نرم‌افزار اغلب در مقایسه با سایر فعالیت مهندسی نرم‌افزار بیشتری به کارهای پروژه‌ای توجه دارد. اگر کار آزمودن به‌صورت تصادفی و اتفاقی انجام شود، زمان هدر می‌رود، انجام کارهای غیرضروری گسترش می‌یابد و در مراحل بدتر خطاهای نامشخصی ایجاد می‌گردد. بنابراین ایجاد یک راهبرد نظام مند برای آزمودن نرم‌افزار منطقی به نظر می‌رسد.

مراحل راهبرد آزمون نرم‌افزاری چه چیزهایی هستند؟ کار آزمون نرم‌افزاری از مراحل کوچک تا مراحل بزرگ ادامه می‌یابد. یعنی آزمون‌های اولیه بر یک جزء واحد تأکید دارند و از آزمون‌های جعبه سفید و جعبه سیاه استفاده می‌کند تا خطاهای موجود در منطق و کارکرد برنامه مشخص شود. بعد از آزمودن هر یک از جزءها باید کار تلفیق آنها انجام شود. کار آزمودن نرم‌افزار همراه با ساخته شدن نرم‌افزار ادامه می‌یابد. نهایتاً آن‌که، مجموعه‌ای از آزمون‌های بالا مرتبه نیز بعد از آماده به کار شدن کل برنامه اجرا می‌شوند. این آزمون‌ها برای مشخص کردن و تعیین خطاهای موجود در نیازمندیها، طراحی شده‌اند.

حاصل کار از راهبرد آزمون نرم‌افزاری چه چیزی می‌باشد؟ مشخصات آزمون با تعریف طرحی که توصیف‌کننده راهبرد کلی می‌باشد و با تعریف رویه ای که مراحل خاص آزمودن را تعیین می‌کند و با تعریف آزمون‌هایی که انجام خواهند شد، رهیافت تیم نرم‌افزاری را برای آزمون، مشخص می‌کند.

چگونه می‌توان مطمئن شد که راهبرد، درست تعیین شده است؟ با بررسی مجدد مشخصات آزمون قبل از آزمودن می‌توان کامل بودن موارد آزمونی و وظائف مربوط به آزمودن را مورد ارزیابی قرار داد. روش و طرح مؤثر آزمون به ساخت منظم و با قاعدهٔ نرم‌افزار و کشف خطاهای موجود در هر یک از مراحل فرآیند ساخت منتهی می‌شود.

راهبرد آزمون نرم‌افزار، شیوه‌های طراحی مورد آزمون نرم‌افزاری را با مجموعهٔ خوب طراحی شدهٔ مراحل مختلف، که موجب به‌وجود آمدن ساختار مناسبی برای نرم‌افزار می‌گردند، تلفیق می‌کند. این راهبرد یک نقشهٔ مسیر را به‌وجود می‌آورد که این نقشه نیز مراحلی را که باید به‌عنوان بخشی از آزمون نرم‌افزار انجام شوند، توصیف می‌کند و همچنین این نقشه نشان می‌دهد که هر یک از این مراحل در چه زمانی باید طرح‌ریزی و اجرا شوند و برای این امر چه مقدار انرژی، زمان و منابع مختلف مورد نیاز می‌باشد. بنابراین هر یک از راهبردهای آزمون نرم‌افزار باید طرح‌ریزی برای آزمون، طراحی مورد آزمون، انجام آزمون و جمع‌آوری اطلاعات حاصل و ارزیابی اطلاعات را با یکدیگر تلفیق کنند.

یک راهبرد آزمون نرم‌افزار باید به اندازهٔ کافی انعطاف‌پذیر باشد تا یک روش آزمون مشتری‌پسند را به‌وجود آورد. همچنین یک راهبرد باید به اندازهٔ کافی جدی و با صلابت باشد تا بتواند هنگام پیشرفت پروژه پی‌گیری منطقی در زمینهٔ طرح‌ریزی و مدیریت به‌وجود آورد. شومن **[SHO83]**[1] در مورد این مسایل این چنین به بحث می‌پردازد که:

"آزمودن در بسیاری از جنبه‌ها یک فرآیند فردی و منحصر به‌فرد می‌باشد و تعداد انواع مختلف آزمون به اندازهٔ تعداد رهیافت‌های مختلف تکمیل شده و توسعه یافته در طول سالیان سال متفاوت و متغیر می‌باشد. تنها وسیلهٔ دفاعی ما در برابر خطای برنامه‌ریزی، طراحی دقیق و هوش خدا دادی برنامه‌نویس می‌باشد. در حال حاضر ما در عصری هستیم که در آن فنون مدرن طراحی و تحقیقات فنی رسمی به ما

1.Shooman,M.L.

کمک می‌کنند تا تعداد خطاهای اولیه‌ای را که در کد به‌وجود می‌آیند به حداقل برسانیم. همچنین شیوه‌های مختلف آزمون نرم‌افزاری، خود را در چندین فلسفه و رهیافت متمایز، طبقه بندی نموده اند."

این "رهیافت‌ها و فلسفه‌ها همان چیزی است که ما آن را راهبرد[1] می‌نامیم. در فصل ۱۷، فنآوری آزمون نرم‌افزاری ارائه گردید.[2] در این فصل بر راهبرد آزمون نرم‌افزاری تأکید شده است.

۱۸-۱ یک رهیافت راهبردی برای آزمون نرم افزار

آزمودن شامل مجموعه‌ای از فعالیت‌ها می‌باشد که امکان طرح‌ریزی آن به‌صورت پیشرفته موجود می‌باشد و انجام این فعالیت‌ها نیز به‌صورت نظام مند است. و به همین دلیل باید یک الگو برای آزمون نرم‌افزار ـ یعنی مجموعه‌ای از مراحل مختلف که طی آن می‌توان فنون خاص طراحی موردی آزمون و شیوه های آزمودن را مشخص نمود ـ برای فرآیند نرم‌افزاری تعریف گردد.

در این مطلب به تعدادی از راهبردهای آزمون نرم‌افزاری اشاره خواهد شد. تمام این راهبردها الگویی را برای آزمودن نرم‌افزار برای افراد توسعه‌دهندهٔ نرم‌افزاری ایجاد کرده‌اند و تمام این راهبردها دارای ویژگی‌های عمومی ذیل می‌باشند:

- آزمودن در سطح اجزا[3] آغاز می‌شود و به سمت تلفیق کل سیستم کامپیوتری گام برمی‌دارد.
- فنون مختلف آزمون، در مقاطع مختلف زمانی مناسب، باید انجام گیرد.
- کار آزمون توسط افراد توسعه‌دهندهٔ نرم افزار انجام می‌شود و در پروژه‌های بزرگ این کار توسط یک گروه مستقل آزمون‌کننده انجام خواهد شد.
- آزمودن و اشکال‌زدایی دو فعالیت متفاوت می‌باشند، اما اشکال‌زدایی باید در هر یک از راهبردهای آزمودن گنجانده شود.

راهبرد آزمون نرم‌افزار باید با آزمون‌های سطح پایین که برای تأیید اجرای صحیح بخش کوچک برنامه منبع و با آزمون‌های سطح بالایی که به کارکردهای اصلی سیستم در برابر نیازمندیهای در نظر گرفته شده از سوی مشتریان اعتبار می‌بخشد، هماهنگی داشته باشد. راهبرد باید دستورالعمل‌هایی را برای فرد متخصص و مجموعه‌ای از نقاط اساسی را برای مدیر فراهم کند. از آن‌جایی‌که مراحل مختلف راهبرد آزمون نرم‌افزاری در هنگام افزایش فشار زمانی اتفاق می‌افتد، بنابراین پیشرفت این مراحل را می‌توان مورد ارزیابی و سنجش قرار داد و مشکلات نیز هرچه زودتر ظاهر می‌شوند.

ارجاع به وب

اطلاعات مفیدی در خصوص راهبردهای آزمون نرم افزار توسط روزنامه آزمون نرم افزار (STN) تهیه گردیده است.

www.ondaweb.com/sti/newsltr.htm

1.Strategy

۲. برای سیستم های شئ گرا، آزمون از سطح شئ یا از سطح کلاس آغاز خواهد شد. فصل ۲۳ را برای جزئیات بیشتر نگاه کنید.

۳. در فصل ۲۳ آزمونهای مربوط به سیستم های شئ گرا توضیح داده شده اند.

۱۸-۱-۱ تعیین صحت و اعتبار سنجی

آزمون نرم‌افزاری یکی از عناصر مهم موضوع گسترده‌تری است که اغلب با نام تعیین صحت و اعتبارسنجی (تصدیق)[1] (V&V) به آن اشاره می‌شود. تعیین صحت[2] به مجموعه‌ای از فعالیت‌هایی اشاره دارد که اجرای یک کارکرد خاص به‌صورت صحیح توسط نرم‌افزار را تضمین و تأیید می‌کند. اعتبارسنجی یا تصدیق[3] به مجموعهٔ مختلفی از فعالیت‌های تضمین‌کنندهٔ این امر اشاره دارد که نرم‌افزار ساخته شده با شرایط موردنظر مشتری هماهنگی دارد. بوهم این تعاریف را به‌صورت زیر بیان می‌کند:[BOE81][4]

تعیین صحت: "آیا ما فرآوردهٔ خود را به‌صورت صحیح و مناسب ساخته‌ایم؟"

اعتبارسنجی یا تصدیق: "آیا ما فرآوردهٔ مناسب و درستی ساخته‌ایم؟"

ارجاع به فصلها

فعالیتهای تضمین کیفیت نرم افزار (SQA) به تفصیل در فصل ۸ توضیح داده شده است.

تعریف تعیین صحت و اعتبارسنجی(تصدیق) بسیاری از فعالیت‌هایی را تحت‌پوشش قرار می‌دهد که ما با نام تضمین کیفیت نرم‌افزار[5] (SQA) به آنها اشاره کرده‌ایم.

تصدیق و تعیین صحت آرایهٔ گسترده‌ای از فعالیت‌های SQA را تحت‌پوشش قرار می‌دهد که این فعالیت‌ها شامل موارد ذیل می‌باشند:

بازبینی فنی و رسمی، وارسی کیفیت و پیکربندی، نظارت بر عملکرد، شبیه‌سازی، مطالعه امکان‌سنجی بررسی مستندات، بررسی بانک اطلاعاتی، تجزیه و تحلیل الگوریتمی، آزمون‌های تکمیلی، آزمون کیفیت، و آزمون مربوط به نصب. اگر چه آزمودن نقش بسیار مهمی را در بازبینی و تصدیق ایفا می‌کند، اما انجام سایر فعالیت‌ها نیز مهم و ضروری می‌باشد.

نقل قول

آزمون، بخش اجتناب ناپذیری از یک تلاش قابل دفاع برای ساخت سیستم نرم افزاری است. ویلیام هودن

آزمون، حفاظتهایی را ایجاد می‌کند که از روی آن می‌توان کیفیت را ارزیابی کرد و خطاها را مشخص نمود. اما آزمودن را نباید به‌عنوان شبکهٔ ایمنی در نظر گرفت. طبق اظهارات موجود " شما نمی‌توانید کیفیت را آزمون کنید. اگر قبل از آن‌که شما کار آزمون را آغاز کنید کیفیتی وجود نداشته باشد، بعد از اتمام کار آزمون نیز کیفیتی وجود نخواهد داشت." کیفیت از کل فرآیند مهندسی نرم‌افزاری به نرم‌افزار اضافه می‌شود. استفادهٔ مناسب از شیوه ها و ابزار، بررسی فنی و رسمی دقیق و مناسب و مدیریت قاطع و اندازه‌گیری دقیق، همگی به بوجود آمدن کیفیت منتهی می‌گردند که این کیفیت نیز در طول کار آزمون مورد تأیید قرار می‌گیرد.[WAL89][6]

---

1. Verification And Validation
2. Verification
3. Validation
4. Boehm,B.
5. Software Quality Assurance
6. Wallace,D.R. and R.U.

میلر [MIL77][1] بیان این مطلب که "انگیزهٔ مهم و اصلی در آزمون برنامه تأیید کیفیت نرم‌افزار با کمک شیوه هایی است که این شیوه ها از لحاظ اقتصادی در سیستم‌هایی با مقیاس کوچک و یا مقیاس بزرگ قابل اعمال هستند"، میان آزمون نرم افزار و تضمین کیفیت رابطه‌ای را ایجاد می‌کند.

۱۸-۱-۲سازمان دهی آزمون نرم افزار

در هر پروژهٔ نرم‌افزاری اختلاف سلیقهٔ عمده‌ای با شروع شدن کار آزمون وجود خواهد داشت. در این مرحله از افراد سازندهٔ نرم‌افزار تقاضا می‌شود تا نرم‌افزار ساخت خود را بیازمایند. و این مسأله به‌خودی خود بی‌ضرر و بی‌خطر می‌باشد از طرف دیگر چه کسی می‌تواند برنامه را بهتر از فرد توسعه‌دهندهٔ آن بشناسد؟ متأسفانه افراد توسعه‌دهندهٔ برنامه‌های مختلف با بیان مطالب زیر به منفعتی خواهند رسید: برنامه عاری از خطا می‌باشد. برنامه مطابق با شرایط موردنظر مشتری کار خواهد کرد. و برنامه بر اساس طرح موجود و در محدودهٔ بودجهٔ موجود تکمیل خواهد شد. هر یک از این منفعت‌ها موجب کاهش آزمون کامل و دقیق می‌گردند.

از نقطه‌نظر روان‌شناسی، تحلیل و طراحی نرم‌افزاری (همراه با برنامه نویسی) جزء کارهای سازنده[2] می‌باشند. مهندسین نرم‌افزار یک برنامهٔ کامپیوتری، اسناد مربوط به آن برنامهٔ کامپیوتری و ساختار داده‌های هر نقطه را ایجاد می‌کنند. مهندسین نرم‌افزار نیز همچون سایر سازندگان به بنای (در اینجا نرم‌افزار) ساخته شده افتخار می‌کنند؟ با کسانی که سعی در نابودسازی این بنای ساخته شده دارند مخالفت می‌کنند. با شروع کار آزمون، کار ظریف و حساس اما محدودی برای "درهم شکستن" آن‌چه که مهندس نرم‌افزار ساخته است، وجود خواهد داشت. از نقطه نظر شخص سازنده آزمون (از نظر فلسفی) نابود کننده[3] است. به عبارت دیگر سازنده به انجام دادن و طراحی آزمون‌هایی می‌پردازد که به‌جای نشان دادن خطاهای موجود در نرم‌افزار به تأیید این نکته بپردازند که این برنامهٔ ساخته شده و به‌وجود آمده به‌خوبی کار می‌کند. متأسفانه در هر برنامه‌ای خطاهایی وجود خواهد داشت و اگر مهندس نرم‌افزار نتواند این خطاها را پیدا کند، مشتری متوجه آنها خواهد شد!

یک گروه مستقل آزمون، فاقد تعارضی هستند که در سازندگان نرم افزار دیده می شود.

اغلب برداشت‌های نادرستی از بحث فوق استنتاج می‌شود. این برداشت‌های نادرست عبارتند از: ۱) خود تکمیل کنند و توسعه‌دهندهٔ نرم‌افزار نباید به هیچ‌وجه کار آزمون را انجام دهد: ۲) نرم‌افزار باید در اختیار افراد غریبه قرار گیرد تا آنها با بی‌رحمی آن را آزمون کنند: ۳) افراد آزمون‌کننده فقط هنگام شروع مراحل آزمون در کار پروژه دخالت خواهند داشت. هر یک از این برداشت‌ها نادرست می‌باشند.

1.Miller,E.

2.Contructive Tasks

3.destructive

فرد تکمیل‌کننده و توسعه‌دهندهٔ نرم‌افزار همیشه مسئولیت آزمون هر یک از واحدهای برنامه را برعهده دارد و با انجام این آزمون‌ها مطمئن می‌شود که هر یک از این واحدها کارکرد طراحی شده برای آنها را به‌خوبی و به‌درستی انجام می‌دهند. در بسیاری از موارد، فرد توسعه‌دهندهٔ نرم‌افزار آزمون تلفیقی - یعنی آزمودن مرحله‌ای که به ساخت و آزمون معماری کامل برنامه منتهی می‌شود - را انجام می دهد.

