آزمون رگرسیون نرم افزار: مفاهیم و روش‌ها

نویسنده: علیرضا خلیلیان

نسخه ۱.۴ - ۹۴/۱۰/۲۸

هر گونه کپی یا پردیسی از محتوای این مقاله با ارجاع به آن و ذکر نام نویسنده و منبع مقاله آزاد است.
آزمون رگرسیون نرمافزار

1- آزمون رگرسیون نرمافزار

چه در جریان توسعه نرمافزار تا مرحله‌ی تکمیل و تحویل آن و چه در جریان فرایند نگهداری نرمافزار، به‌دلیل مطرح شدن نیازمندی‌های جدید از سوی مشتریان، تغییر و اصلاح در

1 System Patching
نیازمندی‌های محصول برای تطبیق با فناوری‌ها و محیط‌های جدید، رفع اشکال‌های پنهانی ایجاد شده در مراحل مختلف توسعه و بالاخره بروزرسانی برای هماهنگی با محیط، مکرراً نرم‌افزار دستخوش تغییرات قرار می‌گیرد. این تغییرات، که با هدف اصلاح کاسته‌ها و خطاهای و یا بهبود خصوصیات موجود انجام می‌شود، ممکن است خود تاثیرات مخرب بر گیفت و قابلیت اطمینان نرم‌افزار داشته باشد به طوری که رفتار آزومون شده سیستم نرم‌افزاری تنزل یا پسرفت نماید.
نتیجه این فرآیند، ایجاد اشکالاتی موسوم به خطاهای پسرفت (خطاهای رگرسیونی) خواهد بود.
با وجود این که عملیات همه جانبه و دقیق تتوسعه امکان مجزاسازی تغییرات را فراهم می‌سازد، پیچیدگی ذهنی سیستم‌های نرم‌افزاری نوین، پیش‌بینی دقیق تأثیر یک تغییر را دشوار می‌سازد. از آنجا که هر اصلاحات، در بخش‌های مختلف نرم‌افزار انجام می‌شود و نتیجه این تغییرات، تنها به بخش‌های اصلاح شده نرم‌افزار محدود نمی‌شود، انجام مکرر "آزومون جامع" برای اطمینان از صحبت عملکرد تمام بخش‌های نرم‌افزاری، اعم از تغییریافته و بدون تغییری ضروری است. اهمیت این موضوع زمانی آشکار می‌شود که بدین‌مدت عدم افت گیفت نرم‌افزار آزومون شده امروز ضروری در چرخه نگهداری آن است. بنابراین، آزومون رگرسیون نرم‌افزاری را در اساس، می‌توان جستجو برای یافتن خطاهای رگرسیونی (تأثیرات نامطلوب تغییرات بر بخش‌هایی اصلاح نشده) دانست [KP02]. جایگاه آزومون رگرسیون در چرخه تتوسعه نرم‌افزار در شکل 1-1 نشان داده شده است.

[۱. Regress
۲. Regression Bugs
۳. Exhaustive Test
۴. Nonregression]
Component-based Software Engineering

HMD02

IEEE

Hsu

RH97

IEEE610.12-90

KPR05

IEEE

Component-based Software Engineering

تعريف رسمي آزمون رگرسیون نرمافزاری به این صورت است:

برای برنامه‌های متعدد P، نسخه‌های اصلاح شده آن مورد استفاده قرار گرفته است؛ روشهای بیایید که از استفاده کند و اطلاعات کافی در مورد روش استفاده به دست دهد.

آزمون رگرسیون نرمافزاری در استاندارد IEEE به این صورت تعیین شده است:

اجرای مجدد انتخاب برای سیستم با مؤلفه برای تشخیص این که اصلاحات منجر به تأثیرات ناخواسته نشده و سیستم مزبور هنوز نیازمندی‌های تغییر شده را پرداخته است;

تعیین مناسب دیگر برای آزمون رگرسیون نرمافزاری به این صورت است:

آزمون رگرسیون نرمافزاری، از فعالیت‌های نگهداری نرمافزار برای برنامه‌های اصلاح شده است؛ تا اطلاعات حاصل شده که تغییرات صحیح هستند و تأثیر عکس (نامطلوب) بر روی بخش‌های تغییریافته برنامه نداشتند [KPR05].

آزمون رگرسیون نرمافزاری، دو فاز مجزا دارد: فاز اولیه و فاز بحرانی [RH97]. فاز اولیه آزمون رگرسیون نرمافزاری، سپس از تحویل یک نسخه جدید از نرمافزار آغاز می‌شود، در خلال این فاز، برنامه‌نویسان به بهبود و تصحیح نرمافزار می‌پردازند. پس از تکمیل تصمیم‌گیری، فاز دوم یا بحرانی آزمون رگرسیون نرمافزاری آغاز می‌شود. در جریان این فاز، فراکسیون غالب، آزمون رگرسیون است که کلیه زمان آن به دلیل ضرب العمل ناپذیر محصول نرمافزار محدود است و از سوی دیگر به دلیل رقابت شدید در بازار نرمافزار، کمیته کردن هزینه‌ها در فاز بحرانی اهمیت دارد. به عنوان مثال یک فن انجام آزمون رگرسیون ممکن است از سال بیش از آزمون 1، اطلاعات تحلیل برنامه که در جریان فاز اولیه جمع آوری شده است، برای دستیابی به کاهش هزینه‌ها و انتخاب موارد مناسب، بهره‌گیرد.

علاوه بر این، سبک‌های جدید توسعه نرمافزار چالش‌های بیشتری را نیز برای انجام کارامد آزمون رگرسیون ایجاد می‌کند [YH12]. برای نمونه در توسعه نرمافزار مبتنی بر مؤلفه از تعداد بسیاری مؤلفه‌هایی از پیش تولید شده‌ی نرمافزاری به صورت آماده و جعبه سیاب استفاده می‌شود که

---

1 Test history
2 Component-based Software Engineering
اغلب از شرکت‌ها، منابعی با افراد خارجی گریخته‌اند. هر تغییری بر این مسئله‌های خارجی بر
کل سیستم نرم‌افزاری تأثیر می‌گذارد و عملاً آزمون نرم‌افزار روي آنها غیر ممکن است‌زیرا
ساختار و عملکرد داخلی این اجزای برنامه‌نویسی که از آن استفاده می‌کند شناخته شده
نیست. هر چه چرخه حیات توسعه نرم‌افزار کوتاه‌تر باشد (مثل توسعه نرم‌افزار با متدولوژی‌های
جایگزین)، محدودیت‌ها و حساسیت‌های بی‌شیبی بر چگونگی انجام آزمون نرم‌افزاری با توجه به منابع
محدود ایجاد می‌شود.