نقش گروه مستقل آزمون‌کننده[1] (ITG) برطرف کردن مشکلات اساسی مربوط به اجازه دادن به سازنده برای آزمون آن‌چه که ساخته شده است، می‌باشد. آزمون‌های مستقل اختلافات موجود در زمینهٔ منافع را برطرف خواهند ساخت و علاوه بر تمام این موارد، به پرسنل حاضر در تیم گروه مستقل مبلغی پرداخته می‌شود تا خطاهای موجود در نرم‌افزار را بیابند.

ترمیه
اگر یک گروه آزمون مستقل (ITG) در سازمان شما وجود ندارد، شما موظفید چنین نقطه نظری را به وجود آورید. هنگامی که مسئول آزمون هستید، سعی نمایید نرم افزار را بشکنید.

با این وجود مهندسین نرم‌افزار کنترل و مدیریت برنامه را به ITG واگذار نمی‌کنند. فرد توسعه‌دهندهٔ نرم‌افزار و گروه ITG با دقت کل پروژهٔ نرم‌افزاری را بررسی می‌کنند تا مطمئن شوند که کارهای مربوط به آزمون نرم‌افزار و برنامه به دقت و به‌طور کامل انجام شده است. در هنگام اجرای کار آزمون، فرد توسعه‌دهندهٔ نرم‌افزار باید حضور داشته باشد تا خطاهای کشف شده در حین آزمون را اصلاح کند.

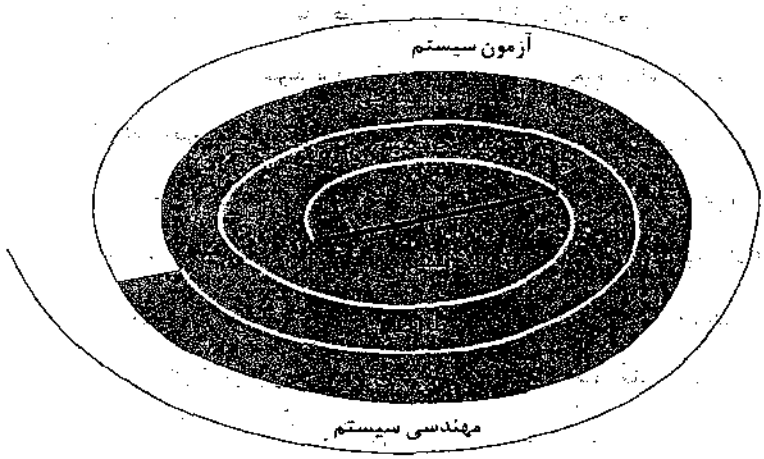
ITG بخشی از تیم پروژهٔ تکمیل نرم‌افزار می‌باشد و این امر بدین معناست که ITG در طول فعالیت تشخیص خطاها و (در طرح‌ریزی و مشخص کردن شیوه‌های آزمون) در کل یک پروژهٔ بزرگ نیز دخالت خواهد داشت. با این وجود در بسیاری از موارد ITG به سازمان تضمین کیفیت نرم‌افزاری گزارش می‌کند که این گروه باید به درجه‌ای از استقلال و مستقل شدن برسد که این استقلال در صورتی‌که این گروه بخشی از سازمان مهندسی نرم‌افزار باشد حاصل نخواهد شد و عملی نمی‌باشد.

پرسش
راهبرد کلی آزمون نرم افزار کدام است؟

۱۸-۱-۳ یک راهبرد آزمون نرم افزار

فرآیندهای مهندسی نرم‌افزار را می‌توان به‌صورت حلزونی نشان داده شده در شکل ۱۸-۱ درنظر گرفت. در ابتدا مهندسی سیستم یا مهندسی اطلاعات نقش نرم‌افزار را تعریف می‌کند و سپس به تحلیل مشخصات نرم‌افزاری می‌رسد که در این مرحله معیارهای مربوط حیطهٔ اطلاعات، کارکرد، رفتار، عملکرد، محدودیت‌ها و معیارهای اعتبارسنجی و تصدیق نرم‌افزار ایجاد می‌شوند. با حرکت به سمت بخش‌های داخلی این نمودار حلزونی شکل به طراحی و برنامه نویسی خواهیم رسید. برای تکمیل و توسعهٔ نرم‌افزار کامپیوتری، ما در امتداد خط جریان کاهش‌دهندهٔ سطح انتزاعی در هر دور به سمت داخل حرکت پیچشی خواهیم داشت.

1 Independent Test Group



شکل ۱۸ - ۱ راهبرد آزمون

راهبرد آزمون نرم‌افزاری را می‌توان در بافت حلزونی (شکل ۱۸-۱) نیز درنظر گرفت. آزمون واحد[1] در مرکز این مارپیچ انجام می‌شود و در این آزمون بر هر یک از واحدهای نرم‌افزاری در هنگام اجرا در برنامه منبع تأکید می‌شود. کار آزمون با حرکت به سمت خارج در امتداد بخش حلزونی ادامه می‌یابد و به آزمون تلفیقی[2] نزدیک می‌شود که در این آزمون نیز بر طراحی و ساخت معماری نرم‌افزار تأکید می شود. با حرکت در حلقهٔ بعدی این مارپیچ با آزمون اعتبارسنجی یا تصدیق[3] مواجه خواهیم شد که در این آزمون نیازمندیهای ایجاد شده به‌عنوان بخشی از تحلیل نیازمندیها و مشخصات نرم‌افزاری در برابر نرم‌افزار ساخته شده مورد تأیید و تصدیق قرار می‌گیرند. و در نهایت به آزمون سیستم[4] می‌رسیم که در آن نرم‌افزار و سایر عناصر سیستم به‌صورت کلی مورد آزمون واقع می‌شوند. برای آزمون نرم‌افزار کامپیوتری در امتداد خط جریان گسترش‌دهندهٔ حیطهٔ آزمون در هر یک از دورهای این نمودار حلزونی به سمت خارج حرکت می‌کنیم.

ارجاع به فصلها

تکنیکهای آزمون جعبه سفید و جعبه سیاه در فصل ۱۷ توضیح داده شده اند.

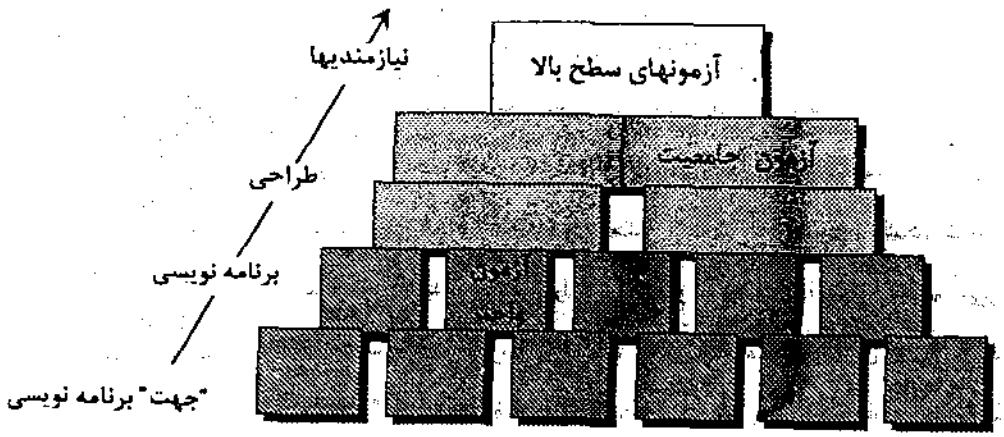
نقل قول

اگر جهت گیری آزمون آن باشد که تنها کاربران نهایی نیازمندیشان رفع شود، مانند ساخت یک ساختمان براساس نظرات یک دکور ساز

با در نظر گرفتن فرایند از نقطه نظر رویه‌ای (روش کار)، آزمودن در محدودهٔ بافت مهندسی نرم‌افزار در مواقع مجموعه‌ای از چهار مرحله خواهد بود که این مراحل پشت سر هم اجرا خواهند شد. این چهار مرحله در شکل ۱۸-۲ نشان داده شده‌اند. در ابتدا آزمون بر هر یک از جزءها (بخش‌ها) به‌صورت مجزا تأکید دارد تا از این امر اطمینان حاصل شود که هر یک از جزءها به‌عنوان یک واحد، به‌خوبی و به صورت مناسبی کار خود را انجام می‌دهند. بنابراین نام این مرحله از آزمون ، آزمون واحد می‌باشد. آزمون واحد از فنون آزمون ورودی و خروجی و روابط میان آنها به میزان زیادی استفاده می‌کند و راههای خاصی را در

1.Unit testing
2.integration testing
3.validation testing
4.system testing

ساختار کنترل واحد به‌کار می‌برد تا مطمئن شود که این آزمون به‌طور کامل تمام بخش‌های واحد را پوشش داده است و کار خطایابی نیز به اندازهٔ حداکثر انجام شده است. در مرحلهٔ بعدی آزمون، جزءها باید با یکدیگر تلفیق شوند تا بستهٔ کامل نرم‌افزاری را به‌وجود آورند. آزمون تلفیقی مسایل مربوط به مشکلات دوگانهٔ تعیین صحت و پساخت برنامه را مورد بررسی قرار می‌دهد. فنون طراحی مورد آزمون جعبهٔ سیاه جزء شایع‌ترین و رایج‌ترین فنون موجود در مرحلهٔ آزمون تلفیقی می‌باشند، اگر چه در این آزمون‌ها میزان محدودی از آزمون جعبهٔ سفید (ورودی و خروجی و روابط میان آنها) نیز برای مطمئن شدن از تحت پوشش قرار گرفتن مسیرهای اصلی کنترل مورد استفاده قرار خواهد گرفت. بعد از ساخته شدن نرم‌افزار، مجموعه‌ای از آزمون‌های سطح بالا نیز انجام خواهد شد. معیار تصدیق (که در طول تحلیل نیازمندیها ایجاد شده است) نیز باید مورد آزمون قرار گیرد. آزمون اعتبارسنجی تضمین‌کنندهٔ نهایی این امر می‌باشد که نرم‌افزار تمام نیازمندیهای کارکردی، رفتاری و عملکردی را دارا می‌باشد. فنون آزمون جعبهٔ سیاه منحصراً برای اعتبارسنجی نرم‌افزار و در طول آن مورد استفاده قرار می‌گیرند.



شکل ۱۸-۲ گامهای آزمون نرم افزار

آخرین مرحلهٔ آزمون سطح بالا در خارج از حیطهٔ مهندسی نرم‌افزار می‌باشد و این آزمون در محدودهٔ بافت مهندسی سیستم کامپیوتر، قرار دارد. نرم‌افزار تعیین اعتبار شده باید در کنار سایر اجزاء سیستم (به‌عنوان مثال سخت‌افزار، افراد، بانکهای اطلاعاتی) قرار داده شود و با آنها ترکیب شود. آزمون سیستم مؤید و تأییدکنندهٔ این مطلب است که تمام اجزا به طرز صحیح و مناسبی در یک شبکه قرار گرفته‌اند و کارکرد/ عملکرد کلی سیستم نیز حاصل شده است.

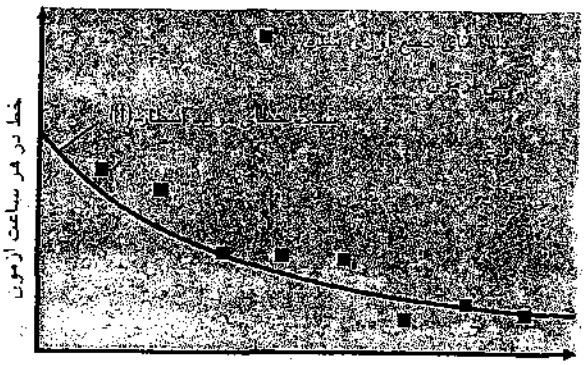
۱۸-۱-۴ معیار تکمیل آزمون

پرسش

چه هنگام آزمون را به پایان برده ایم؟

هنگام مورد بحث قرار دادن آزمون نرم‌افزاری، سؤال قدیمی و کلاسیک ذیل در ذهن نقش می‌بندد: "چه زمانی ما کار آزمودن را به‌طور کامل انجام داده‌ایم - از کجا می‌توان فهمید که کار آزمودن به اندازهٔ کافی انجام شده است؟" متأسفانه پاسخ قطعی به این سؤال وجود ندارد، اما پاسخ‌های عملی معدودی در این راستا وجود دارد و کارهای مقدماتی معدودی در ارتباط با دستورالعمل‌های تجربی انجام شده است.

یکی از پاسخ‌های موجود به سؤال فوق به‌صورت زیر می‌باشد: "شما هرگز کار آزمودن را انجام نمی‌دهید و مسئولیت آزمون نرم‌افزار را به مشتری خود محول می‌کنید." در هر دفعه که مشتری/ مصرف‌کننده یک برنامهٔ کامپیوتری را اجرا می‌کند، برنامهٔ مفروض بر اساس مجموعهٔ جدیدی از اطلاعات مورد آزمون قرار می‌گیرد. این واقعیت عاقلانه بر اهمیت سایر فعالیت‌های تضمین کیفیت تأکید دارد. پاسخ دیگری که به این سؤال داده شده است به‌صورت زیر می‌باشد: "شما کار آزمون را هنگامی انجام می دهید که وقتتان یا پولتان تمام شده است."



شکل۱۸-۳ شدت خطا وشکست به عنوان تابعی از زمان اجرا

ارجاع به فصلها

Use-case. که یک سناریو را برای استفاده از نرم افزار توصیف می کند. در فصل ۱۷ توضیح داده شده است.

اگر چه فقط تعداد معدودی از کارورزان در مورد این پاسخ‌ها به بحث و بررسی پرداخته‌اند، اما مهندسین برای تعیین زمان انجام شدن آزمون کافی به معیار دقیق‌تری نیاز دارند. موسا و آکرمن [MUS89][1] در پاسخ به این سؤال از معیارهای آماری استفاده کرده‌اند و پاسخ خود را بدین‌صورت بیان کرده‌اند: " نه، ما نمی‌توانیم به‌طور کامل مطمئن باشیم که نرم‌افزار هرگز از کار نخواهد افتاد، اما با توجه به مدل آماری تصدیق شده از لحاظ تجربی و نظری ما آزمون کافی را انجام داده‌ایم و می‌توانیم با ۹۵ درصد اطمینان بگوییم که انجام کار و عملکرد این نرم‌افزار در ۱۰۰۰ ساعت کار پردازنده، بدون به‌وجود آمدن مشکل و عیب در یک محیط، تعریف شده به‌صورت احتمالی حداقل ۰/۹۵۵ می‌باشد." [MUS89]

1.Musa ,J.D. and Ackerman

با استفاده از مدل‌سازی آماری و تئوری اعتبار نرم‌افزاری می‌توان مدل‌های عیب نرم‌افزاری را به شکل تابعی از زمان اجرا نشان داد. نسخه‌ای از مدل عیب نرم‌افزاری به نام مدل زمان اجرای پواسن لگاریتمی[1] به شکل تابع زیر می‌باشد:

$$f(t) = (1 / P) \ln [I_0 pt + 1] \qquad (۱-۱۸)$$

که در این فرمول:

$f(t)$ = مجموع تعداد معایبی که انتظار می‌رود در هنگام آزمون نرم‌افزار در مقادیر مشخصی از زمان اجرا، یعنی t، به‌وجود آید.

I_0 = شدت عیب اولیهٔ نرم‌افزار در ابتدای آزمون،

P = کاهش نمایی در شدت عیب با کشف شدن خطاها و انجام اصلاحات لازم.