هزینه‌های آزمون نرم‌افزار، شامل زمان لازم برای اجرای کل موارد آزمون بر روی نرم‌افزار و نیز
هزینه‌های منابع و نیروی انسانی صرف شده برای اجرای آزمون‌ها (موارد آزمون) در هر بار اجرای
آزمون نرم‌افزار است.

با تحلیل نرم‌افزار و برای هر نسخه‌ی جدید، هر بار موارد آزمون جدیدی به مجموعه آزمون
افزوده می‌شوند. به‌عنوان مثال، برای یک نرم‌افزار صنعتی، بخشی از هفته طول می‌کشد تا تمام
دنباله‌ای آزمون (مجموعه‌ی مربی موارد آزمون برای اجرا) برای یک برنامه با 20 هزار خط کد اجرای
شوند [RUC01]. بنابراین آزمون‌گران در آزمون نرم‌افزار، با مشکل حجم سپاس زیاد و روبه
رشد آزمون‌ها و از طرفی نیاز به تکرار فراوان آزمون کل، پس از هر تغییر جزئی برای اعتبارسنجی
نسخه‌ی جدید نرم‌افزار مواجه می‌شوند. (شکل 1-2). بدبیایی است که کنار گذاشتن تصادفی موارد
آزمون، ریسک فراوانی دارد و قابلیت اطمینان نسخه‌ی جدید را شدتگزین دی‌کس در مخاطره قرار می‌دهد. لذا
یافتن مجموعه‌ی کوچکتر و غیرافزونه از موارد آزمون، که بتواند به‌طور مؤثر و کارا نرم‌افزار را
بی‌پاساژی، از اهمیت زیادی برخوردار است [RUC01], [EMR02], [RUC01]. بدیلیل تکرارهای زیاد
آزمون نرم‌افزار، بخشی از این هزینه‌های نگهداری نرم‌افزار را به خود اختصاص می‌دهد.

به‌طوری‌که، تخمین زده می‌شود که آزمون نرم‌افزار
می‌مکن است در اثر اجراهای مکرر، تقریباً نیمی از کل هزینه‌های نگهداری نرم‌افزار را به‌خود اختصاص
دهد [RUC01]. بنابراین، مهم‌ترین مسئله‌ای که آزمون‌گران نرم‌افزار در این مرحله با آن مواجه‌ند،
این است که چگونه در هر مرحله از آزمون نرم‌افزار، زیرمجموعه‌ای مناسب از موارد آزمون را
برای آزمون مؤثر بر روی نسخه‌ی جدید انتخاب کنند؟
برای به صرفه‌بودن آزمون رگرسیون، آزمون‌گران سعی می‌کنند که دنباله آزمون‌ها را ذخیره کنند که برای اجرایی مجدد در مراحل بعدی آزمون رگرسیون از آنها استفاده کنند. در این‌ها زمینه‌ها فن‌جوا و متداول‌های گوناگونی در دست مطالعه قرار گرفته‌اند [MRE02][REM02][OTP98]

آزمون‌های باکیفیتی که در خلال توسعه نسخ متعدد آزمون‌افزار صورت می‌گیرد [You08]، آنها در مجموعه‌ای را می‌توان با حذف موارد آزمون منسوخ با شناسایی و علامت‌گذاری موارد آزمون افزونه به‌دست آورد. موارد آزمون افزونه با انواع منسوخ آن کامل متفاوت هستند. این تفاوت‌از آن‌چه‌ی ناشی می‌شود که آزمون‌های افزونه اجرایی هستند، اما با درنظر گرفتن یک معيار آزمون به‌خصوص اهمیت کهن‌اند ندارند. مثلًا موارد آزمون‌های کم‌سیر‌های یکسانی می‌پوشاند با توجه به معیاری ساختاری افزونه هستند اما موارد افزونه که با نخست‌های یکسانی تطبیق دارند، از لحاظ عملکردی افزونه‌های هستند. موارد آزمون افزونه ممکن است در اثر فعالیت هم‌زمان دو طراح آزمون یا تغییرات هم‌زمان در کد ایجاد شوند. این‌گونه آزمون‌ها کارایی کلی فراوان آزمون‌ها تحت تأثیر قرار نمی‌دهند. اما تأثیر شدیدی بر موازنتی میان‌هزینه و منفعت آزمون و می‌گذراند. موارد آزمون افزاهم نه تنها خاطرین آن‌ها نیز می‌باشد، بلکه هزینه‌های فراوان آزمون را نیز افزایش می‌دهند. با این وجود، این‌گونه موارد آزمون دور انتخاب‌ها نمی‌شوند زیرا ممکن است در آزمون نسخ آنی نرم‌افزار مفید باشد.

\[1\] Obsolete
برای افزایش کارایی آزمون رگرسیون از فنون یا متدولژی‌هایی استفاده می‌شود. این فنون را به دو دسته می‌توان تقسیم کرد:

- فنونی که از مجموعه آزمون‌های موجود به شکلی مجدداً استفاده می‌کنند.
- فنونی که مورد الایه آزمون جدید ساخته و به مجموعه آزمون فعلي اضافه می‌کنند.

چهار متدولژی که بر پایه استفاده‌های مجدد از دنباله‌آزمون‌های موجود، از نخست‌الاصلی
نرم‌افزار هستند، عبارتند از: آزمون‌های مجدد کامل، آزمون‌های رگرسیون اختیاری، کاهش
(کمینه‌سازی) مجموعه آزمون و اولویت‌دهی موارد آزمون. خانواده‌ای از فنون که با روش‌های
ایجاد مورد الایه آزمون جدید کار می‌کنند، افزایش مجموعه آزمون نام‌دارند.

در ادامه بحث، ابتدا انواع آزمون‌های رگرسیون و موردالهای آزمونی معرفی می‌شوند و در بخش
بعدی تا حدی که به این تحقیق مربوط می‌شود، گذراً مختصر خواهیم داشت بر بهبود درک و
اندازه‌گیری نرم‌افزار. در نهایت متدولژی‌های اصلی آزمون رگرسیون تشريح خواهند شد.

1- انواع آزمون رگرسیون و موردالهای آزمون

آزمون‌های رگرسیون را می‌توان به دو دسته بیشتر و اصلاحی تقسیم کرد [YH12]. آزمون‌های رگرسیون
پیشرو آزمونی است که حاوی تغییر مشخصات برنامه تغییر کرده است. بنی ساخته‌ای تغییر کرده
باید با توجه به مشخصات جدید آن آزمون‌های شود. به این نوع آزمون به فنون تولید مجموعه آزمون
نیاز دارند. آزمون‌های رگرسیون اصلاحی حاوی تغییرات در تصمیمات طراحی و کدهای واقعی است و
مشخصات تغییر نمی‌کند. پس بدون تغییر موردالهای آزمون موجود می‌توان از آنها به منظور آزمون
برنامه تغییر کرده استفاده کرد.