شدت لحظه‌ای عیب، یعنی $I(t)$ را می‌توان با مشتق‌گیری از $f(t)$ بدست آورد. بنابراین مقدار $I(t)$ به‌صورت زیر خواهد شد:

$$I(t) = I_0 / (I_0 pt + 1) \qquad (۲-۱۸)$$

آزمون‌کنندگان با استفاده از روابط ذکر شده در فرمول ۱۸-۲ می‌توانند با انجام شدن مراحل مختلف آزمون میزان کاهش خطا را پیش‌بینی کنند. می‌توان شدت خطای واقعی را در برابر منحنی پیش‌بینی شده قرار داد (شکل ۱۸-۳). اگر داده‌های واقعی در طول آزمون، جمع‌آوری شده باشند و مدل زمان اجرای پواسون لگاریتمی در تعداد نقاط داده‌ای به‌صورت منطقی نزدیک به یکدیگر باشند، می‌توان آن مدل را برای پیش‌بینی کل زمان آزمون مورد نیاز برای دستیابی به شدت عیب کاهش یافته، مورد استفاده قرار داد. [GIL95][2]

چه رهنمودهایی برای یک راهبرد موفق آزمون، وجود دارند؟

با جمع‌آوری متریک‌ها در طول آزمون نرم‌افزار و استفاده از مدل‌های موجود در زمینهٔ قابلیت اعتبار نرم‌افزار می‌توان دستورالعمل مفیدی را برای پاسخ‌گویی به سؤال ذیل ایجاد کرد: "چه موقع کار آزمون نرم‌افزار به‌طور کامل انجام شده است؟" شک و شبههٔ کمی در این زمینه وجود دارد که کارهای بیشتری قبل از ایجاد شدن قوانین کمی برای آزمون ، انجام داده نشده باقی خواهند ماند اما روش‌های تجربی موجود در این زمینه بهتر از اطلاعات و عقاید خام می‌باشند.

1.logarithmic Poisson execution-time model

2.Gilb, T.

۱۸-۲ موضوعات راهبردی

در قسمت‌های بعدی این فصل یک راهبرد نظام مند برای آزمون نرم‌افزار ارائه خواهد شد. اما حتی بهترین راهبردها نیز در صورت مورد بررسی قرار نگرفتن مجموعه‌ای از مشکلات و مسایل درجه اول، به‌خوبی در زمینهٔ آزمون نرم‌افزارها عمل نخواهند کرد. *تام جیلب* اظهار داشته است که مسایل ذیل باید برای اجرای راهبرد مؤفق آزمون نرم‌افزاری مورد بررسی قرار داده شوند:

مشخص کردن مشخصات محصول به شکل قابل اندازه‌گیری، مدت‌ها قبل از شروع آزمون. اگر چه هدف مهم و اصلی آزمون، یافتن خطاها می‌باشد، اما راهبرد آزمون باید سایر ویژگی‌های کیفی از جمله قابلیت حمل، قابلیت حفظ و نگهداری و قابلیت استفاده (فصل ۱۹) را نیز مورد ارزیابی قرار دهد. و این کارها باید به شیوه‌ای قابل اندازه‌گیری انجام شوند، به‌گونه‌ای که نتایج کار آزمون بدون ابهام باشد.

بیان کردن اهداف آزمون به‌صورت واضح و مشخص. اهداف خاص آزمون باید با استفاده از واژه‌های قابل سنجش بیان شوند. به‌عنوان مثال مؤثر بودن آزمون، پوشش‌دهی آزمون، میانگین زمان برای به‌وجود آمدن عیب نرم‌افزاری، هزینهٔ لازم برای یافتن مفایب و برطرف نمودن آنها، میزان عیب نرم‌افزاری باقی‌مانده، یا تعداد دفعات به‌وجود آمدن عیب نرم‌افزاری و ساعت کار آزمون در هر آزمون رگرسیون، باید در طرح آزمون گنجانده شوند.

درک نمودن کاربران نرم‌افزار و ایجاد یک پروفیل جداگانه برای هر یک از طبقات کاربران. موارد کاربرد که توصیف‌کنندهٔ سناریوی محاوره و تعامل در هر یک از کلاسهای کاربران نرم‌افزاری می‌باشد می‌توانند با متمرکز ساختن آزمون بر روی کاربرد واقعی محصول از میزان کل کارهای لازم برای انجام آزمون بکاهند.

ایجاد یک طرح در زمینهٔ آزمون که بر "چرخهٔ سریع آزمون" تأکید دارد. جیلب تاکید می‌کند که تیم مهندسی نرم‌افزار باید بیاموزند که کار آزمون را در چرخه‌های سریع و در زمینه‌های مفید برای مشتری انجام دهند که این زمینه‌ها در افزایش کارکرد یا بهبود کیفیت قابل آزمون می‌باشند. بازخوردایجاد شده از طریق این آزمون‌های چرخه سریع را می‌توان برای کنترل سطح کیفیت و راهبردهای مربوط به آزمون استفاده کرد.

ساخت یک نرم‌افزار "مقاوم" که برای آزمون، خود طراحی شده است. نرم‌افزارها باید به شیوه‌ای طراحی شوند که بخوانند از فنون ضد خطا استفاده کنند. یعنی نرم‌افزار باید بتواند انواع خاص خطاها را تشخیص داده و شناسایی کند. علاوه بر این موارد در طراحی نرم‌افزار، باید آزمون خودکار و آزمون رگرسیون نیز گنجانده شده باشد.

استفاده کردن از بازبینی‌های فنی و رسمی به‌عنوان یک فیلتر قبل از شروع آزمون. بازبینی‌های فنی و رسمی (فصل ۸) می‌توانند به اندازه آزمون در تعیین خطاها، مفید و مؤثر واقع شوند. و

به همین دلیل این بازبینی‌ها می‌توانند از میزان کار مربوط به آزمون که برای تولید نرم‌افزار با کیفیت مورد نیاز می‌باشد، بکاهند.

انجام بازبینی‌های فنی و رسمی برای ارزیابی راهبرد آزمون و موارد آزمون. بازبینی‌های فنی و رسمی می‌توانند خطاهای مربوط به ناسازگاری، حذف یا کامل نبودن را در رهیافت آزمون نرم‌افزاری مشخص کنند و این امر موجب صرفه‌جویی در وقت و بهبود کیفیت محصول می‌شود.

ابعاد یک رهیافت مداوم اصلاح و بهبود برای فرآیند آزمون نرم‌افزاری. راهبرد آزمون نرم‌افزاری باید مورد ارزیابی و سنجش قرار داده شود. متریک‌های جمع‌آوری شده در طول آزمون نرم‌افزاری باید به‌عنوان بخشی از روش کنترل فرآیند آماری برای آزمون نرم‌افزار مورد استفاده قرار گیرند.

۱۸-۳ آزمون واحد

در آزمون واحد بر تعیین صحت کوچک‌ترین واحد طراحی نرم‌افزار ـ یعنی جزء و مؤلفهٔ نرم‌افزاری یا پیمانه نرم‌افزاری ـ تأکید می‌شود. آزمون واحد مبتنی بر جعبه سفید بوده و به طور موازی برای چند جزء قابلیت کاربرد دارد.

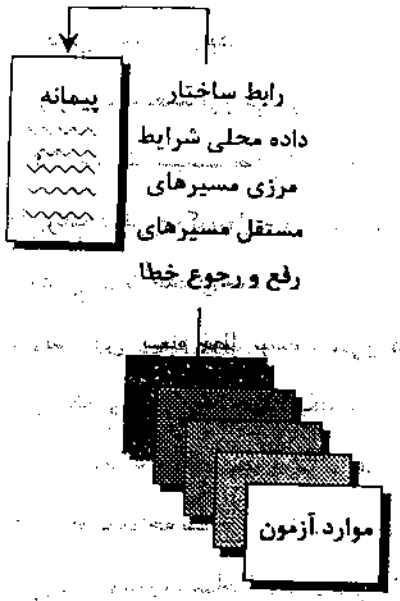
۱۸-۳-۱ ملاحظات آزمون واحد

آزمون واحد

آزمون‌هایی که به‌عنوان بخشی از آزمون‌های واحد انجام می‌شوند در شکل ۱۸-۴ به‌صورت الگو نشان داده شده‌اند. رابط پیمانه برای مطمئن شدن از جریان یافتن صحیح اطلاعات به واحد برنامهٔ مورد آزمون و خارج شدن صحیح اطلاعات از این واحد مورد آزمون قرار داده می‌شوند. ساختار داده‌های محلی برای مطمئن شدن از این امر که داده‌های موقتاً* ذخیره شده، یکپارچگی خود را در طول تمام مراحل موجود در اجرای الگوریتم حفظ می‌کنند، مورد آزمون قرار می‌گیرند. برای مطمئن شدن از عملکرد مناسب و صحیح پیمانه‌ها در مرزهای ایجاد شده برای محدودسازی فرآیند، باید کرانه‌ها را مورد آزمون قرار داد.

تمام مسیرهای مستقل در سر تا سر ساختار کنترل نیز برای مطمئن شدن از این امر که تمام دستورات و جملات موجود در پیمانه حداقل یک‌مرتبه اجرا شده‌اند، آزموده شده و در نهایت آن‌که تمام مسیرهای کنترل خطا نیز در آزمون نرم‌افزاری آزمون می‌شوند.

آزمون جریان داده‌ها در رابط پیمانه باید قبل از شروع سایر آزمون‌ها انجام شود. اگر اطلاعات به طرز صحیح وارد یا خارج نشوند، سایر آزمون‌ها نیز مورد تردید قرار خواهند گرفت. به‌علاوه ساختار داده‌های محلی نیز باید مورد آزمون واقع شوند و تأثیر محلی آنها بر داده‌های سراسری نیز باید در طول آزمون واحد مشخص گردد.



شکل ۱۸-۴ آزمون واحد

پرسش

کدام خطاهای مشترک در طی یک آزمون یافت خواهند شد؟

آزمون انتخابی مسیرهای اجرایی یکی از کارهای مهمی است که باید در طول آزمون واحد انجام شود. موارد آزمون باید برای شناسایی خطاهای حاصل از محاسبات نادرست، مقایسه‌های نادرست و یا جریان نامناسب کنترل طراحی شوند. آزمون مسیر و حلقهٔ اصلی برای مشخص شدن آرایهٔ گسترده‌ای از خطاهای مسیری مؤثر و مفید می‌باشند.

از میان خطاهای شایع و رایج در محاسبات می‌توان به موارد زیر اشاره کرد: ۱) عدم درک از حق تقدم محاسباتی صحیح ۲) عملکردهای دستور ترکیبی، ۳) مقدار دهی اولیه نادرست، ۴) عدم دقت محاسباتی، ۵) نمایش نمادین نادرست یک عبارت. جریان مقایسه و جریان کنترل به‌صورت دقیقی با یکدیگر جفت می‌شوند (یعنی تغییر جریان به‌صورت مداوم و پشت سر هم پس از مقایسه اتفاق می‌افتد). موارد آزمونی باید خطاهای ذیل را مشخص کنند: ۱) مقایسه انواع مختلف داده‌ها، ۲) ترتیب‌های منطقی نادرست ۳) امید برابری هنگامی که خطاهای مربوط به دقت، برابری را غیر متحمل می‌سازد، ۴) مقایسهٔ نادرست متغیرها؛ ۵) پایان یافتن نامناسب یا عدم خروج از حلقه ۶) عدم توانایی برای خروج هنگام تکمیل شمارش شمارنده حلقه و ۷) تغییر نادرست متغیرهای حلقه.

طراحی خوب و مناسب حکم می‌کند که شرایط خطا باید پیش‌بینی شوند و مسیرهای کنترل خطا نیز برای مسیریابی مجدد یا پایان دادن به کار پردازش هنگام به‌وجود آمدن خطا تنظیم شده‌اند. یوردان [YOU75][1] این روش را روش ضد خطا[2] نامیده است. متأسفانه گرایشی برای اضافه کردن

1.Yourdon, E.

2.antibugging

مسیر کنترل خطا به نرم‌افزار و نیازمودن آن وجود دارد. استفاده از این گرایش برای اعمال و نشان دادن حالت زیر مورد استفاده قرار می‌گیرد:

طبق قرارداد، یک سیستم اصلی با طراحی محاوره‌ای توسعه می یابد. در یکی از واحدهای پردازش تراکنش ها، برنامهٔ عملی جوکرپیغام ذیل را در ارتباط با کنترل خطا پس از انجام مجموعه‌ای از آزمون‌های شرطی بوجود آورندهٔ شاخه‌های مختلف جریان کنترل، نشان می‌دهد: "خطا! شما هیچ راهی برای رسیدن به این‌جا ندارید." این "پیغام خطا" توسط مشتری در طول آموزش کاربر کشف شده است!

توصیه

اطمینان حاصل نمایید که آزمون طراحی شده شما، هر مسیر رفع خطایی را اجرا نماید. اگر اینگونه نکنید یک مسیر خطا ممکن است فراخوانده شود و منجر به تشدید وضعیت بدی شود.

از میان خطاهای بالقوه‌ای که باید هنگام ارزیابی کنترل خطا می‌توان به موارد زیر اشاره نمود:

۱- توصیف خطا غیرقابل فهم و مبهم می‌باشد.

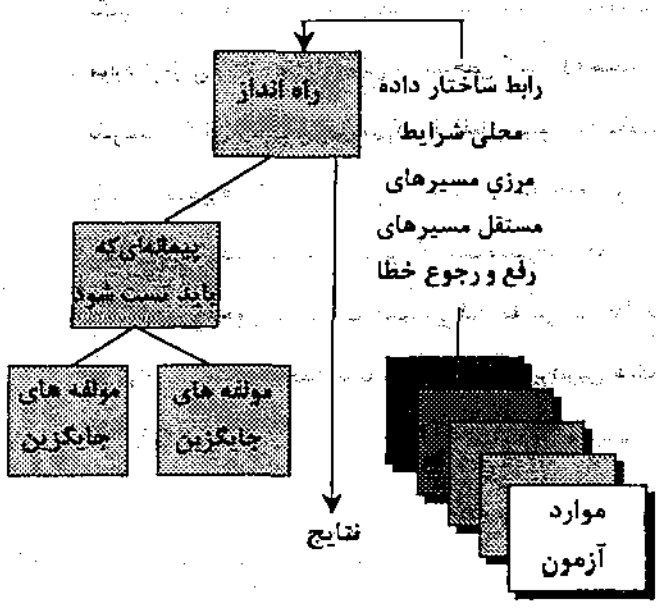
۲- ذکر خطا با مواجه شدن با خطا متفاوت است.

۳- شرایط خطا موجب مداخلهٔ سیستم قبل از کنترل خطا می‌گردد.

۴- پردازش شرایط استثناء نادرست می‌باشد.

۵- توصیف خطا نمی‌تواند اطلاعات کافی برای شناسایی منشاء خطا ارائه کند.

آزمون مرزها آخرین کاری است (و شاید مهم‌ترین کاری است) که باید در مراحل آزمون واحد انجام شود. نرم‌افزارها اغلب در مرزهای خود دچار مشکل و نقص می‌شوند. یعنی خطاها اغلب در هنگام پردازش عنصر n ام آرایهٔ n بعدی به‌وجود می‌آیند؛ یا وقتی که تکرار i ام حلقه، با اجرای i بار تحریک می‌شود این خطاها به‌وجود می‌آیند و یا هنگامی‌که حداکثر یا حداقل مقدار مجاز حاصل می‌شود این خطاها نیز به‌وجود می‌آیند. موارد آزمون، با آزمودن ساختار داده‌ها، جریان کنترل و مقدار داده‌های پایین‌تر از حد کمینه و بیشتر از حد بیشینه، می‌توانند خطاها را تعیین و مشخص کنند.