موردهای آزمون را می‌توان به پنج دسته تقسیم کرد [YH12] که دسته‌های اول تا سوم
حاوی موردالهای آزمون داخل مجموعه آزمون فعلي هستند. دسته‌های چهارم و پنجم موردالهای

1 Retest All
2 Regression Test Selection
3 Test Suite Reduction (Minimization)
4 Test Case Prioritization
5 Test Suite Augmentation
6 Progressive
7 Corrective
آزمونی هستند که برای آزمون برنامه‌ی تغییر یافته بايد تولید شوند.

1. قابل استفاده مجدد: موردهای آزمونی هستند که فقط بخش‌هایی از برنامه را که بین دو نسخه تغییر نکرده‌اند، می‌آزمایند. اجرای این موردهای آزمون برای آزمون نسخه تغییر یافته برنامه ضروری نیست ولی آن‌ها در مجموعه آزمون نگه می‌دارند زیرا برای آزمون رگرسیون نسخه‌های بعدی برنامه لازم می‌شوند.

2. قابل آزمون مجدد: موردهای آزمونی هستند که بخش‌هایی از برنامه تغییر کرده از برنامه فعلی در نسخه جدید آن‌ها می‌آزمایند. بنابراین برای آزمون نسخه تغییر گردیده این موردهای آزمون حتماً بايد اجرا شوند.

3. منسوخ: موردهای آزمونی هستند که:

   a. ارتباط بین ورودی/خروجی‌های آن‌ها به‌خاطر تغییر مشخصات دیگر معنی نیست.
   b. به‌خاطر تغییرات برنامه، بخش‌هایی که برای آزمون آن‌ها طراحی شده بودند را دیگر نمی‌آزمایند.

ت. موردهای آزمون ساختاری هستند که دیگر ممکن در پوشش ساختاری برنامه ندارند.

4. ساختاری جدید: ساختار‌های برنامه تغییری‌افته‌ا را می‌آزمایند و پوشش ساختاری در بخش‌های تغییر یافته نسخه جدید برنامه اجرا می‌کنند.

5. مشخصات جدید: موردهای آزمون جدیدی هستند که مشخصات برنامه تغییری‌افته‌را می‌آزمایند؛ بنابراین کدهای جدید تولید شده حاصل از بخش‌هایی تغییر‌یافته مشخصات نسخه جدید.

فنون آزمون رگرسیون انتخابی موردهای آزمون قابل آزمون مجدد را شناسایی می‌کند. فنون کمی‌سازی مجموعه آزمون مورد‌های آزمون منسوخ را شناسایی می‌نمایند. فنون اولویت‌دهی هم در حقيقة روشی بیشتر به براي ساخت یک طرح آزمون هستند.
1- درک و انداده گیری نرم‌افزار

اندازه‌گیری و درک نرم‌افزار، دو جنبه مهم‌دشی نرم‌افزار هستند که اخیراً تأکید زیادی بر آن‌ها شده و به‌شکل‌های مختلفی داشته‌اند. بسیاری از ای‌ده‌های این دو زمینه‌های تحقیقاتی با موضوع تحقیق حاضر مرتبط هستند. از این‌رو در یک‌جا در حد ضرورت، اطلاعاتی را پیرامون این دو جنبه که با این تحقیق ارتباط دارند، بیان می‌نماییم.

یکی از طریقی که مهندسین می‌توانند از انداده‌گیری بهره ببرند، انداده‌گیری خصوصیات محصول فعلي و فرآیندهای کنونی به‌منظور پیش‌بینی خصوصیات محصول آتی است [Mir08]. مدل‌های پیش‌بینی حاصل، کمک بسیاری در برطغی طراحی و کتاب‌سازی آزمون‌های رگرسیون خواهد داشت. از این مدل‌ها می‌توان جهت تخمین خصوصیات به‌کاربردی آزمون‌های رگرسیون مرتبط استفاده کرد.

انتخاب فنون انداده‌گیری مورد استفاده و نیز متغیرهای محاسبه، یک چالش اساسی در این زمینه است. این موضوع کاملاً به‌منظور اطلاعات مرتبط به مسأله آزمون رگرسیون و نیز آن‌چه (به لحاظ هزینه‌ای باصرفه) دردسترس قرار دارد، و استفاده از اساس آن‌چه از تعریف آزمون رگرسیون ارائه شده، می‌توانیم چند فن انداده‌گیری مرتبط را مشخص نماییم [Mir08].

1. تغییرات مصنوعات نرم‌افزار: روشی مهمی است که ممکن به ایجاد خطاهای رگرسیونی می‌گردد.

2. انداده‌گیری خطای اصلی نرم‌افزار: یکی از مشکلات اصلی در این زمینه‌ها آزمون‌های رگرسیون مربوط می‌شود. تغییراتی که نیاز به ممکن‌سازی کننده‌های نواحی منطقی به‌دست آمده، برای در این صورت جنبه‌ای را به‌طور متعادل در مرحله‌ی آزمون رگرسیون، تحت آزمون تعداد قرار می‌دهیم.

3. انداده‌گیری اطلاعات پوشش آزمون‌ها: مهم‌ترین جنبه‌ایست که اطلاعات دقیق و قاطعی را برای آزمون رگرسیون به‌دست می‌دهد. این اطلاعات در برنامه‌ریزی آزمون‌های رگرسیون کمک می‌کند که به پوشش مطلوب از مصنوعات نرم‌افزاری دست یابیم. استفاده از فنون تحلیل

[1] Fault-prone
زوامان اجرا همچون فروآگذاری کدها را قادر می‌سازد تا میزان پوشش کد توسط هر مورد آزمون را اندازه‌گیری نماییم. اطلاعات پوشش کد اساس اغلب فنون آزمون رژیسیون فعلي را تشکیل می‌دهند.

به واسطهٔ اهمیت اندازه‌گیری اطلاعات پوشش آزمون‌ها و نیز کاربرد آن‌ها در فنون فعلي، آزمون‌های پوشش در ادامه آنها به تفصیل بیشتری مورد بحث قرار می‌یابد.