شکل ۱۸ـ۵ محیط آزمون واحد

۱۸-۳-۲ رویه های آزمون واحد

توصیه

ممکن است در برخی مواقع شما منابع لازم برای آزمون تمام واحدها را در اختیار نداشته باشید. در این حال پیمانه های بحرانی را انتخاب کنید، آنهایی که پیچیدگی سیکلوماتیک دارند، و آزمون واحد را تنها بر آنها انجام

از آن‌جایی‌که یک جزء یک برنامهٔ مستقل نمی‌باشد، بنابراین برنامهٔ رابط یا مؤلفهٔ جایگزین نرم‌افزار باید برای آزمون هر یک از واحدها به‌صورت مجزا ایجاد شود. محیط آزمون واحد در شکل ۱۸-۵ نشان داده شده است. در اکثر نمونه‌های کاربردی، برنامه رابط (راه‌انداز)[1] برنامه‌ای بیش از یک "برنامهٔ اصلی" نمی‌باشد که داده‌های موارد آزمون را می‌پذیرد و این داده‌ها را به جزء انتقال می‌دهد و نتایج مربوطه را چاپ می‌کند. جزءهای جایگزین[2] نرم‌افزار، جایگزین پیمانه‌هایی می‌شوند که این پیمانه‌ها تابع جزءهای در نظر گرفته شده برای آزمون می‌باشند. جزءهای جایگزین یا "برنامه‌های فرعی ساختگی" از رابط پیمانه تابع استفاده می‌کنند، و یا حداقل کاربر روی داده‌ها را انجام می‌دهند، تصدیق ورودی را چاپ می‌کند، و کنترل را به واحد تحت آزمون باز می‌گردانند.

برنامه‌های رابط و جزءهای جایگزین، سربار خواهند بود. یعنی هر دوی این موارد، نرم‌افزاری هستند که باید نوشته شوند، اما این نوشته به محصول نهایی نرم‌افزار ملحق نخواهد شد. اگر برنامه‌های رابط و جزءهای جایگزین به‌صورت ساده نگاه‌داشته شوند، سربار واقعی نسبتاً کم خواهد بود. متأسفانه بسیاری از اجزاء را نمی‌توان با استفاده از نرم‌افزار سربار "ساده" و با استفاده از آزمون واحد، آزمود. در چنین مواردی، آزمون کامل تا زمان تلفیق مراحل آزمون به یکدیگر به تعویق خواهد افتاد. آزمون واحد هنگامی که یک جزء با انسجام بالایی طراحی شده باشد، بسیار آسان خواهد بود. هنگامی‌که فقط یک وظیفه توسط یک جزء مورد بررسی قرار می‌گیرد و انجام می‌شود، تعداد موارد آزمون نیز کاهش می‌یابد و می‌توان خطاها را به آسانی پیش‌بینی و شناسایی نمود.

۱۸-۴ آزمون جامعیت[3]

توصیه

انتخاب رهیافت انفجار عظیم جهت جامعیت، راهبردی کاهل است که محکوم به شکست خواهد بود. آزمون جامعیت لازم است به گونه ای افزایشی انجام شود.

یک فرد نوآموز در دنیای نرم‌افزاری پس از مورد آزمون واقع شدن واحدی تمام پیمانه‌ها ممکن است این سؤال ظاهراً منطقی و به حق را مطرح کنند که: "اگر تمام این واحدها به‌صورت مجزا کار خود را انجام می‌دهند پس چرا شما نسبت به این مطلب شک دارید که این واحدها هنگام قرارگیری در کنار یکدیگر می‌توانند وظایف خود را انجام دهند؟ البته مشکل موجود گذاشتن این واحدها در کنار یکدیگر- یعنی وصل کردن از راه میانجی می‌باشد. در امتداد یک رابط ممکن است بعضی از داده‌ها گم و ناپدید شوند؛ یک پیمانه می‌تواند تأثیر ناخواسته یعنی تأثیر معکوس بر مؤلفهٔ دیگری داشته باشد، عملکردهای فرعی هنگام ترکیب شدن با یکدیگر ممکن است نتوانند عملکرد اصلی و مطلوبی را ایجاد کنند، هر یک از بی‌دقتی‌های

1.driver

2.Stubs

۳. راهبردهای جامعیت و تمامیت برای سیستمهای شئ گرا، در فصل ۲۳ توضیح داده شده اند.

قابل قبول(به طور مستقل) ممکن است به سطوح غیرقابل قبولی تشدید شوند، ساختارهای داده‌های سراسری می‌توانند مشکلاتی را به همراه داشته باشند. و متأسفانه لیست این مشکلات هم‌چنان در حال رشد می‌باشد.

آزمون تلفیقی یک تکنیک نظام مند برای ساختن ساختار برنامه و در عین حال انجام آزمون‌ها برای مشخص شدن خطاهای مربوط به ایجاد رابط می‌باشد. هدف از انجام این آزمون در نظر گرفتن جزءهای آزمون شدهٔ واحد و ایجاد ساختار برنامه‌ای تعیین شدهٔ طراحی می‌باشد.

اغلب تمایل شدیدی برای آزمون تلفیق غیر افزایشی می‌باشد، یعنی ایجاد برنامه با استفاده از یک رهیافت "توسعه بزرگ و یکباره"[1] است. تمام جزءها به شکل پیشرفته‌ای با یکدیگر ترکیب می‌شوند. کل برنامه به‌صورت یک مجموعهٔ کلی آزموده می‌شود، و معمولاً نوعی آشفتگی نیز حاصل می‌شود! در هنگام انجام این آزمون‌ها مجموعه‌ای از خطاهای موجود کشف می‌گردد. و اصلاح این خطاها در این مرحله بسیار مشکل است به‌دلیل آن‌که جداسازی دلایل به‌وجود آورندهٔ این خطاها به‌واسطهٔ گسترده بودن کل برنامه بسیار پیچیده است. با اصلاح شدن این خطاها اخطارهای جدیدی ظاهر می‌شوند و این فرآیند در یک حیطهٔ بی‌انتها ادامه می‌یابد.

تلفیق افزایشی نقطهٔ مقابل روش توسعه یکباره می‌باشد. برنامه در نمودهای کوچک ایجاد شده و مورد آزمون واقع می‌گردد که در این حالت جداسازی خطا و اصلاح آنها بسیار راحت‌تر می‌باشد؛ و احتمال آزمودن کامل رابطها بیشتر می‌باشد، و از روش آزمون نظام مند نیز در این حالت می‌توان استفاده کرد. در بخش‌های بعدی تعدادی از راهبردهای تلفیقی افزایشی مختلف مورد بررسی قرار خواهد گرفت.

۱-۴-۱۸ جامعیت بالا به پایین

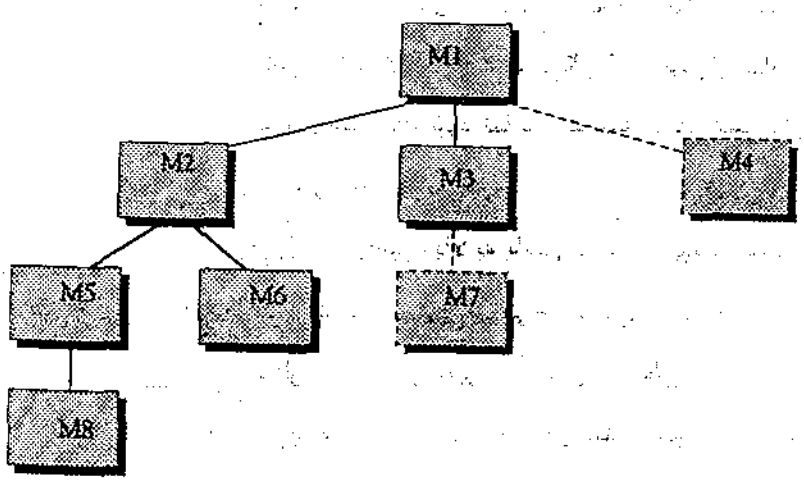
آزمون تلفیق بالا به پایین[2] یک رهیافت افزایشی برای ساخت ساختار برنامه‌ای می‌باشد. پیمانه ها با حرکت کردن به سمت پایین در امتداد سلسله مراتب کنترل که با پیمانه اصلی کنترل آغاز می‌گردد (برنامهٔ اصلی) با یکدیگر تلفیق می‌شوند. جایگزین‌های فرعی پیمانه ها به‌جای پیمانه کنترل اصلی به روش اول عمق یا روش اول عرض، به ساختار اضافه می‌شوند.

توصیه
هنگامی که یک زمان بندی جزئی و مفصل را برای یک پروژه تدارک می بینید، باید به این نکته توجه کنید که کدام ویژگیهای جامعیت اجزاء هنگام نیاز باید در دسترس باشد.

با مراجعه به شکل ۱۸-۶ می‌توان دریافت که در **تلفیق عمقی**[3] در درجهٔ اول تمام جزءها در مسیر اصلی کنترل ساختار با یکدیگر تلفیق می‌شوند. انتخاب مسیر اصلی به‌صورت دلخواه انجام می‌شود و به بکارگیری ویژگی‌ها و مشخصات خاص بستگی دارد. به‌عنوان مثال با انتخاب مسیر دست چپ ابتدا جزءهای M_5, M_2, M_1 با یکدیگر تلفیق خواهند شد. سپس مسیر مرکزی و بعدی مسیر کنترل مرکزی و سمت

1. BigBang Appraoch
2. Top-down integration testing
3. depth-first integration

راست ایجاد می‌شوند. در تلفیق عرضی[1] تمام جزءها در هر سطح مستقیماً یا جزءهای جایگزین تلفیق می‌شوند و به‌صورت افقی در امتداد ساختار به حرکت درمی‌آیند. با توجه به این شکل می‌توان متوجه شد که جزءهای M_2, M_3, M_4 (جایگزین مؤلفهٔ جایگزین S_4) در ابتدا با یکدیگر تلفیق خواهند شد. و سطح بعدی کنترل یعنی M_5 و M_6 با یکدیگر تلفیق می‌گردند.



شکل ۱۸-۶ تلفیق بالا- پایین

فرآیند تلفیق در مجموعه‌ای از پنج مرحله انجام می‌شود که این مراحل عبارتند از:

۱- پیمانه اصلی کنترل به‌عنوان برنامهٔ رابط مورد استفاده قرار می‌گیرد و جزءهای جایگزین نیز در این مرحله تمام جزءهای تابع را جایگزین پیمانه اصلی کنترل می‌کنند.

۲- بسته به روش تلفیقی انتخاب شده جزءهای جایگزینی فرعی فقط یک‌مرتبه جایگزین جزءهای واقعی می‌شوند.

۳- با تلفیق شدن هر یک از جزءها کار آزمودن نیز انجام می‌شود.

۴- بعد از تکمیل شدن هر یک از مجموعه‌های در نظر گرفته شده برای آزمون، مؤلفهٔ جایگزین دیگری جایگزین مؤلفهٔ واقعی می‌شود.

۵- آزمون رگرسیون (بخش ۱۸-۴-۳) را می‌توان برای مطمئن شدن از این امر که خطای جدید به‌وجود نیامده است، انجام داد.

این فرآیند از مرحلهٔ دوم ادامه می‌یابد تا آن‌که کل برنامه ساخته شود.

راهبرد تلفیق بالا به پایین[2] نقاط اصلی کنترل یا تصمیم‌گیری را در ابتدای فرآیند آزمون صدیق و تأیید می‌کند. در ساختار برنامهٔ خوب تقسیم بندی شده، تصمیم‌گیری در سطوح بالاتر سلسله مراتب موجود انجام می‌شود و بنابراین در ابتدا و در مراحل اولیهٔ مسأله تصمیم‌گیری وجود خواهد داشت. اگر

پرسش

جامعیت و انسجام بالا به پایین مترتب بر کدام گامها است؟

ارجاع به فصلها

فاکتورگیری برای سبک معماری متمرکز، از اهمیت برخوردار است. فصل ۱۴ را برای جزئیات بیشتر نگاه کنید.

1.Breadth-first integration

2.bottom-up testing

مشکلات اصلی کنترل در هنگام تصمیم‌گیری وجود داشته باشد. شناسایی به‌موقع این مشکلات (و یافتن راه‌حلی برای آنها) امری ضروری است. در صورت انتخاب کردن تلفیق عمق در درجهٔ اول وظیفهٔ کامل نرم‌افزار باید طرح‌ریزی و اجرا شود. به‌عنوان مثال یک ساختار سنتی و کلاسیک تراکنش (فصل ۱۴) را در نظر بگیرید که در آن ساختار مجموعهٔ پیچیده‌ای از ورودی‌های محاوره‌ای پیشنهاد شده، بدست‌آمده و از طریق مسیر درآمدی مورد تأیید و تصدیق قرار می‌گیرد. ممکن است مسیرهای درآمدی به روش بالا به پایین با یکدیگر تلفیق شوند. ممکن است پردازش تمام داده‌های ورودی قبل از تلفیق شدن سایر اجزاء ساختار نشان داده شود. نمایش اولیهٔ قابلیت‌های عملی (کاری) یک سازندهٔ مطمئن برای فرد توسعه‌دهنده و مشتری محسوب می‌شود.

راهبرد بالا به پایین نسبتاً غیر پیچیده به‌نظر می‌رسد، اما در عمل در این روش ممکن است مشکلات لوجستیکی (منطقی-استدلالی) پدید آید. شایع‌ترین این مشکلات هنگامی رخ می‌دهد که کار پردازش در سطوح پایین به آزمودن در سطوح بالاتر نیاز داشته باشد. جزءهای جایگزین در ابتدای آزمون بالا به پایین جایگزین پیمانه های سطح پایین می‌شوند؛ بنابراین هیچ‌گونه اطلاعاتی نمی‌تواند در ساختار برنامه به سمت بالا حرکت کند. فرد آزمون‌کننده می‌تواند یکی از سه روش زیر را برای کار آزمون انتخاب کند: ۱) به تأخیر انداختن بسیاری از آزمون‌ها تا زمانی‌که جزءهای جایگزین بتوانند جایگزین پیمانه های واقعی شوند. ۲) تکمیل نمودن جزءهای جایگزینی که وظایف محدودی را انجام می‌دهند که در این وظایف پیمانه های واقعی شبیه‌سازی می‌شوند. ۳) تلفیق نرم‌افزار از پایین به سمت بخش‌های بالایی سلسله مراتب موجود.

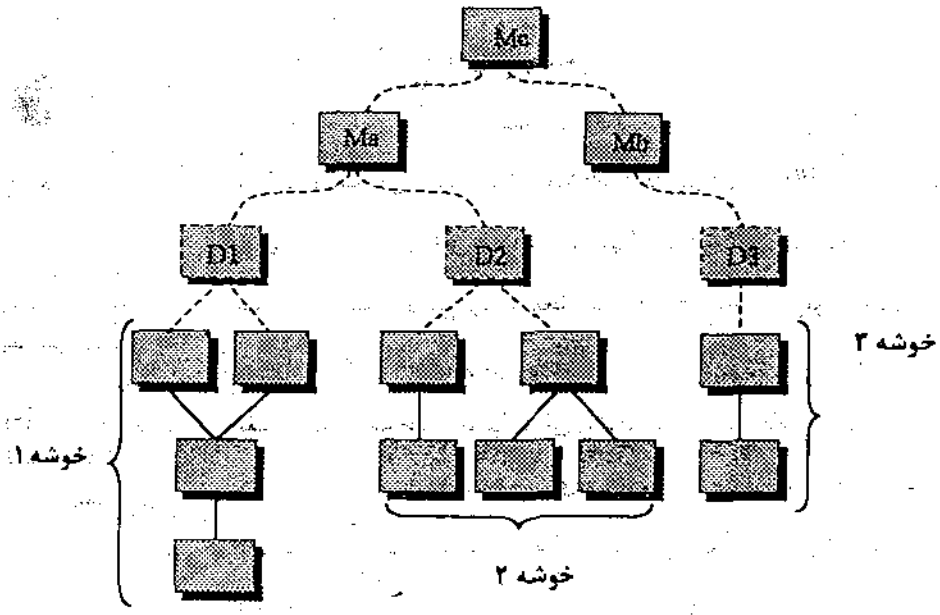
روش اول (یعنی به تأخیر انداختن آزمون‌ها تا زمان جایگزین شدن جزءهای جایگزین به‌جای پیمانه های واقعی) موجب می‌شود که ما نتوانیم بر تناظر موجود میان آزمون‌های خاص و مشارکت پیمانه های خاص کنترل داشته باشیم. و این امر به بوجود آمدن مشکل در تعیین علت خطا منتهی می‌شود و از طبیعت محدود شدهٔ روش بالا به پایین تخطی خواهد کرد. روش دوم قابل اجرا و عملی می‌باشد. اما این روش نیز ممکن است به اتلاف و سربار قابل ملاحظه‌ای منتهی شود چرا که جزءهای جایگزین در این روش پیچیده و پیچیده‌تر می‌شوند. روش سوم با نام آزمون پایین به بالا در بخش بعدی مورد بررسی قرار خواهد گرفت.