مفهوم پوشش آزمون، جزو لاینفک عملیات آزمون بوده و نیز سهم عمده‌ای را در تحقیقات آکادمیک به‌خود اختصاص داده است. بخش عظیمی از تحقیق در حوزه آزمون، پیرامون روش‌هایی جهت تولید مجموعه آزمون‌هایی است که با توجه به یک معيار پوشش به‌خصوص به‌دست آمده‌اند. از منظر تولید آزمون، نیاز به مجموعه آزمون‌هایی داریم که کفايت پوشش را نسبت به معيار پوشش خاص تامین می‌نماید. با این وجود، مفهوم پوشش تنها با تولید موارد آزمون محدود نمی‌شود. فنون آزمون رژیسیون نیز عموماً اطلاعات پوشش را به‌کار می‌گیرند. در این حوزه، اندازه‌گیری پوشش موارد آزمونی که هم‌کنون موجود هستند، مورد نظر است. با وجودی که پوشش را می‌توان برحسب مصنوعات گوناگونی که در چرخه جهات نرمافزار موجود هستند (مثل نیازمندی‌ها) تعريف كرد، اما معمولاً برحسب ساختارهایی كه تعريف می‌شود، اندازه‌گیری پوشش کد، عبارت است از شناسایی ساختارهایی از کد که در خلال اجراي یك مورد آزمون بكار می‌رود (اجرا می‌شوند [Mip08]). استفاده از اندازه‌گیري پوشش کد، نویدبخش اين موضوع است كه هرچه یك مجموعه آزمون كد بيشتری را بپوشد، باعث ايجاد قابلیت اطمینان بالاتری در نرمافزار تحت آزمون می‌گردد. اگر مدل رفتاری نرمافزار باقي ماند، در اين صورت پوشش مورد آزمون از یک اجرا تا اجراي ديگر بلا تغيير می‌ماند. چنین داده‌هاي خصوصيات اتکاپيزي از مورد آزمون را به‌دست می‌دهد که در عملیات مختلف آزمون به‌منظور درک بهتر آن، می‌تواند بكار گردد.

در نظر گرفتن ساختارهای مشخصي از کد، متریک‌های پوشش مختلفی را معرفی می‌کند.

در ادبیات موضوع، معیارهای پوشش بسیاری مانند پوشش اشتعال، بلاک‌های اولیه، تعريف و استفاده و غيره معرفی شده است. به‌عنوان نمونه، پوشش دستورات مشخصی که آنها یک

---

1 Basic block
دستور مشخص از کد (یا نمایش شخصی از آن مثل پایت برد) توسط یک مورد آزمون به اجرا درآمده است یا خیر.

جهت انتخاب گیری پوشش یک مورد آزمون، می‌توانیم مسیر اجرایی آن را در برنامه هنگام اجرای مورد آزمون را ببینیم (البته که معمولاً به آن گویی زیادی از برنامه 2 می‌گویند. این کار را می‌توان از طریق تغییر کد، با اضافه کردن کد اضافی به برنامه را در ضمن اجرا ثبت می‌کند و محقق نمود. این کد اضافی در فراپنده به نام فراکدگذاری درج می‌شود. فراکدگذاری، عملیات خودکاری است که می‌تواند در مراحل مختلفی از جمله زمان کامپایل برنامه انجام شود. مزیت استفاده از فراکدگذاری مستقیم، فایل‌هایی اجرا، عدم تأثیر بر افزایش سیستمی همجون کامپایلرهای دیگر است. ضمن این که باعث تسهیل ادامه‌گیری مفصلتر شده و بهدیل عدم نیاز به کامپایل مجدد، کل فاواند را ساده‌تر می‌نماید. به‌طور کلی فراکدگذاری سه مرحله دارد:

1. تحلیل برنامه‌ای اصلی جهت تشخیص اختلال کد.
2. حاصل تحلیل فوق برای فراکدگذاری و یافتن محل‌هایی که باید آنها را درج نمود، مورد استفاده قرار می‌گیرد.
3. کد مورد نظر با فراکدگذاری تغییر داده می‌شود به‌طوری که رفتار برنامه حفظ شود.

پس از فراکدگذاری، موارد آزمون را می‌توان روی آن اجرای کرد و نشانه‌هایی نسخه‌ای روی آن نمودار دارد. روبیت‌ها را برای استخراج پوشش یک مورد آزمون به‌طور کامل بررسی نمود.

در جهت انجام آزمون رگرسیون، اطلاعات پوشش عموماً از نسخه‌های پیشین نرمافزار مورد استفاده قرار می‌گیرد. به‌منظور انتخاب گیری پوشش، مورد آزمون، گزارش‌های اجرایی آزمون رگرسیون را که چگونگی از اجرای مجدد موارد آزمون غیر ضروری است، به‌مطابقت هم‌بسته‌ی برنامه ارائه می‌شود. برای حل این مشکل، محتوای پربررسی و ارزیابی کرده‌اند که داده‌ها یک مورد حاصل از نسخه‌ی قابلی، اگر در دسترس باشند، تخمین منطقی از داده‌های پوشش کنونی هستند. بنابراین، فنون آزمون رگرسیون مثبتی بر پوشش فرض می‌کنند که در هنگام اجرای موارد آزمون روی نسخه‌ی قابلی، از پیش کد فراکدگذاری شده و

1 Trace
2 Program profiling
3 Linker
دآده های پوشش جمع آوری شدند. داده های پوشش موجود را می توان برای هدایت بهبودسازی نسخه کنونی بکار گرفت.

1- آزمون مجدد کامل

این متدولوژی به بیان ساده، به معنی استفاده مجدد تمام موارد آزمون تولید شده در گذشته است [REM02]. به همراه تمام موارد آزمون مورد نیاز برای آزمون نسخه اولیه. مورد آزمونی که در هر مرتبه تغییر یافتن نسخهی آزمون دریافت شده اند. در هر تکرار آزمون رگرسیون، مجدداً اجرا می شود. روشن است که در جریان تغییرات نسخ، بسیاری از موارد آزمون منسوخ خواهند شد. لذا همان طور که گفته شد در جریان انجام مکرو آزمون رگرسیون، حجم مخزن آزمون بسیار زیاد خواهد شد؛ بهطوری که علیرغم سادگی و اطمنان بالایی که حاصل می شود، باعث می شود آزمون رگرسیون، به دلیل نیاز به میزان توجه نابنیز زمان و کار انسانی و غیرعملی باشد در واقع به دلیل هدر رفتند جدی مانع آزمون در اثر هزینه زیاد اجرای غیر لازم آزمون به این میزان آزمونگران سیستم‌های نرم‌افزاری بزرگ، از اجرای آزمون جامع در هر مرتبه خودداری می کنند.

2- آزمون رگرسیون انتخابی

متدهای انتخاب در آزمون رگرسیون که به اختصار RTS گفته می شوند، هر بار انتخاب‌های مجموعه‌ای از کل موارد آزمون موجود، بر اساس عواملی مانند تغییرات رخ داده در کد و رفتار اجرای آزمون‌ها برای اجرای انتخاب می‌گند و به این ترتیب، هزینه‌ی آزمون رگرسیون را به میزان قابل توجهی کاهش می دهند [REM04][MRE02][RH97][WL92]. در عین حال فنون انتخاب در آزمون رگرسیون، مجموعه آزمون کل را کاهش دنیای نسخه دهنده و موارد آزمون را حذف نمی کنند. مساله‌ای اجرای مجدد انتخابی موارد آزمون را می توان به‌صورت زیر بیان کرد [RH97]:

مورد آزمون ۱ مجرور نامیده می شود هرگاه در نسخه جدید، ویژگی تحت آزمون توضیح و وجود نشانه دهندگان به‌دست اجرا دیگر. مورد آزمون ۲ مجرور است که اگر وابستگی و نیاز اجرای آزمونی باشد که برای نسخه جدید برقرار نشده باشد. مورد آزمون ۳ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۴ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۵ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۶ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۷ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۸ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۹ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۰ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۱ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۲ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۳ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۴ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۵ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۶ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۷ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۸ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۱۹ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۰ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۱ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۲ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۳ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۴ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۵ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۶ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۷ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۸ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۲۹ مجرور است که به‌طور کلی اجرا نشده باشد. مورد آزمون ۳۰ مجرور است که به‌طور کلی اجرا نشده باشد.
تاریخچه، اگر $T'$ مجموعهٔ کل موارد آزمون موجود، $P'$ نسخهٔ اولیه برنامه و $T''$ نسخهٔ جدید $P$ مجموعهٔ از موارد آزمون برای اجرای روی $T'$ در نظر می‌گیریم، $P'$ در نظر نیاز $T'$ توصیه‌ها مشخص می‌شود.

با آزمودن $P'$ توصیه‌های مورد نیاز $T''$ توصیه‌ها مشخص می‌شود.

در صورت نیاز، ایجاد می‌شود، که $T''$ به عنوان مورد نیاز جدیدی برای $P'$ است. این ایجاد آزمون‌های جدید برای $P'$ و $T'', T'$ برای آن.

یافته‌ها، همچنین شامل زیر مسائلی یعنی آزمون‌هایی در $T$ است که مسأله‌ش واقعی مشخص‌ش می‌پردازد، که مسأله، این مسأله، یعنی بخش‌هایی از نسخهٔ جدید $P'$ و مجموعه‌ی مشخصه‌های جدید $S$ که نیاز به آزمون بیشتر دارند. گام را (2) و (4)، بر مسأله‌ی اجرای دنباله آزمون دلالت دارند، که مسأله برای اجرای کارای آزمون‌ها و بررسی نتایج آزمون‌ها برای تصحیح برنامه، است. گام (5) مسأله، به‌طور کامل قطعی دریافت آزمون است.

با بیان ساده‌تر، فنون انتخاب در آزمون‌گرایی در هر مرحله، به انتخاب یک زیر مجموعهٔ از کل مخزن آزمون موجود با استفاده از اطلاعات برنامه و مجموعه آزمون می‌پردازند و در عین حال حجم مخزن آزمون کاهش نمی‌یابد. این انتخاب می‌تواند بر اساس ملاک‌های گوناگون صورت گیرد، مثلاً آگر ملاک انتخاب بر اساس رشته آزمونی با پوشش کامل کد باشد، این انتخاب انتخاب امین در آزمون‌گرایی یک گویند. مسأله بسیار مهم در مورد این فنون، متغیر بین ایمنی، یا کارایی است. فنون RTS این شرایط به‌خصوص، تضمین می‌کند که موارد آزمون انتخاب نشده، در اجرا بر روی نسخهٔ جدید نمی‌توانستند خطا‌پذیر اشکال کنند. استراتژی اکثر

1 Safe Regression Test Selection
2 Efficiency
رثمن رگرسیون نرمافزار

فناوری

امنی، این است که با هر پیچیدگی در نسخه و نیاز به تکرار آزمون رگرسیون، با بررسی و مقایسه گراف چرخانی کنترلی نسخه جدید و نسخه قبل، موارد آزمونی انتخاب می‌شوند که بخش‌های اصلاح شده در نسخه جدید را می‌آزمایند [RH97] مطالعات نشان داده‌اند که میزان تغییرات بین دو نسخه، به‌شتی کارایی فنون آزمون رگرسیون انتخابی را تحت تأثیر قرار می‌دهد.

[KPR05]

معلوماً فنون انتخاب موارد آزمون براساس شناسایی تغییرات و میزان تأثیر آنها می‌باشد که با استفاده از فنونی همچون تفاوت معنادار [VF97], تفاوت متساوی [Bnk97], عملیات بررسی‌نگی [ORE07], و گرافهای رابطه‌ای بین کلاسی [GHS96] می‌پذیرد.

از این چاکه انتخاب دقیق موارد آزمون اشکال گنده‌ای خطا که ایجاد ال آزمون رگرسیون انتخابی است، ممکن نیست. چالش اصلی که آزمون‌گران در تمام فنون آزمون رگرسیون انتخابی با آن مواجهند، موانعی بین منفعت- هزینه در مورد این فنون است [KP02]

1- اولویت‌دهی موارد آزمون

ترتب‌دهی موارد آزمون بر اساس یک معیار ساده‌تر به هدف آزمون که عموماً کشف سریع‌تر خطاهای بی‌دستیابی به پویش سریع‌تر کد نرم‌افزار است، اولویت‌دهی موارد آزمون نامیده می‌شود [EMR01][RUC01][RUC99] به پیش‌ساده، فن اولویت‌دهی موارد آزمون در حالی کلی، کل موارد آزمون موجود در مجموعه آزمون را نگه می‌دارد؛ در عین حال پیش‌ از اجرای موارد آزمون، آن‌ها را مربوط می‌کند تا به این صورت در دستیابی سریع‌تر به اهداف آزمون، به آزمون‌گران کمک کند.

تعیین رسمی مسائلی اولویت‌دهی موارد آزمون خستگی بار توسط رودرمل [1 و گروه در سال 2001 ارائه شد [RUC01], به این ترتیب که اگر پر رشته آزمون، PT را مجموعه‌ی تمامی جایگزینی موارد آزمون موجود در PT و را تابعی از بر روی مجموعه‌ی اعداد حقیقی در نظر بگیریم، مسائلی اولویت‌دهی موارد آزمون به‌صورت زیر تعیین می‌شود:

1 Rothermel
مساله: \( T \in PT \)

\[
\forall T(T \in PT)(T \neq T')(f(T') \geq f(T))
\]

که در اینجا \( PT \) بیانگر تمامی اولویت‌های (تریپتی) ممکن و \( f \) به هر کدام از این تریپتی‌های موجود اعمال شود و مقدار عددی به دست آمده، نشان‌دهنده اولویت آن تریپت در میان سایر تریپتی‌ها (بر اساس معیار شیاستگی) 1 در نظر گرفته شده در است.