۱۸-۴-۲ جامعیت پایین به بالا

آزمون تلفیقی پایین به بالا همان‌طور که از نامش پیداست ساخت و آزمودن را با پیمانه های اتمیک[1] (یعنی جزءهای موجود در پایین‌ترین سطح در ساختار برنامه) آغاز می‌کند. از آن‌جایی‌که جزءها از پایین به

1.atomic modules

بالا با یکدیگر تلفیق می‌شوند. بنابراین پردازش مورد نیاز برای جزءهای تابع تا رسیدن به سطح مفروض همواره موجود می‌باشد و نیاز موجود برای وجود جزءهای جایگزین برطرف شده است.



شکل ۱۸_۷ تلفیق پایین ـ بالا

گامهای جامعیت پایین به بالا کدامهاست؟

راهبرد تلفیق پایین به بالا ممکن است طبق مراحل زیر اجرا شود:

۱- جزءهای سطح پایین با یکدیگر ترکیب می‌شوند و خوشه‌هایی را به‌وجود می‌آورند (که گاهی سازندگان نامیده می‌شود)[1] این خوشه‌ها نیز کارکرد فرعی و خاص نرم‌افزاری را اجرا می‌کنند.

۲- برنامهٔ رابط (برنامهٔ کنترل برای کار آزمون) برای هماهنگ کردن ورودی و خروجی موارد آزمون نوشته و تدوین می‌شود.

۳- خوشه آزموده می‌شود.

۴- برنامه‌های رابط کنار گذاشته می‌شوند و خوشه‌ها هنگام حرکت به سمت بالا در ساختار برنامه با یکدیگر ترکیب می‌شوند.

تلفیق ذکر شده در این قسمت، از الگوی نشان داده شده در شکل ۱۸-۷ تبعیت می‌کند.

جامعیت پایین به بالا، نیاز به روالهای غیر اجرایی (STUB) پیچیده را مرتفع می سازد.

جزءها با یکدیگر ترکیب می‌شوند تا خوشه‌های ۱ و ۲ و ۳ را به‌وجود آورند. هر یک از این خوشه‌ها با استفاده از یک برنامهٔ رابط آزموده می‌شوند. جزءهای موجود در خوشه‌های ۱ و ۲ تابع M_a هستند. هم‌چنین برنامهٔ رابط D_3 برای خوشهٔ شماره ۲ نیز قبل از تلفیق با پیمانه M_b کنار گذاشته می‌شود. M_a و M_b فردی در نهایت با مؤلفهٔ M_3 تلفیق می‌شوند. با حرکت تلفیق جزءها به سمت بالا از ضرورت موجود برنامه‌های رابط مجزا برای انجام آزمون‌ها کاسته می‌شود. در حقیقت، اگر دو سطح بالا از ساختار برنامه‌ها از

بالا به پایین با یکدیگر تلفیق شوند، تعداد برنامه‌های رابط را می‌توان کاهش داد و تلفیق خوشه‌ها نیز بسیار ساده‌تر می‌شود.

۱۸-۴-۳ آزمون رگرسیون

توصیه

آزمون رگرسیون یک راهبرد مهم برای کاهش " پیامد ها " ست. آزمون های رگرسیون هر زمان که یک تغییر عمده در ساخت نرم افزار به وجود اید، عمل می کنند. (مشتمل بر جامعیت پیمانه های جدید)

هر مرتبه که پیمانه جدیدی به‌عنوان بخشی از آزمون تلفیقی اضافه می‌شود، نرم‌افزار تغییر پیدا می‌کند. مسیرهای جدید جریان اطلاعات ایجاد می‌شوند و I/O جدید نیز ممکن است به‌وجود آید و منطق جدید کنترل نیز در این شرایط ایجاد می‌گردد. این تغییرات ممکن است مشکلاتی را در کارکردهایی که قبلاً بدون نقص کار می‌کردند، ایجاد کند. در بافت راهبرد آزمون تلفیقی، آزمون رگرسیون[2] اجرای مجدد برخی از مجموعه‌های فرعی آزمون‌ها می‌باشد، آزمونهایی که قبلاً برای مطمئن شدن از این امر که تغییرات، تأثیرات جانبی ناخواسته‌ای را ایجاد نکرده‌اند، انجام شده‌اند.

در بافت گسترده‌تر، آزمون‌های مؤفق موجب کشف شدن خطاها شده‌اند و این خطاها باید اصلاح و تصحیح شوند. هر دفعه که نرم‌افزاری اصلاح و تصحیح می‌شود، برخی از جنبه‌های مربوط به پیکربندی نرم‌افزار نیز تغییر پیدا می‌کند. آزمون رگرسیون فعالیتی است که به ما کمک می‌کند تا مطمئن شویم که تغییرات (به‌واسطهٔ انجام آزمون‌ها یا به‌واسطهٔ سایر دلایل موجود) رفتار ناخواسته یا خطای اضافی را ایجاد نکرده‌اند.

آزمون رگرسیون ممکن است به صورت دستی انجام شود که در این حالت از مجموعهٔ فرعی تمام موارد آزمون و یا از ابزار گردآوری ـ عقبگرد[3] استفاده می‌شود. ابزارهای گردآوری ـ عقبگرد مهندسین نرم‌افزار را قادر می‌سازند ـ موارد آزمون شدنی و نتایج را گردآوری کنند و برای مقایسه‌ها و خواندن مجدد از آنها استفاده کنند.

مجموعهٔ برنامه‌های کاربردی آزمون رگرسیون (مجموعهٔ فرعی برنامه‌هایی که باید اجرا شوند) دارای سه کلاس متفاوت موارد آزمون می‌باشند که این سه نوع عبارتند از:

- نمونهٔ بازنمای آزمون‌ها که تمام کارکردهای نرم‌افزاری را می آزماید.
- آزمون‌های اضافی که بر آن گروه از کارکردهای نرم‌افزاری تاکید دارند که احتمالاً تحت تأثیر تغییرات انجام شده قرار می‌گیرند
- آزمون‌هایی که بر جزءهای نرم‌افزاری تغییر یافته تأکید دارند.

با انجام مراحل مختلف آزمون تلفیقی، تعداد آزمون‌های رگرسیون نیز به‌طور قابل ملاحظه‌ای افزایش می‌یابد. بنابراین مجموعهٔ برنامه‌های کاربردی آزمون رگرسیون باید به‌گونه‌ای طراحی شود که فقط آن گروه

1.builds

2.regression testing

3.capture/playback tools

از آزمون‌هایی را در بربگیرد که یک کلاس از خطاها یا چند کلاس از خطاها را در هر یک از کارکردهای اصلی برنامه مورد بررسی قرار می‌دهند. انجام مجدد هر یک از آزمون‌ها برای هر یک از کارکردهای برنامه هنگام به‌وجود آمدن تغییر، غیر عملی و غیر مفید می‌باشد.

۱۸-۴-۴ آزمون دود

آزمون دود[1] یک رهیافت آزمون تلفیقی است که اغلب در مواردی استفاده می‌شود که یک محصول نرم‌افزاری جمع و جور شده، در حال تکمیل شدن و توسعه یافتن باشد. این آزمون به‌عنوان مکانیسم اندازه‌گیری برای پروژه‌های بحرانی از لحاظ زمان طراحی شده است و امکان ارزیابی پروژه بر مبنای تکرار را برای تیم نرم‌افزاری فراهم می‌کند. در موارد ضروری، رهیافت آزمون دود فعالیت‌های زیر را پوشش می‌دهد:

آزمون دود، می تواند به عنوان راهبرد چرخشی جامعیت قلمداد شود. (طی آن) نرم افزار هر روز با اجزاء جدیدش مجددا" ساخته و آزمایش می شود

۱- جزءهای نرم‌افزاری تبدیل شده به برنامه که برای تشکیل یک ساختمان با یکدیگر تلفیق می‌شوند. این ساختمان شامل فایل تمام داده‌ها، کتابخانه‌ها، پیمانه هایی با قابلیت استفاده مجدد و شامل جزءهای طراحی شدهٔ لازم برای اجرای یک یا چند کارکرد محصول می‌باشد.

۲- مجموعه‌ای از آزمون‌ها که برای شناسایی خطاهایی طراحی شده‌اند که این خطاها مانع از آن می‌شوند که این ساختمان وظایف خود را به طرز صحیحی انجام دهد. هدف از این آزمون‌ها باید شناسایی خطاهای "متوقف‌کننده نمایش" باشد که این خطاها بیشترین احتمال را برای به تعویق انداختن برنامهٔ پروژه دارا می‌باشند.

۳- این ساختمان با سایر ساختمان‌ها تلفیق می‌شوند و در کل محصول آزمون دود به‌صورت روزانه انجام می‌شود. رهیافت تلفیقی ذکر شده در این قسمت، ممکن است بالا به پایین و یا پایین به بالا باشد.

تکرار روزانهٔ آزمون مربوط به کل محصول ممکن است باعث تعجب برخی از خوانندگان شود. با این وجود انجام آزمون‌های مکرر و پی در پی ارزیابی واقعی از پیشرفت آزمون تلفیقی را در اختیار مدیران و کارورزان قرار می‌دهد. مک‌کانل[MCO96][2] آزمون دود را به شکل ذیل تعریف می‌کند:

"آزمون دود باید به آزمودن کل سیستم بپردازد. این آزمون نباید کامل و جامع باشد، اما باید بتواند مشکلات اصلی و عمده را مشخص کند. آزمون دود باید به اندازهٔ کافی کامل باشد که اگر ساختمان در حال عبور کردن باشد شما بتوانید فرض کنید که این ساختمان به اندازهٔ کافی ثابت است و می‌توان آزمون دود را به‌طور کامل بر روی آن انجام داد."

1.Smoke testing

2.McConnell, S.

آزمون دود هنگام استفاده در پروژه‌های پیچیده و حساس به زمان مهندسی نرم‌افزاری مزیت‌های زیر را در برخواهد داشت:

- **ریسک تلفیق به حداقل می‌رسد.**[1] از آن‌جایی‌که آزمون دود به‌صورت روزانه انجام می‌شود بنابراین ناهمخوانی و سایر خطاهای متوقف‌کنندهٔ نمایش زودتر تشخیص داده می‌شوند و بدین‌ترتیب احتمال تأثیرگذاری جدی و شدید آنها بر روی زمان‌بندی، هنگام شناسایی خطاها کاهش می‌یابد.
- **کیفیت محصول نهایی بهبود می‌یابد.**[2] از آن‌جایی‌که این آزمون یک روش مبتنی بر ساختمان است. بنابراین می‌تواند خطاهای کاری و هم‌چنین نواقص طراحی در سطح جزءها و نواقص معماری را شناسایی کند. اگر این معایب و نواقص زودتر اصلاح شوند محصول با کیفیت‌تری نیز به‌وجود خواهد آمد.
- **تشخیص خطا و تصحیح خطا آسان‌تر و ساده‌تر می‌شود.**[3] خطاهای شناسایی شده در طول آزمون دود نیز همانند خطاهای شناسایی شده از طریق تمام روش‌های آزمون تلفیقی احتمالاً با افزودنی‌های جدید نرم‌افزار در ارتباط می‌باشند ـ یعنی نرم‌افزار اضافه شده به ساختمان، عامل احتمالی خطای جدید کشف شده می‌باشد.
- **پیشرفت برای انجام ارزیابی آسان‌تر است.**[4] هر روز که سپری می‌شود، نرم‌افزارهای بیشتری با یکدیگر تلفیق می‌شوند و این امر علامت مناسبی مبنی بر به وجود آمدن پیشرفت در کارهای نرم‌افزاری را در اختیار مدیران قرار می‌دهد.

نقل قول

توسعه روزانه را به عنوان تپش قلب یک پروژه قلمداد کنید. اگر تپشی نباشد، پروژه مرده است. جیم مک کارتی

۱۸-۴-۵ توضیحاتی بر آزمون جامعیت

در ارتباط با مزیت‌ها و معایب نسبی آزمون تلفیقی بالا به پایین[BEI84][5] در مقایسه با آزمون پایین به بالا بحث‌های زیادی وجود دارد. به‌طور کلی مزیت‌های یک راهبرد معمولاً معایبی را برای راهبرد دیگر به‌وجود می‌آورد. نقطه ضعف (عیب) اصلی روش تلفیقی بالا به پایین آن است که این روش به جزءهای جایگزین نیاز دارد و مشکلات خاص آزمون نیز در ارتباط با این جزءها وجود دارد. مشکلات مربوط به جزءهای جایگزین را می‌توان با مزیت آزمودن وظایف اصلی کنترل در مراحل اولیه برطرف و جبران نمود. عیب اصلی روش تلفیقی پایین به بالا آن است که "این برنامه به‌عنوان یک موجودیت تا زمان اضافه

یک پیمانه بحرانی چیست و چرا باید آن را بشناسیم؟

1.Integration risk is minimized

2.The quality of the end-product is improved

3.Error diagnosis and correction are simplified

4.Progress is easier to assess

5.Beizer,B.

شدن آخرین پیمانه وجود نخواهد داشت." این نقطه ضعف با طراحی آسان‌تر موارد آزمودنی و عدم وجود جزءهای جایگزین تعدیل می‌شود.[MYE79] [1]

انتخاب راهبرد تلفیقی به ویژگی‌های نرم‌افزار و در بعضی موارد به جدول زمانی پروژه بستگی دارد. به‌طور کلی یک روش ترکیبی (که در بعضی موارد آزمون ساندویچی[2] نیز نامیده می‌شود) که برای سطوح بالای ساختار برنامه از روش تلفیقی بالا به پایین استفاده می‌کند با روش پایین به بالا برای سطوح تابع که بهترین راه‌حل خواهد بود، جفت می‌شود.

هنگام انجام آزمون تلفیقی، فرد آزمون‌کننده باید پیمانه های حساس و بحرانی[3] را شناسایی کند. یک پیمانه بحرانی یک یا چند عدد از ویژگی‌های زیر را دارا می‌باشد: ۱) چندین عدد از شرایط و ویژگی‌های نرم‌افزاری را مورد بررسی قرار می‌دهد؛ ۲) دارای کنترل در سطح بالا می‌باشد. ۳) پیچیده یا مستعد خطا می‌باشد؛ و یا ۴) دارای شرایط و ویژگی‌های عملکردی محدود و مشخص می‌باشد. پیمانه های بحرانی باید تا حد ممکن فی‌الفور مورد آزمون قرار گیرند. به‌علاوه آزمون‌های رگرسیون باید بر وظایف پیمانه بحرانی تأکید داشته باشند.

۱۸-۵ آزمون اعتبارسنجی

مانند دیگر گامهای آزمون، اعتبارسنجی به دنبال خطاهای پوشش نیافته است. اما با تمرکز بر سطح نیازمندیها به دنبال چیزهایی است که مستقیما" با کاربر نهایی سرو کار دارند

در نتیجهٔ نهایی آزمون تلفیقی، نرم‌افزار به‌صورت کامل به شکل بسته‌ای مونتاژ می‌شود، در این مرحله خطاهای عمل‌کننده به‌عنوان رابط شناسایی و تصحیح شده‌اند و مجموعه‌های نهایی نرم‌افزار آزمون شده‌اند و ممکن است آزمون اعتبارسنجی یا تصدیق[4] آغاز شود. تصدیق را می‌توان به اشکال مختلف تعریف نمود، اما تعریف سادهٔ آن عبارت است از این‌که تصدیق و تایید اعتبازی، زمانی حاصل می‌شود که وظایف انجام شده توسط نرم‌افزار به‌صورت منطقی مورد قبول و مطابق با انتظارات مشتری باشد. در این مرحله فرد تکمیل‌کنندهٔ نرم‌افزار می‌تواند به این مسأله اعتراض کند که "چه کسی یا چه چیزی تعیین‌کنندهٔ انتظارات معقول و منطقی می‌باشد؟"

انتظارات منطقی در مشخصات نیازمندیهای نرم‌افزاری[5] - یعنی در سندی که توصیف‌کنندهٔ تمام ویژگی‌های آشکار نرم‌افزار می‌باشد - تعریف شده است. در این مشخصات بخشی بنام معیار تصدیق[6] گنجانده شده است. اطلاعات موجود در این بخش مبنای روش آزمون تصدیق را تشکیل می‌دهند.