فضای جستجو در مسأله اولویت‌دهی موارد آزمون، در حقيقة جایگزینی از کلیه موارد آزمون موجود در رشته آزمون است، به‌گونه‌ای که این تریپت اجرای آشکار و نیز بهترین حداکثر خطاهای ممکن باشد. بنابراین این مسأله اولویت‌دهی موارد آزمون، در حقیقت یک مسأله جستجو است که در برخی مراجع این مسأله را معادل با مسأله کوله‌پشتی صفر و یک دانسته‌اند، که مسأله فوق، سخن و بدون راه حل قطعی است. لذا فنون ارائه شده برای مسأله اولویت‌دهی موارد آزمون MKA، مکانش‌های بوده و با استفاده از فنون وزن دهی و معیارهای پوشش 1 گوناگون، برای بهبود سرعت کشف خطای کارایی بیشتر آزمون با هم رقابت می‌کند و همچنین در حالی که بر دیگری برتری [LHH07][KKT07][ERK04][KP02][EMR02][RUC01].

1- کاهش مجموعه آزمون

روشن است که هر چه تعداد موارد آزمون در فرایند آزمون بیشتر باشد، احتمال تأمین نیازمندی‌های آزمون بیشتر می‌گردد. اما مشکل اینجا است که رشد ب رویه مجموعه آزمونی تواند مشکل ساز شود. به عبارت دیگر، موارد آزمون جدید ممکن است تولید و در مجموعه درج شود تا نیازمندی‌های بیشتری را تأمین نماید. در نتیجه موارد آزمون موجود در مجموعه آزمون، ممکن است بیشتر از حد نیاز برای تأمین نیازمندی‌های آزمون باشد. مسأله اینجاست که اگر تعدادی از موارد آزمون از مجموعه آزمون حذف شود، این مجموعه کمک کننده‌ای می‌باشد. مسأله اینجاست که اگر آزمون‌ها که مجموعه‌ای اولیه تأمین می‌کرد، برآورده می‌نماید. این مسأله به "کاهش مجموعه آزمون" (کمینه‌سازی) مجموعه آزمون یا پالایش موارد آزمون) مشهور بوده و زیرمجموعه حاصل.

\(^1\) Fitness Metric
\(^2\) Coverage Criteria
مجموعه‌ی نماینده‌ی یک شاخه نامیده می‌شود. اگر هیچ زیرمجموعه‌ای از مجموعه‌ی نماینده، تمام نیازمندی‌ها را تأمین نکند، آن‌ها مجموعه‌ی بینه‌ی یا مجموعه‌ی کمی‌تهمام

[RHR02][WHL97]

تعریف رسمي مساله‌ی کمینه‌سازی مجموعه‌ی آزمون بهصورت زیر است [93]:

فرض: یک مجموعه آزمون $T$ از موارد آزمون $\{t_1, t_2, ..., t_m\}$، یک مجموعه از نیازمندی‌های آزمون $\{r_1, r_2, ..., r_n\}$ که باید برآورده شوند تا یوشش مطلوب از برنامه حاصل شود، و زیرمجموعه‌های $T$ از $\{t_1, t_2, ..., t_m\}$ به هرکدام به یکی از $r_i$ ها متسبس است به بقیه‌ی که هر یک از موارد آزمون $t_i$ متعلق به $r$, تا تامین نماید.

نتیجه: زیرمجموعه‌ی کمینه‌ای از $T$ که به همیشگی $r_i$ هایی که مجموعه‌ی غیرکمینه‌ی $T$ پیدا کنید که همهی $r_i$ تامین نماید.

فیگور کاوش مجموعه‌ی آزمون با تشخیص و حذف موارد آزمون افزونه، دنباله آزمون را به‌طور دائمی کاوش می‌دهد (برخلاف فیگور آزمون رگرسیون انتخابی). اما این کاوش ممکن است در بلند مدت بر گرفتار ترم افزار تأثیر سوء داشته باشد. از سوی دیگر، کاوش تعداد موارد آزمون به‌عنوان که همهی نیازمندی‌های آزمون را برآورده سازد، سبب افزایش کارایی فایندر آزمون می‌شود. علت این مسئله‌ای این است که زمان اجرای موارد آزمون کمتر شده و نهایتاً منجر به کاهش زمان کل آزمون می‌شود. بنابراین مسئله‌ی کاوش مجموعه‌ی آزمون در واقع به مسئله‌ی یافتن یک زیرمجموعه‌ی کمینه‌ای اصلی که بتواند تمام نیازمندی‌های آزمون را هم‌اکنون مجموعه‌ی اصلی برآورده سازد، بی‌دیده می‌شود. این کاوش باید به‌گونه‌ای انجام شود که قابلیت اشکارکندگی خطای مجموعه‌ی حاصل نسبت به مجموعه‌ی اولیه، بدون تغییر یا با کاهش قابل اطمینان (نسبت به میزان کاوش به‌دست آمده در زمان و منابع آزمون ناشی از کاوش مجموعه‌ی آزمون) همراه باشد. برقراری تعادل بین کاوش توان اشکارسازی خطای مجموعه‌ی آزمون حاصل نسبت به کاوش به‌دست آمده در زمان و منابع مصرفی به دلیل کاهش حجم آزمون‌ها، هم‌ووه هم‌وضعی

چالشی بوده است. اصولاً در مورد کاوش مجموعه‌ی آزمون، دو هزینه اهمیت دارد [RHR02]

1. هزینه‌ی اجرای فایندر (ابزار) کاوش مجموعه‌ی آزمون برای تولید مجموعه‌ی کاوش‌پذیر: یک ایزبر کاوش مجموعه‌ی آزمون را می‌توان درپی ارائه نشخه‌های نرم افزار اجرای اجرای آن می‌تواند

1 Representative Set
بهصورت خودکار و در ساعات خلوتی، انجام شود و در این صورت هزینه‌ی اجرای ایزاز خیلی مشکل‌زا نخواهد بود. علاوه بر این هنگامی که با استفاده از یک ایزاز، مجموعه آزمون کاهش داده شد، هزینه‌ی این کاهش به‌تدریج در اجرای‌های مکرر آزمون و استفاده از مجموعه‌ی کاهش یافته در نگارش‌های بعدی توزیع شده و در مقایسه با سایر هزینه‌ها کم اهمیت می‌گردد.

۲- حذف برخی از موارد آزمون که در صورت اجرای منجر به آشکارسازی خطای می‌شوند: این هزینه از مورد قبلی مهم‌تر است. حذف چنین موارد آزمونی موجب کاهش درصد کشف خطای مجموعه آزمون شده و از کاربرای آن می‌کاهد. اثر این کاهش به‌تدریج با یک کاهشی مجدد آن در نسخه‌های بعدی نرم‌افزار افزایش یافته تا جایی که حجم خطاهای از دسترفته زیاد شده و فراپید آزمون دچار اختلال و عدم کارایی می‌گردد. به روش برای پرداخت هزینه‌های مربوط به حذف موارد آزمون آشکار کننده خطای وجود دارد:

آ- بر مبنای هر مورد آزمون؛ با داشتن برنامه خطدار $P$ و مجموعه آزمون $T$ یکی از راه‌های انداره‌گیری هزینه‌ی مجموعه آزمون کاهش یافته بر کشف خطای شناسایی موارد آزمونی $T_{\text{min}}$ نیستند. این است که در $P$ وجود دارد و خطاهایی را از آشکار می‌کند اما در $T$ تعداد کاهش کننده خطای موجود کرد. نکته مهم اینجاست که ممکن است چندین مورد آزمون یک خطای واحد را آشکار نمایند. در این حالت می‌توان بعضی از آن‌ها را حذف نمود بدون اینکه تأثیری در کشف خطای داشته باشند.

ب- بر مبنای هر مجموعه آزمون؛ روش دیگر دسته‌بندی نتایج کاهش مجموعه آزمون بر حسب یک خطا در $P$ به یکی از سه روش زیر است: ۱) هیچ مورد آزمونی در $T$ به یکی از سه روش زیر است: $T_{\text{min}}$ نمی‌کند. ۲) بعضی موارد آزمون هم در $T$ و هم در $T_{\text{min}}$ نمی‌کند. ۳) بعضی از موارد آزمون در $T$ آشکار کننده خطای موجود اما در $T_{\text{min}}$ هیچ مورد آزمون آشکار کننده خطای وجود ندارد. مورد ۱ حالتی را نشان می‌دهد که برای تأثیر است. حالات ۲ حالتی است که کاهش مجموعه موجب کاهش درصد کشف

1 Off-peak
2 On a per-test-case basis
3 On a per-test-suite basis
خطا نمی‌شود و حالت سوم حالتی است که کاهش مجموعه آزمون باعث می‌شود.
کشف خطا با اختلال مواجه شود. توجه کنید که خطاهایی که توسط اکثر موارد آزمون
قابل کشف باشد، پس از کمینه‌سازی نیز می‌توان کشف نمود در حالت که خطاهایی که
توسط تعداد محدودی از موارد آزمون قابل کشف باشد، پس از کمینه‌سازی در معرض
عدم کشف قرار می‌گیرند.

عوامل کاهش مجموعه آزمون یک مساله‌ی بپیچیده‌تر تاکید می‌شود. اما در برخی
بررسی‌های اخیر کاهش مجموعه آزمون چندهدفه‌ی نیز مورد توجه قرار گرفته است که برخی از
این اهداف پوشش کد، تاریخ‌های کشف خطای جذبیت و هزینه‌های اجرایی هستند.

[YSR07][BMK04][RHO98]

در کل مساله کاهش مجموعه آزمون‌ها و انتخاب موارد آزمون از این جهت که زیر مجموعه‌ای
از مجموعه اصلی را تولید می‌کنند شیبی یک‌دیدگی می‌باشد. معیار کاهش مجموعه آزمون این است
که آیا زیر مجموعه‌ی مناسب، همه نیازمندی‌های آزمون را تأمین می‌کند در حالی که تمرکز
انتخاب موارد آزمون بر خصوصیتهای تغییراتی سیستم نرم‌افزاری است، در بعضاً از تفسیری‌های
اولیه [LWh90] کاهش مجموعه آزمون در حوزه‌ی انتخاب موارد آزمون دست‌بندی می‌شود.
بعضی از روش‌هایی که ارائه شده می‌تواند تا مسائل تاریک و لازم بودن اخلاقیت حل نمایند. مثالاً
روش ارائه شده توسط وکالات و همکاران [WSK06] از مجموعه‌ی زیریک استفاده می‌کند تا
مسأله اولویت‌دهی و انتخاب موارد آزمون را همزمان حل نماید. با تحقیقاتی که در [LHH07] انجام
شده مجموعه‌ای روش‌های ابتدایی و الگوریتم‌های زیریکی را برای حل مسأله‌ی کاهش مجموعه
آزمون و اولویت‌دهی موارد آزمون ارائه نموده است.

از آن‌جایی که هر مورد آزمون نیاز به یک یا چند نیازمندی آزمون را برآورد نماید، ارتباط
میان آن‌ها را می‌توان با یک ماتریس بنام ماتریس مورد آزمون-نیازمندی نمایش داد. مثلاً از آن‌ها
در جدول 1-1 مشاهده می‌نمایید.

---
1 Single Objective
2 Multi Objective
3 Walcott
جدول (1-1) ماتريس مورد آزمون-نیازمندی

<table>
<thead>
<tr>
<th>نیازمندی اموردهای آزمون</th>
<th>نیازمندی ۱</th>
<th>نیازمندی ۲</th>
<th>نیازمندی ۳</th>
<th>نیازمندی ۴</th>
<th>نیازمندی ۵</th>
</tr>
</thead>
<tbody>
<tr>
<td>مورد آزمون ۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
</tr>
<tr>
<td>مورد آزمون ۲</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
</tr>
<tr>
<td>مورد آزمون ۳</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
</tr>
<tr>
<td>مورد آزمون ۴</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
<td>۱</td>
</tr>
</tbody>
</table>

مجموعه‌ای (مورد آزمون ۱، مورد آزمون ۲، مورد آزمون ۳، مورد آزمون ۴) معرف موارد آزمون و مجموعه‌ای (نیازمندی ۱، نیازمندی ۲، نیازمندی ۳) معرف نیازمندی‌های آزمون می‌باشد. عدد ۱ در سطح ۱ و سنتون آریان می‌گوید که مورد آزمون قائم می‌تواند نیازمندی از آن نیازمندی را تأمین نماید و عدد ۰ به معنای عدم وجود رابطه می‌باشد. معمولاً فنون کاهش مجموعه آزمون‌ها ماتریس اخیر را به‌عنوان ورودی گرفته و در خروجی زیرمجموعه‌ای تولید می‌کند که کماکان همه‌ی نیازمندی‌های آزمون را پرآورده نماید. مسائل پایین‌کننده‌ی زیرمجموعه‌ای نیم‌به‌نیم تولید کلی NP کمی می‌کند.