1.Myers,G.

2.sandwich testing

3.critical modules

4.validation testing

5.Software Requirements Specification

6.Validation Criteria

۱۸-۵-۱ معیار آزمون اعتبارسنجی

تصدیق و تأیید نرم‌افزار از طریق مجموعه‌ای از آزمونهای جعبهٔ سیاه که نشان‌دهندهٔ مطابقت با شرایط می‌باشد، حاصل می‌شود. در طرح آزمون انواع آزمون‌هایی که قرار است انجام شود به‌صورت خلاصه بیان شده و رویه ای برای آزمون همه موارد آزمودنی تعریف خواهد شد. مواردی که تطابق با شرایط مورد استفاده و خواسته ها در آن لحاظ شده باشند. طرح و روش هر دو برای مطمئن شدن از این مطلب طراحی شده‌اند که تمام شرایط کاری مطلوب و رضایت‌بخش هستند؛ تمام ویژگی‌های رفتاری حاصل شده‌اند و تمام نیازمندیهای عملکردی ایجاد شده‌اند، مستندسازی درست است و سایر نیازمندیها (از جمله قابلیت حمل، قابلیت سازگاری، کشف خطا، قابلیت حفظ و نگهداری) نیز تأمین شده‌اند.

بعد از انجام هر یک از موارد آزمون تصدیقی یکی از دو شرایط احتمالی ذیل وجود خواهد داشت:

۱- ویژگی‌های کارکرد یا عملکرد با مشخصات مفروض هماهنگی داشته و مورد قبول می‌باشند. ۲) انحراف از مشخصات تعیین شده است و لیستی از نواقص و معایب[1] موجود نیز ایجاد شده است. انحراف یا خطای کشف شده در این مرحله از پروژه، بندرت قابل اصلاح قبل از تحویل زمان‌بندی شده می‌باشد. برای ایجاد یک شیوه برای برطرف نمودن معایب و نواقص موجود لازم است مذاکراتی با مشتریان انجام شود.

۱۸-۵-۲ بازبینی پیکربندی

یکی از اجزاء مهم فرایند تصدیق، بازبینی پیکربندی[2] نرم‌افزار می‌باشد. هدف از این بازبینی مطمئن شدن از این امر است که تمام اجزاء پیکربندی نرم‌افزار به طرز صحیحی تکامل و توسعه یافته‌اند، به طرز صحیحی فهرست‌بندی شده‌اند و برای ایجاد مرحلهٔ حمایت از طول عمر نرم‌افزار،

ویژگی‌های لازم را دارا می‌باشند. در بعضی موارد بازبینی پیکربندی نرم‌افزار، وارسی[3] پیکربندی نامیده می‌شود که این موضوع به‌طور کامل و مفصل در فصل ۹ بررسی گردیده است.

۱۸-۵-۳ آزمون الف و ب (آلفا و بتا)

پیش‌بینی نحوهٔ استفادهٔ واقعی از نرم‌افزار توسط مشتری، برای فرد توسعه‌دهنده و تکمیل‌کنندهٔ نرم‌افزار غیرممکن می‌باشد. دستورالعمل‌های ارائه شده برای مصرف نرم‌افزار ممکن است به‌صورت نادرست تعبیر و تفسیر شوند، ترکیبات عجیب داده‌ها ممکن است توسط مشتری مورد استفاده قرار بگیرد، خروجی واضح و آشکار از نظر فرد آزمون‌کننده ممکن است از نظر مصرف‌کنندهٔ این خروجی غیرقابل فهم و مبهم باشد.

1.deficiency list

2.configuration review

3.audit

هنگامی‌که یک نرم‌افزار سفارشی برای یک مشتری ساخته می‌شود، مجموعه‌ای از آزمون‌های پذیرش[1] نیز انجام می‌شود تا مشتری بتواند تمام شرایط موجود در نرم‌افزار را تصدیق و تأیید کند. آزمون پذیرش که توسط کاربر نهایی انجام می‌شود و نه مهندسین نرم‌افزار، می‌تواند از آزمون رسمی برنامهٔ رایط تا مجموعه‌ای از آزمون‌های طرح‌ریزی شده و نظام مند متفاوت باشد. در حقیقت آزمون پذیرش را می‌توان در دوره‌های زمانی هفتگی یا ماهانه انجام داد و بدین‌ترتیب امکان کشف خطاهای انباشته شده که موجب از کار افتادن سیستم در طول زمان می‌شوند، میسر می‌گردد.

اگر نرم‌افزاری برای استفادهٔ تعداد زیادی از مشتریان به‌وجود آمده و تکمیل شده است، بنابراین انجام آزمون‌های رسمی پذیرش در ارتباط با هر یک از مشتریان غیرممکن و غیر عملی می‌باشد. بسیاری از سازندگان محصولات نرم‌افزاری از فرایندی به نام آزمون (الف و ب) آلفا و بتا استفاده می‌کنند تا خطاهایی را شناسایی کنند که فقط کاربر نهایی قادر به شناسایی آنها می‌باشد.

آزمون (الف)[2] یا آلفا در محل توسعه و تکمیل نرم‌افزار توسط مشتری انجام می‌شود. نرم‌افزار در شرایط زمانی و مکانی طبیعی مورد استفاده قرار می‌گیرد که در این حالت فرد تکمیل‌کنندهٔ نرم‌افزار به آزمودن نرم‌افزار در حضور مشتری می‌پردازد و خطاها و مشکلات کاربرد را ثبت می‌کند. آزمون‌های آلفا در محیطی کنترل شده انجام می‌شوند.

آزمون (ب)[3] یا بتا در محل مشتری و توسط کاربر نهایی نرم‌افزار انجام می‌شود. در این آزمون بر خلاف آلفا، فرد تکمیل‌کنندهٔ نرم‌افزار به‌طور کلی حاضر نمی‌باشد (حضور ندارد). بنابراین آزمون بتا کاربرد "زندهٔ" نرم‌افزار در محیطی می‌باشد که شخص تکمیل‌کنندهٔ نرم‌افزار، کنترلی بر آن محیط ندارد. مشتری تمام مشکلات (اعم از مشکلات واقعی و فرضی) که در طول آزمون بتا با آنها مواجه شده است را ثبت می‌کند و این مشکلات را در فواصل زمانی منظم و مرتب به فرد تکمیل‌کنندهٔ نرم‌افزار گزارش می‌دهد. در نتیجهٔ مشکلات ثبت شده در طول آزمون بتا، مهندسین نرم‌افزار تغییرات و اصلاحاتی را در نرم‌افزار ایجاد می کنند و سپس نرم‌افزار نهایی را به مشتریان ارائه می‌کنند.

۱۸-۶ آزمون سیستم

نقل قول

همانند مرگ و مالیات، آزمون نیز ناخوشایند است و البته اجتناب ناپذیر می‌نماید. اد یوردان

در آغاز این کتاب ما بر این حقیقت تأکید کرده‌ایم که نرم‌افزار فقط جزیی از سیستم بزرگتر کامپیوتری می‌باشد. و نهایتاً آن‌که نرم‌افزار به سایر اجزاء سیستم (از جمله سخت‌افزار، افراد، اطلاعات) ملحق می‌شود و مجموعه‌ای از آزمون‌های تلفیق سیستم و آزمون‌های تصدیق نیز انجام می‌شوند. این آزمون‌ها در خارج از حیطهٔ فرآیند (پردازش) نرم‌افزاری قرار دارند و فقط توسط مهندسین نرم‌افزار انجام

1.acceptance
2.alpha test
3.beta test

نمی‌شوند. با این وجود مراحل طی شده در طول طراحی نرم‌افزار و آزمون نرم‌افزار می‌تواند احتمال تلفیق مؤفق نرم‌افزاری را در یک سیستم بزرگتر بهبود بخشند.

مشکل کلاسیک سیستم آزمونی "مقصر دانستن کامل دیگری در بروز مشکل" می‌باشد. این مشکل هنگام کشف شدن خطا به‌وجود می‌آید و هر یک از اجزاء سیستم تکمیل‌کننده، جزء دیگری را مقصر به‌وجود آمدن این مشکل می‌دانند. مهندسین نرم افزار به‌جای افراط کردن در چنین موارد و مشکلات بی‌خودی باید به مشکلات بالقوهٔ رابطها بپردازند و (۱) مسیرهای کنترل خطاها که تمام اطلاعات وارد شده از سایر اجزاء سیستم را آزمون می‌کنند، طراحی نمایند؛ (۲) مجموعه‌ای از آزمون‌های شبیه‌سازی‌کنندهٔ داده‌های نامناسب یا آزمون‌های مربوط به سایر خطاهای احتمالی در رابط نرم‌افزاری را انجام دهند. (۳) نتایج آزمون‌های انجام شده را ثبت کنند تا از آن به‌عنوان دلیلی در صورت به‌وجود آمدن مشکل مقصر دانستن عامل دیگر در بروز مشکل، استفاده نمایند. (۴) در طرح‌ریزی و برنامه‌ریزی و طراحی آزمون‌های سیستم شرکت کنند تا مطمئن شوند که نرم‌افزار به طرز صحیح و به‌اندازهٔ کافی آزمون شده است.

آزمون سیستم در واقع مجموعه‌ای از آزمون‌های مختلف می‌باشد که هدف اولیهٔ آنها آزمون سیستم کامپیوتری می‌باشد. اگرچه هر یک از این آزمون‌ها اهداف مختلف و متفاوتی را دارند، اما تمام آنها برای تصدیق تلفیق درست اجزای سیستم با یکدیگر و تصدیق عملکرد تعیین شده در وظایف انجام می‌شوند. در بخش‌های بعدی در مورد انواع آزمون‌های سیستم که برای سیستم‌های نرم‌افزاری مفید هستند، به بحث می‌پردازیم [BEI84] [1].

۱۸-۶-۱ آزمون بازیابی

بسیاری از سیستم‌های کامپیوتری باید از معایب موجود در کامپیوتر خلاص شوند و پردازش را در محدودهٔ زمانی از پیش تعیین شده ادامه دهند. در بسیاری از موارد، سیستم باید مشکل موجود را تحمل کند، یعنی پردازش خطاها نباید موجب متوقف شدن کار کل سیستم شود. در سایر موارد، مشکل سیستم باید در دورهٔ زمانی مشخص شده اصلاح شود، در غیر این‌صورت خسارت اقتصادی شدیدی به‌وجود خواهد آمد.

آزمون بازیابی یا ترمیمی[2]، یک آزمون سیستمی است که نرم‌افزار را وادار می‌کند تا به طرق مختلف خراب شود و تصدیق می‌کند که ترمیم و بهبودی به طرز صحیحی انجام شده است. اگر بهبود خودکار باشد، صحت شروع مجدد صحت مکانیسم‌های کل بازرسی، صحت بهبود داده‌ها و صحت شروع مجدد مورد ارزیابی قرار خواهد گرفت. اگر بهبود به مداخلات انسانی نیاز داشته باشد، میانگین زمان برای ترمیم، مورد ارزیابی قرار می‌گیرد تا مشخص شود که آیا زمان این کار (تعمیر) در محدودهٔ قابل قبول می‌باشد یا خیر؟

1.Beizer,B.

2.Recovery testing

۱۸-۶-۲ آزمون امنیتی

هر یک از سیستم‌های کامپیوتری کنترل‌کننده اطلاعات حساس یا قادر به انجام انجام اقداماتی که به خطر انداختن منافع افراد را به دنبال دارد، هدف کاوش و نفوذ غیرقانونی و نادرست می‌باشد. کاوش و نفوذ در محدودهٔ

گسترده‌ای از فعالیت‌های مختلف از جمع موارد ذیل مشاهده شده است: مزاحمان کامپیوتری که سعی دارند در زمینهٔ ورزشی در سیستم نفوذ کنند؛ کارمندان ناراضی که سعی دارند در ارتباط با حقوق خود در سیستم نفوذ کنند؛ افراد نامطمئن که سعی دارند برای دستیابی به سود شخصی غیرقانونی در سیستم نفوذ کنند.

آزمون امنیتی[1] سعی دارد به تصدیق و تأیید این امر بپردازد که مکانیسم‌های حفاظتی ایجاد شده در سیستم در حقیقت از نفوذ غیرقانونی افراد جلوگیری می‌کنند و از سیستم محافظت می نمایند. طبق اظهارات بیزر[BEI84][2]: "البته امنیت سیستم باید در زمینهٔ مصون بودن سیستم از حملهٔ ناگهانی آزمون شود ـ اما امنیت سیستم باید در زمینهٔ مصون بودن سیستم در برابر حملات بعدی و پشت جبهه‌ای نیز آزمون گردد."

در طول انجام آزمون امنیت، فرد آزمون‌کننده نقش افرادی را بازی می‌کند که مایل هستند در سیستم نفوذ کنند. در این آزمون فرد آزمون‌کننده سعی می‌کند اسم رمزها را از طریق ابزار خارجی و دفتری کشف کند، ممکن است با نرم‌افزارهای سفارشی طراحی شده برای از کار انداختن سپرهای دفاعی ساخته شده در سیستم اقدام کند، ممکن است سیستم را از هم بپاشد و بدین‌ترتیب مانع ارائه خدمات به سایر افراد شود، یا از روی عمد خطاهایی را در سیستم ایجاد کند به این امید که در طول دورهٔ بهبودی بتواند در سیستم نفوذ کند، ممکن است داده‌های غیر امنیتی را کاوش و بررسی کند به این امید که راه‌حل‌های کلیدی برای نفوذ در سیستم بیابد.

با ارائهٔ دقت کافی و منابع کافی، آزمون امنیتی، مناسب در نهایت مانع نفوذ در سیستم می‌شود. نقش طراح سیستم در این زمینه گران ساختن هزینهٔ نفوذ در مقایسه با ارزش اطلاعاتی حاصل از نفوذ، می‌باشد.

نقل قول

اگر شما سعی بر یافتن اشکالات پنهان سیستم دارید و نرم افزار را در آزمون فشار قرار نمی دهید، زمان بسیاری را باید صرف نمایید.

بوریس بیزر

۱۸-۶-۳ آزمون فشار

در طول مراحل ابتدایی‌تر آزمون نرم‌افزاری، فنون جعبهٔ سفید و جعبهٔ سیاه موجب به‌وجود آمدن ارزیابی عملکرد برنامه‌های عادی شده‌اند. آزمون‌های فشار برای مواجه ساختن برنامه‌ها با موقعیت‌های غیر

1.Security testing

2.Beizer,B.

عادی طراحی می‌شوند. در صورت لزوم فرد آزمون‌کننده که انجام‌دهندهٔ آزمون فشار است می‌تواند این سؤال را مطرح سازد که: "چگونه می‌توان این آزمون را قبل ازآنکه سیستم از کاربیفتد، اعمال نمود؟"

آزمون فشار[1] سیستم را به شکلی اجرا می کند که سیستم در آن حالت به منابع در مقادیر، دفعات یا حجم غیرعادی نیاز خواهد داشت. به‌عنوان مثال (۱) آزمون‌های خاصی ممکن است طراحی شده باشند که این آزمون‌ها در هر ثانیه ۱۰ وقفه ایجاد کنند در حالی‌که تعداد وقفه کمّ مقدار میانگین یک یا دو عدد می‌باشد؛ (۲) میزان داده‌های ورودی ممکن است به‌ترتیب مقدار افزایش داده شود تا مشخص گردد که عکس‌العمل وظایف ورودی چگونه خواهد بود؛ (۳) موارد آزمونی که به حداکثر حافظه یا سایر منابع نیاز دارند در این آزمون (آزمون فشار) انجام می‌شود؛ (۴) موارد آزمونی که ممکن است اشکالی را در پیکربندی یا برنامهٔ سیستم عامل مجازی به‌وجود آورند در این آزمون طراحی می‌گردند؛ (۵) موارد آزمونی ایجادکنندهٔ کند و کاو گسترده برای داده‌های موجود در دیسک طراحی می‌شوند. فرد آزمون‌کننده ضرورتاً باید برای از کار انداختن برنامه و تفکیک آن تلاش کند.

نسخه‌ای از آزمون فشار، تکنیکی است که آزمون حساسیت[2] نامیده می‌شود. در بعضی مواقع (که در الگوریتم‌های ریاضی شایع می‌باشد) دامنهٔ کوچکی از اطلاعات موجود در محدودهٔ اطلاعات معتبر برای برنامه می‌تواند پردازش شدید یا حتی پردازش نادرستی را به‌وجود آورد. آزمون حساسیت سعی دارد ترکیبات داده‌ای مختلف در محدودهٔ گروههایی از داده‌های معتبر را که بی‌ثباتی یا فرآیند نادرست را بوجود می‌آورند، کشف کند.

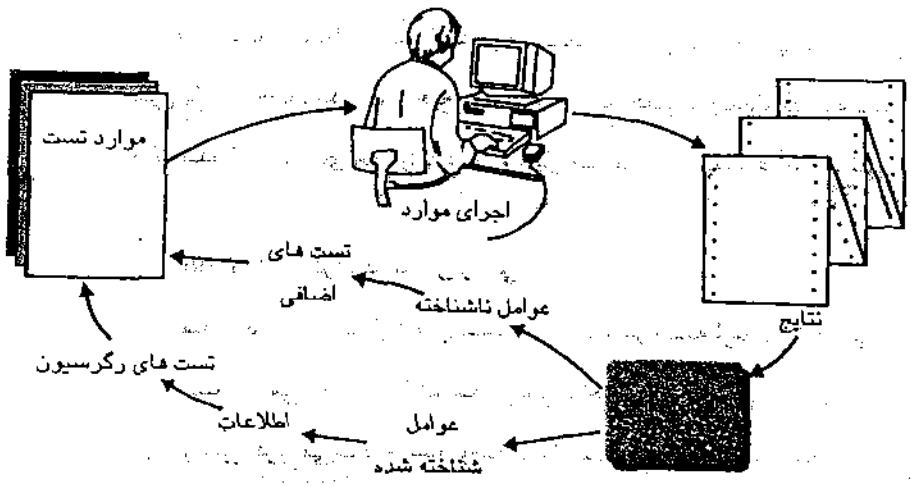
۱۸-۶-۴ آزمون عملکرد

در سیستم‌های جاسازی شده و بلادرنگ، نرم‌افزاری که فراهم‌کنندهٔ عملکرد مورد نیاز می‌باشد. اما با نیازمندیهای عملکردی و کارآیی هماهنگی ندارد، قابل قبول نمی‌باشد. آزمون عملکرد[3] برای آزمون کردن عملکرد نرم‌افزار در زمان کار و در محدودهٔ بافت سیستم تلفیقی طراحی شده است. آزمون عملکرد در تمام مراحل فرآیند آزمون انجام می‌شود. حتی در سطح واحد نیز عملکرد هر یک از پیمانه ها هنگام انجام آزمون‌های جعبهٔ سفید باید مورد ارزیابی قرار داده شود. با این وجود این کار تا زمانی‌که تمام اجزا سیستم به‌طور کامل با یکدیگر تلفیق شده‌اند و یا به‌عبارت دیگر تا زمانی‌که عملکرد واقعی سیستم مشخص نشده است، آزمون عملکرد(کارآمدی) نیز انجام نخواهد شد.

1.Stress testing

2.sensitivity

3.Performance testing



شکل۱۸-۸ فرایند اشکال زدایی

آزمون‌های عملکرد اغلب همراه با آزمون‌های فشار انجام می‌شوند و معمولاً به کاربرد وسایل سنجش سخت‌افزاری و نرم‌افزاری نیاز دارند. یعنی برای ارزیابی میزان مصرف منابع به شکل دقیق انجام این آزمون ضروری می‌باشد. کاربرد وسایل سنجش خارجی می‌تواند فواصل زمانی اجرا، ثبت رویدادها (وقفه‌ها) هنگام وقوع این رویدادها و وضعیت های نمونه دستگاه های را کنترل و نظارت کند. با اضافه کردن رمز به سیستم، فرد آزمون‌کننده می‌تواند شرایط و موقعیت‌هایی را تشخیص دهد که به از کار افتادن احتمالی سیستم منتهی می‌شود.

۱۸-۷ هنر اشکال زدایی

آزمون نرم‌افزار فرآیندی است که قابل طرح‌ریزی و مشخص شدن نظام مند می‌باشد. طراحی موارد آزمون باید انجام شود، راهبرد تعریف گردد و نتایج حاصله در مقایسه با انتظارات تجویز و تعیین شده مورد ارزیابی قرار گیرد.

اشکال‌زدایی[1] در نتیجه انجام مؤفق آزمون حاصل می‌گردد. یعنی هنگامی که یکی از موارد آزمونی خطایی را کشف می‌کنند، اشکال‌زدایی یعنی فرآیند از بین بردن و برطرف ساختن خطاها نیز به دنبال این کشف انجام می‌شود. اگر چه اشکال‌زدایی می‌تواند یک فرآیند منظم باشد و باید یک فرآیند منظم باشد، اما

ارجاع به وب

باگنت، پیگیر مسائل امنیتی و اشکالات پنهان نرم افزارهای مبتنی بر کامپیوترهای شخصی می باشد و در این راستا، اطلاعات مفیدی را تحت عناوین رفع اشکال مهیا نموده است :
www.bugnet.com

1.Debugging

انجام این کار یک نوع هنر است. یک مهندس نرم‌افزار که نتایج حاصل از آزمون را مورد ارزیابی قرار می‌دهد، اغلب با نشانهٔ مشخص‌کنندهٔ یک مشکل نرم‌افزاری مواجه می‌شود. یعنی نشانهٔ خارجی خطا و دلیل داخلی خطا ممکن است هیچ رابطهٔ واضح و مشخصی با یکدیگر نداشته باشند. فرآیند ذهنی ضعیف درک شده که نشانه‌های خطا را به عامل و دلیل خطا مرتبط می‌کند، فرآیند اشکال‌زدایی می‌باشد.

۱۸-۷-۱ فرآیند اشکال زدایی

اشکال‌زدایی متفاوت از آزمون است. اما همیشه در نتیجهٔ آزمودن اتفاق می‌افتد.[1] با مراجعه به شکل ۱۸-۸ مشخص می‌گردد که فرآیند اشکال‌زدایی با اجرای موارد آزمون آغاز می‌شود. نتایج حاصل از آزمون‌ها مورد ارزیابی قرار می‌گیرد و عدم وجود رابطه میان مورد پیش‌بینی شده و مورد واقعی نیز مشخص می‌گردد. در بسیاری از موارد داده‌های غیر متناظر بیانگرعامل مهمی است که مخفی شده است. فرآیند اشکال‌زدایی سعی دارد علامت‌های موجود را با عامل به‌وجودآورندهٔ آن تطابق دهد و بدین‌ترتیب موجب اصلاح خطا می‌شوند.

فرآیند اشکال‌زدایی همیشه یکی از دو نتیجهٔ زیر را در برخواهد داشت: ۱) عامل به‌وجودآورندهٔ خطا مشخص شده، اصلاح و برطرف می‌گردد یا (۲) عامل به‌وجودآورندهٔ خطا مشخص نخواهد شد، در حالت دوم، فرد انجام‌دهندهٔ اشکال‌زدایی ممکن است به عاملی مشکوک باشد و آزمون موردی را برای کمک کردن به اعتبار بخشیدن به شک خود و مطمئن شدن از شک خود طراحی می‌کند و در جهت تصحیح خطا به شکل تکراری گام برمی‌دارد.

چرا انجام اشکال‌زدایی بسیار مشکل است؟ در بسیاری از موارد روان‌شناسی انسانی با پاسخ به فنآوری نرم‌افزاری در ارتباط بیشتری می‌باشد. با این وجود تعداد کمی از مشخصه‌های خطا سر نخ‌های زیر را فراهم می‌کنند:

۱- علامت خطا و عامل خطا باید از لحاظ جغرافیایی و محل قرارگیری کنترل شود. یعنی ممکن است علامت مربوط به خطا در یک بخش از برنامه ظاهر شود و این در حالی است که عامل به‌وجودآورندهٔ خطا ممکن است در محلی دورتر از علامت قرار گرفته باشد. ساختارهای کاملاً جفت شدهٔ برنامه‌ای (فصل ۱۳) این وضعیت را وخیم‌تر می‌سازد.

۲- هنگامی که خطای دیگری اصلاح می‌شود ممکن است علامت مربوط به خطا (به‌صورت موقت) ناپدید شود.

۳- علامت‌های مربوط به خطا ممکن است از طریق عوامل غیرخطایی ایجاد شود.

نقل قول

تنوعی که در زبانهای برنامه سازی (جهت رفع اشکال)، با آن مواجهیم، از تنوع تمام سیستمهای متعارف دیگر بیشتر است.
جان گولد

---

۱. در ساخت این جمله ما وسیع ترین دیدگاه را نسبت به آزمون ایجاد کرده ایم. نه تنها سازنده، نرم افزار را برای تحویل مورد آزمون قرار می دهد که مشتری و کاربر نیز هر بار که آن را استفاده می نمایند، در بوته آزمون قرار داده اند!

۴- علامت مربوط به خطا ممکن است توسط خطای انسانی ایجاد شود که این نوع خطا به راحتی قابل پی‌گیری نمی‌باشد.

۵- علامت مربوط به خطا ممکن است در نتیجه مشکلات زمان‌بندی ایجاد شود و نه به‌واسطهٔ مشکلات پردازشی.

۶- ساخت مجدد و دقیق و درست شرایط ورودی (به‌عنوان مثال کاربرد بلادرنگ که در آن نظم داده‌های ورودی غیرمشخص و تامعین می‌باشد) مشکل است.

۷- علامت مربوط به خطا ممکن است متناوب باشد. که این حالت در سیستم‌های جاسازی شده‌ای که نرم‌افزار و سخت‌افزار را به‌صورت پیچیده‌ای با یکدیگر هماهنگ می‌سازند، شایع می‌باشد.

۸- علامت مربوط به خطا که ممکن است در نتیجهٔ عوامل توزیع شده در تعدادی از وظائف در حال اجرا در پردازنده های مختلف به‌وجود آمده باشد.[CHE90] [1]

در طول اشکال‌زدایی ما ممکن است با خطایی با دامنهٔ متفاوت از خفیف (به‌عنوان مثال شکل نادرست خروجی) تا خطاهای فاجعه برانگیز (مانند خرابی سیستم که خسارات شدید اقتصادی و فیزیکی را به‌وجود می‌آورد) مواجه شویم. با افزایش پیامدهای خطا میزان فشار برای یافتن عامل به‌وجودآورندهٔ خطا نیز افزایش می‌یابد. اغلب فشار فرد تکمیل‌کننده و توسعه‌دهندهٔ نرم‌افزار را وادار می‌سازد تا یکی از خطاهای موجود را برطرف کند و در عین حال دو خطای دیگر را معرفی کند.

۱۸-۷-۲ ملاحظات روان شناسی

متأسفانه به‌نظر می‌رسد که مطابق با بعضی از شواهد و مدارک موجود، قابلیت و قدرت اشکال‌زدایی یکی از ویژگی‌های فطری انسانی می‌باشد. بعضی از افراد می‌توانند کار اشکال‌زدایی را به‌خوبی انجام دهند و گروه دیگری از افراد نمی‌توانند این کار را انجام دهند. اگر چه مدارک و شواهد تجربی موجود در زمینهٔ اشکال‌زدایی در معرفی تعبیر و تفسیرهای بسیاری قرار دارد، اما تفاوت‌های عمده‌ای در قابلیت اشکال‌زدایی برای برنامه‌نویسانی با پیش زمینهٔ تجربی و تحصیلی مشابه گزارش شده است.

اشنایدرمن نظریهٔ خود در ارتباط با منابع انسانی اشکال‌زدایی به‌صورت‌ریز بیان می‌کند:

اشکال‌زدایی یکی از خسته‌کننده‌ترین بخش‌های برنامه‌نویسی می‌باشد. اشکال‌زدایی، مشتمل بر عناصر مسئله یا حل معما است که با شناخت خطا و اشتباه انجام شده از سوی شما هماهنگ می‌باشد. اضطراب و عدم تمایل افزایش یافته برای پذیرش احتمال وقوع خطا موجب افزایش مشکل بودن وظیفه می‌شود. خوشبختانه هنگام اصلاح شدن کامل خطا پس از شناسایی خطا میزان تنش به شدت کاهش می‌یابد.

1.Cheung,W.H.

اگر چه "یاد گرفتن" کار اشکال‌زدایی بسیار مشکل می‌باشد. اما تعدادی از روش‌های موجود در زمینهٔ حل مشکل را می‌توان برای این کار پیشنهاد نمود. این روش‌ها در بخش بعدی بررسی خواهند شد.

۱۸-۷-۳ رهیافت های اشکال زدایی

صرف‌نظر از رهیافت اتخاذ شده برای اشکال‌زدایی، اشکال‌زدایی دارای هدفی فوق‌العاده است و این هدف عبارت است از: یافتن و تصحیح عامل خطای نرم‌افزاری. این هدف را با ترکیب ارزیابی نظام‌مند، مشهود و شانس حاصل می‌شود. بردلی[BRA85][1] روش اشکازدایی را به شکل زیر توصیف می‌کند:

اشکال‌زدایی کاربرد مشخص شیوه علمی تکمیل شده در طول ۲۵۰۰ سال می‌باشد. مبنای اشکال‌زدایی یافتن منبع مشکل است که با تقسیم‌بندی دودویی از طریق هزینه کار که پیش‌بینی‌کنندهٔ مقادیر جدید موارد مورد آزمون می‌باشد، حاصل می‌شود.

مثال ساده و غیرنرم‌افزاری این مورد بدین‌صورت است: یک لامپ در خانه روشن نمی‌شود. اگر عامل خاموش نشاندن آن در داخل خانه نباشد، بنابراین عامل خاموش بودن لامپ را باید در مدار اصلی در خارج از خانه یافت. من چرخی در اطراف خانه می‌زنم تا ببینم که آیا بقیهٔ همسایه‌ها برق دارند یا خیر؟ لامپ مشکوک را به سوکتی که در حال کار کردن و یک دستگاه در حال کار کردن را به مدار مشکوک متصل می‌کنم و همین ترتیب در فرضیات آزمون نیز دنبال می‌شود.

به‌طور کلی به مقولهٔ مربوط به روش‌های اشکال‌زدایی عبارتند از: ۱) نیروی مادی ۲) عمل برگشت و جستجوی مجدد ۳) حذف عامل خطا[MYE79][2].

یک محدودیت زمانی برای رفع اشکال تعیین کنید. برای مثال دو ساعت برای هر اشکال. بعد از آن، از دیگری کمک بخواهید!

مقولهٔ نیروی مادی[3] اشکال‌زدایی احتمالاً رایج‌ترین شیوه جذاسازی عامل خطای نرم‌افزاری می‌باشد. هنگامی که سایر راه‌حل‌های موجود در اشکال‌زدایی با شکست مواجه می‌شوند. ما شیوه اشکال‌زدایی نیروی مادی را اعمال می‌کنیم. با استفاده از فلسفهٔ "اجازه دادن به کامپیوتر برای یافتن خطا"، تخلیه از حافظه حاصل می‌شود، علائم حین اجرا به‌وجود می‌آیند و برنامه با عبارت WRITE بارگذاری می‌شود. ما امیدواریم که در بخشی از محل قرارگیری اطلاعات سر نخی بیابیم که ما را به سمت یافتن عامل خطا هدایت کند. اگر چه مجموعه اطلاعات تولید شده ممکن است، در نهایت به مؤفق ساختن ما منتهی شود، اما این اطلاعات اغلب به وقت و تلاش اتلاف شدهٔ در راه یافتن خطا منتهی می‌گردند.