1-۹- افزایش مجموعه آزمون

انواع روبیکردهای مورد استفاده در آزمون رگرسیون (انتخاب، اولویت‌دهی و کاهش) از مجموعه آزمون موجود استفاده مجدد می‌نمایند. اما هنگام رگرسیون سالساله‌ی دیگری بررسی می‌کند که مجموعه آزمون موجود آن بخش‌ها را نمی‌آزماید. بنابراین یکی از فاقدت‌های توسه‌های دهنده‌گان هنگام رگرسیون سنجش کیفیت مجموعه آزمون بعد از تغییر برنامه است. هدف این است که رفتارهای جدید یا تغییر کرده که با مجموعه آزمون موجود آزموده‌نیم‌شونده و نیازمندی مورد آزمون جدید هستند، شناسایی شود. لذا برای افزایش سودمندی مجموعه آزمون در پاسخ به تغییرات، دو کار باید انجام شود [SCA08] :

- وارسی شود آیا مجموعه آزمون‌های موجود برای تغییرات ایجاد شده در برنامه کافی
1- رویکرد کلی برای طراحی الگوریتم‌های کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون

الگوریتم‌های کاهش مجموعه آزمون، به عنوان ورودی مجموعه آزمون \( T \) را گرفته و در خروجی یک مجموعه آزمون جدید تولید می‌کند. در حالی که الگوریتم‌های اولویت‌دهی مجموعه \( T \) را گرفته و یک توالی از موارد آزمون را در خروجی می‌سازند. برای درکی در بحثی که در ادامه می‌آید، هر دو خروجی را \( T' \) می‌نامیم. الگوریتم‌های کمیته‌سازی مجموعه آزمون و اولویت‌دهی موارد آزمون

صفات مشترکی دارند [JHr03].

1. همه الگوریتم‌ها در ساخت \( T' \) میزان سهم و مفید بودن یکی از موارد آزمون را بر اساس خصوصیات برنامه تحت آزمون \( T \) و هدف \( T' \) مورد ارزیابی قرار می‌دهند. با وجودی که خصوصیات متنوعی مطرح شده‌اند، اما همگی را می‌توان به دو نوع پوستی و هزینه تقسیم‌بندی نمود. پوستی یک نیاز مناسبی به برنامه \( P \) می‌دارند. خصوصیات مختلفی از مشترکی دارند [JHr03].

2. انشعاب و جریان داده‌ها، خطرات، سالمندی و خطا‌ها بودن مؤلفه‌های \( T \) را در بر می‌گیرد. هزینه نیز شامل زمان اجرای \( P \) به ازای یک مورد آزمون، زمان برای و آماده‌سازی براي اجرای یک مورد آزمون به‌خصوص در حالتی که از شیپ‌سازی استفاده شده باشد. الگوریتم‌ها موارد آزمون \( T \) را بر اساس ترتیبی از این خصوصیات ارزیابی می‌نمایند. مثلاً مرجع [HGSr93] انواع مختلفی از پوستی را جهت کاهش مجموعه آزمون به دست آورده‌اند. فن آن‌ها هنگام تصمیم‌گیری درباره‌ی موارد آزمونی که باید در \( T' \) قرار گیرند، به ازای هر یک از

\[^{1}\text{Contribution and goodness}\]
وجود‌یت‌هایی برای انتخاب می‌کند که برای ایجاد ترتبیه از موارد آزمون در 'T' انواع مختلفی از پوشش را همراه با میزان خطای بردن پیمان‌ها مورد استفاده قرار داده‌اند. ۲. الگوی‌ها: کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون، در دفعاتی (فرکانسی) که سهم یک مورد آزمون را مجدد محسوب می‌نماید، با هم فرق می‌کند. یک روش ممکن است این سهم را یکبار محسوب کرده و از آن برای شمول یا عدم شمول موارد آزمون در استفاده کنده، در مرحله [RUC01] این فرکانس را جمع کل‌ نامیده شده و آن برای محاسبه اولویت‌دهی مبتنی بر افزایش استفاده می‌شود.

۳. افزایش می‌تواند از سهم مورد آزمون برای تعیین مورد آزمونی که باید به 'T' اضافه شود استفاده کند (مورد استفاده‌ای کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون) و یا از آن در جهت تعیین مورد آزمونی که باید از حذف شود، بهره ببرد (مورد استفاده‌ای کاهش مجموعه آزمون). اولی را یا افزایش یا دومی را فن 'تجزیه' یا نامیده. فن افزایشی کار خود را با مجموعه خالی 'T' شروع می‌کند و موارد آزمون را یکی یکی از دیگری به آن اضافه می‌کند. اما فن تجزیه با مجموعه 'T' شروع کرده و به تدریج موارد آزمون را از آن حذف می‌کند تا به‌دست آید مهم‌ترین مزیت روش تجزیه در کاهش مجموعه آزمون که است که می‌توان در حین فراوانی کاهش، در هر زمان الگویی را متوافق نمود و مجموعه‌ای باقی‌مانده، نیازمندی‌های موارد آزمونی که پوشش از آنها خالی خانه‌های یاد می‌توان تضمین کردن که پوشش کامل حاصل شده است. ترکیب این صفات یا بعضاً محاسبه دقیق می‌برد مورد آزمون، فرکانس ارزیابی (مجدید) سهم هر مورد آزمون و روش ساخت 'T' می‌تواند الگوی‌های متفاوتی تولید نماید.

1 Total
2 Total-based
3 Additional-based
4 Build-up
5 Break-down
1-11- نتیجه‌گیری

آزمون رگرسیون نرم‌افزاری، از فعالیت‌های نگهداری نرم‌افزار برای برنامه‌ای اصلاح شده است. تا اطمینان حاصل شود که تغییرات صحیح هستند و تأثیر عكس (نامطلوب) بر روی بخش‌های تغییریافته‌ی برنامه نداشته‌اند. برای حل مشکل هزینه‌ی زیاد آزمون رگرسیون، متدولوژی‌های مختلفی ارائه‌شده‌اند که مهم‌ترین آن‌ها عبارتند از: آزمون رگرسیون انتخابی، کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون.

در مورد کاهش مجموعه آزمون، دو هزینه اهمیت دارد: ۱) هزینه اجرای فایل‌برد (ابزار) کاهش مجموعه آزمون برای تولید مجموعه‌ی کاهش‌یافته و ۲) حذف برخی از موارد آزمون که در صورت اجرا منجر به آشکارسازی خطایی می‌شوند. هزینه‌ی دوم اهمیت بیشتری دارد. بوشک و هزینه‌ی خصوصیت مهم موارد آزمون است که مجموعه‌های کاهش در ساخت مجموعه‌ی کاهش‌یافته میزان سرمای و مفید بودن هر یک از موارد آزمون را بر اساس آن می‌سنجد.

همان‌گونه فنو آزمون رگرسیون نرم‌افزار با تولید مورد آزمون، آزمون رگرسیون چند هدفه، مساله اوراکل‌های آزمون و هزینه آن‌ها، سایر دامنه‌های کاربرد و آزمون رگرسیون غیر وظیفه‌مندی از مباحثی هستند که در مقاله‌ی بعدی مورد بررسی قرار می‌گیرند.