1.Bradley,J.H.

2.Myers,G.

3.brute force

ابتدا باید در مورد نوع یافتن خطا با دقت کامل تفکر نمود. عمل برگشت و جستجوی مجدد[1] یکی از روش‌های اشکال‌زدایی نسبتاً رایج است که می‌توان با موفقیت آن را در برنامه‌های کوچک دنبال نمود. با شروع از نقطه‌ای که علامت مربوط به خطا در آن‌جا کشف شده است. کد منبع (به‌صورت دستی) و به سمت عقب و به حالت برگشتی جستجو می‌شود تا محل قرارگیری عامل خطا مشخص شود. متأسفانه با افزایش تعداد خطوط منبع خطا مقدار مسیرهای مهم برگشت و جستجوی مجدد نیز به‌طور قابل ملاحظه‌ای افزایش می‌یابد.

روش سوم اشکال‌زدایی - یعنی حذف عامل خطا[2] - با اضافه کردن یا کم کردن و ارائه طرح تقسیم‌بندی دودویی مشخص می‌شود. داده‌های مربوط به وقوع خطا، سازمان‌دهی می‌شوند تا عامل اصلی خطا مشخص شود. "فرضیهٔ عاملی" تعبیه شده است و اطلاعات فوق‌الذکر برای اثبات این فرضیه و یا در آن مورد استفاده قرار می‌گیرند. هم‌چنین لیستی از داده‌های فوق‌الذکر به‌وجود آمده و تکمیل شده است و آزمون‌هایی نیز برای حذف هر یک از آنها انجام شده است. اگر آزمون‌های اولیه نشان‌دهند که فرضیهٔ عامل خاص نوید‌بخش و امیدوارکننده می‌باشد، بنابراین داده‌های موجود در تلاش برای جداسازی خطاها اصلاح می‌شوند.

هر یک از روش‌های فوق‌الذکر اشکال‌زدایی را می‌توان با ابزارهای اشکال‌زدایی کامل و تکمیل نمود. ما می‌توانیم از مجموعهٔ متنوعی از کامپایلرهای اشکال‌زدایی، وسایل کمکی اشکال‌زدایی پویا استفاده کنیم. هم‌چنین می‌توانیم از مولدهای خودکار موارد آزمون، تخلیه از حافظه و نقشه‌های ارجاع متقابل نیز استفاده کنیم. با این وجود این ابزارها نمی‌توانند جایگزین ارزیابی دقیق بر اساس سند طراحی نرم‌افزار کامل و کد منبع واضح و آشکار باشند.

هر بخشی در ارتباط با روش‌های اشکال‌زدایی بدون توضیح دادن در مورد حامی قوی - یعنی افراد - ناقص خواهد بود. هر یک از افراد می‌توانند چندین ساعت یا چندین روز به معمای ایجاد شده در ارتباط با خطای موجود به بررسی و پژوهش بپردازد. ما می‌توانیم مشکل موجود را برای همکاری که از موضوع پرت و ناامید است توضیح دهیم و لیست این خطاها را در اختیار وی قرار دهیم. به‌نظر می‌رسد که بلافاصله عامل به‌وجودآورندهٔ خطا کشف شود.

بلافاصله پس از کشف شدن خطا باید خطای کشف شده را تصحیح و اصلاح کنیم. اما همان‌طور که قبلاً نیز اشاره شد، تصحیح خطا ممکن است خطاهای دیگری را نیز به‌وجود آورد و بنابراین خسارت بیشتری را بر سیستم وارد کند. ون‌وک [VAN89][3] سه سؤال سادهٔ ذیل را مطرح می‌کند و از هر یک از

1.Backtracking

2.cause elimination

3.Van Vleck,T.

مهندسین نرم‌افزار می‌خواهد که قبل از اصلاح خطا که حذف‌کنندهٔ عامل به‌وجودآورنده خطا می‌باشد این سؤالات را از خود بپرسند:

پرسش

هنگامی که مشغول اصلاح یک خطا هستیم، چه پرسشهایی باید از خود بپرسیم؟

۱- آیا عامل خطا مجدداً در بخش دیگری از برنامه نیز به‌وجود آمده است؟ در بسیاری از مواقع عیب به‌وجود آمده در برنامه حاصل الگوی نادرست منطقی است که در بخش دیگری به‌وجود آمده است. توجه دقیق به الگوی منطقی می‌تواند به کشف خطاها منتهی شود.

۲- **با تغییری که من قصد انجام آن را دارم کدام‌یک از "خطاهای بعدی" به‌وجود خواهند آمد؟** قبل از انجام اصلاح، کد منبع باید مورد ارزیابی قرار داده شود تا هماهنگی ساختار داده‌ها و ساختار منطقی ارزیابی شود. اگر اصلاحی که قرار است انجام شود، بخش کاملاً هماهنگ شده و جفت شدهٔ یک برنامه می‌باشد باید به تغییرات انجام شده توجه و دقت بیشتری مبذول شود.

۳- ***برای جلوگیری از به‌وجود آمدن این خطا در وهلهٔ اول چه کاری می‌توانیم انجام دهیم.*** این سؤال اولین مرحله در را ایجاد یک روش آماری تضمین کیفیت نرم‌افزار می‌باشد (فصل ۸)، اگر ما فرآیند و محصول را اصلاح کنیم، خطا از برنامهٔ فعلی ما حذف خواهد شد و احتمالاً خطا از تمام برنامه‌های بعدی ما نیز حذف خواهد شد.

۸-۱۸ خلاصه

آزمون نرم‌افزاری به بیشترین درصد تلاش‌های فنی در فرآیند (پردازش) نرم‌افزاری توجه دارد. اما ما در حال حاضر فقط درک نکات ظریف طرح‌ریزی آزمون سیستماتیک، اجرا و کنترل این آزمون‌ها را آغاز کرده‌ایم!

هدف از آزمون نرم‌افزاری کشف خطاها می‌باشد. برای دستیابی به این هدف مجموعه از مراحل مختلف آزمون - یعنی آزمون واحد، تلفیقی، تصدیق و آزمون‌های سیستم - طرح‌ریزی و اجرا می‌شوند.

آزمون‌های واحد و تلفیقی بر تصدیق و تعیین صحت کارکردی یک واحد (جزء) و بر ملحق ساختن یک واحد به ساختار برنامه تأکید دارند. آزمون تصدیق قابلیت پی‌گیری نیازمندیهای نرم‌افزاری را نشان می‌دهد و آزمون سیستم نیز نرم‌افزار را پس از ملحق شدن آن، به سیستمی بزرگتر تصدیق و تأیید می‌کند.

هر یک از مراحل آزمون از طریق مجموعه‌ای از فنون نظام‌مند آزمون که در طراحی موارد آزمون مؤثر بوده‌اند، تکمیل می‌گردد. در هر یک از مراحل آزمون سطح انتزاعی که نرم‌افزار با توجه به آن در نظر گرفته می‌شود بسیار گسترده است.

اشکال‌زدایی برخلاف آزمون، باید به‌عنوان یک هنر در نظر گرفته شود. فعالیت اشکال‌زدایی با آغاز کردن فعالیت خود با توجه به علامت نشان‌دهندهٔ مشکل باید به سمت عامل به‌وجودآورندهٔ خطا هدایت شود. از میان منابع بسیاری که در طول اشکال‌زدایی موجود هستند، با ارزش‌ترین، نمونه مشورت با سایر

اعضاء تیم مهندسی نرم‌افزار می‌باشد. شرایط موجود برای نرم‌افزار با کیفیت‌تر مستلزم وجود یک روش سیستماتیک‌تر برای آزمون می‌باشد. دان واولمن [DUN82][1] این مطلب را به‌صورت زیر عنوان می‌کند:

"آنچه که مورد نیاز می‌باشد یک راهبرد کلی است که فضای آزمون راهبردی را گسترش می‌دهد. این راهبرد نیز همانند تکمیل نظام‌مند نرم‌افزار که تحلیل، طراحی و برنامه نویسی بر مبنای آن انجام می‌شود، از لحاظ فراروش خود کاملاً سنجیده شده می‌باشد."

در این فصل ما فضای (محدوده) آزمون راهبردی را با در نظر گرفتن مراحلی که بیشترین احتمال تأمین هدف آزمون را دربرداشته‌اند، مورد بررسی قرار داده‌ایم. یعنی به یافتن و برطرف کردن خطاهای موجود به شکل مؤثر و منظم پرداخته‌ایم.

1.Dunn,R.andR.Ullman

مسایل و نکاتی برای تفکر و تعمق بیشتر

۱۸-۱ به زبان خودتان، اختلاف میان تعیین صحت و اعتبارسنجی را بیان کنید. آیا هر دو از راهبردهای آزمون و شیوه های طراحی مورد آزمون استفاده می‌کنند؟

۱۸-۲ مشکلاتی را لیست کنید که در صورت ایجاد یک گروه مشتقل آزمون بوجود خواهد آمد. آیا گروه ITG و گروه SQA از افراد یکسان تشکیل می شوند؟

۱۸-۳ آیا همواره امکان توسعه یک راهبرد برای آزمون نرم‌افزار که از مراحل شرح داده شده در بخش ۱۸-۱-۳ استفاده کنند، وجود دارد؟ برای سیستم‌های جاسازی شده چه عواقبی خواهد داشت؟

۱۸-۴ اگر شما تنها می‌توانستید سه شیوه طراحی مورد آزمون را برای انجام آزمون واحد انتخاب کنید، کدام‌ها انتخاب می‌شدند و چرا؟

۱۸-۵ چرا اجرای آزمون واحد برای پیمانه‌های به هم پیوسته دشوار است؟

۱۸-۶ مفهوم "ضد خطا" (بخش ۱۸-۳-۱) یک راه کاملا" موثر برای تهیه یاریگر ساخته شده اشکال زدایی هنگام عدم پوشش خطا می باشد :

الف. مجموعه رهنمودهایی برای " ضدخطا

ب. مزایای استفاده از این تکنیک را شرح دهید.

پ. معایب آن را توضیح دهید.

۱۸-۷ یک راهبرد آزمون جامعیت برای هر یک از سیستمهای پیاده‌سازی شده در مسائل۱۶-۴ تا ۱۶-۱۱ توسعه دهید. مراحل آزمون را تعریف کنید، ترتیب جامعیت را ذکر کنید، یک نرم‌افزار آزمون اضافی را مشخص کنید و ترتیب انسجام خود را توجیه کنید. فرض کنید تمام پیمانه‌ها (یا کلاس‌ها)، مورد آزمون واحد قرار گرفته‌اند و قابل دسترسی می باشند. توجه: ممکن است ابتدا اندکی کار طراحی لازم باشد.

۱۸-۸ زمان‌بندی پروژه چه تاثیری بر آزمون جامعیت خواهد داشت؟

۱۸-۹ آیا آزمون واحد در همهٔ شرایط امکان‌پذیر و یا حتی مطلوب است؟ مثالی برای توجیه پاسخ ارائه کنید.

۱۸-۱۰ چه کسی باید آزمون اعتبارسنجی را انجام دهد ـ سازنده نرم‌افزار یا کاربر نرم افزار؟ پاسخ خود را توجیه کنید.

۱۸-۱۱ یک راهبرد آزمون کامل برای سیستم خانه امن که در فصلهای اول کتاب معرفی شد، توسعه دهید. آن را در قالب یک مشخصه آزمون مستند کنید.

۱۸-۱۲ به‌عنوان یک پروژه کلاسی، یک راهنمای اشکال‌زدایی برای خود توسعه دهید. این راهنما باید زبان و تذکرات و دقدقه های مهم مربوط به سیستم که در دانشگاه آموخته‌اید را، در برداشته باشد. کار خود را با چارچوبی از مباحث مرور شده توسط شما و استادتان آغاز کنید. این راهنما را در محیط محلی خود برای استفاده دیگران انتشار دهید.

فهرست منابع و مراجع

[BEI84] Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand-Reinhold, 1984.

[BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981, p. 37.

[BRA85] Bradley, J .I-", "The Science and Art of Debugging," *Computerworld*, August 19,1985,pp.35-38.

[CHE90] Cheung, W.H., J.P. Black, and E. Manning, "A Framework for Distributed Debugging," *IEEE Software*, January 1990, pp. 106-115.

[DUN82] Dunn, R. and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, 1982, p. 158.

[GIL95] Gilb, T., "What We Fail to Do in Our Current Testing Culture," *Testing Techniques Newsletter*, (on-line edition, ttn@soft.com), Software Research, January 1995.

[MCO96] McConnell,S., "Best Practices: Daily Build and Smoke Test", *IEEE Software*, vol. 13, no. 4, July 1996, 143-144.

[MIL77] Miller, E., "The Philosophy of Testing," in *Program Testing Techniques*, IEEE

[MUS89] Musa, J.D. and Ackerman, A.F., "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989, pp. 19-27.

[MYE79] Myers, G., *The Art of Software Testing*, Wiley, 1979.

[SHO83] Shooman, M.L., *Software Engineering*, McGraw-Hill, 1983.

[SHN80] Shneiderman, B., *Software Psychology*, Winthrop Publishers, 1980, p. 28.

[VAN89] Van Vleck, T., "Three Questions About Each Bug You Find," *ACM Software Engineering Notes*, vol. 14, no. 5, July 1989, pp. 62-63.

[WAL89] Wallace, D.R. and R.U. Fujii, "Software Verification and validation: An Overview," *IEEE Software*, May 1989, pp. 10-17.

[YOU75] Yourdon, E., *Techniques of Program Structure and Design*, Prentice-Hall, 1975.

خواندنیهای دیگر و منابع اطلاعاتی

Books by Black *(Managing the Testing Process*, Microsoft Press, 1999); Dustin, Rashka, and Paul *(Test Process Improvement.' Step-by-Step Guide to Structured Testing*, Addison-Wesley, 1999); Perry *(Surviving the Top Ten Challenges of Software Testing: A People-Oriented Approach*, Dorset House, 1997); and Kit and Finzi *(software Testing in the Real World: Improving the Process*, Addison-Wesley, 1995) address software test ing strategies.

Kaner, Nguyen, and Falk *(Testing Computer Software*, Wiley, 1999); Hutcheson *(Software Testing Methods and Metrics: The Most Important Tests* McGraw-Hill, 1997); Marick *(The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing*, Prentice-Hall, 1995); Jorgensen *(Software Testing: A Craftsman's Approach*, CRC Press, 1995) present treatments of the subject that consider testing methods and strategies.

In addition, older books by Evans *(Productive Software Test Management*, Wiley-Interscience, 1984), Hetzel *(The Complete Guide to software Testing*, QED Information

Sciences, 1984), Beizer [BEI84], Quid and Unwin *(Testing in Software Development*, Cambridge University Press, 1986), Marks *(Testing Vel)' Big Systems*, McGraw-Hill, 1992, and Kaner et al. *(Testing Computer Software*, 2nd ed., Van Nostrand-Reinhold, 1993), delineate the steps of an effective testing strategy, provide a set of techniques and guidelines, and suggest procedures for controlling and tracking the testing process. Hutcheson *(Software Testing Methods and Metrics*, McGraw-Hill, 1996) presents testing methods and strategies but also provides a detailed discussion of how measurement can be used to achieve efficient testing.

Guidelines for debugging are contained in a book by Dunn *(Software Defect Removal*, McGraw-Hill, 1984). Beizer [BEI84] presents an interesting "taxonomy of bugs" that can lead to effective methods for test planning. McConnell *(Code Complete*, Microsoft Press, 1993) presents pragmatic advice on unit and Integration testing as well as debugging.

A wide variety of information sources on software testing and related subjects is available on the Internet. An up-to-date list of World Wide Web references that are relevant to testing concepts, methods, and strategies can be found at the SEPA Website: http://www.mhhe.com/engcs/compsci/pressman/resources/test-strategy.mhtml