

آزمون رگرسیون نرم افزار: مفاهیم و روش ها

نویسنده: علیرضا خلیلیان

نسخه ۱.۴ - ۹۴/۱۰/۲۸

هرگونه کپی برداری از محتوای این مقاله با ارجاع به آن و ذکر نام نویسنده و منبع مقاله آزاد است.

۱-۱- مقدمه

آزمون رگرسیون نرم افزار، با مفهوم رگرسیون نرم افزار سروکار دارد. رگرسیون نرم افزار عبارت است از ایراد نرم افزاری، که سبب توقف عملکرد یک ویژگی می شود؛ به طوریکه بعد از یک رویداد به خصوص (به عنوان مثال بروزرسانی یا تولید وصله های نرم افزاری^۱) باید این ویژگی انجام شود [HMD02]. به عبارت دیگر، بعد از اصلاح نرم افزار و بروز تغییر، آزمونگران مایلند بدانند که آیا کد تغییر نیافته تحت تأثیر نامطلوب ناشی از این تغییر قرار گرفته است یا خیر. در صورتی که کد تغییر نیافته، تحت اثر نامطلوبی قرار گرفته باشد (عملکرد صحیح قبل دچار اختلال شده باشد)، گفته می شود که خطای رگرسیونی اتفاق افتاده است [KP02]. آزمون رگرسیون، به هر قسم از آزمون نرم افزاری اطلاق می شود که هدف آن، آشکارسازی رگرسیون های نرم افزاری باشد. این رگرسیون ها، هر زمان که عملکرد نرم افزار، که پیش تر به درستی کار می کرد، دچار توقف در این عملکرد مورد نظر شود، رخ می دهند. عموماً رگرسیون ها، نتیجه ی تأثیرات نامطلوب ناشی از تغییرات در نرم افزار هستند. مقاله حاضر مروری به مفاهیم، تعاریف و ادبیات موضوع در حوزه آزمون رگرسیون خواهد داشت.

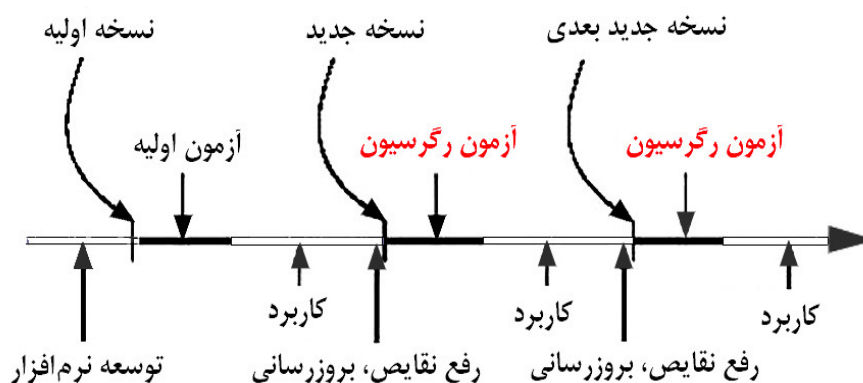
ساختار ادامه مقاله به شرح زیر است: در بخش دوم آزمون رگرسیون نرم افزار تشریح می شود. بخش سوم انواع آزمون رگرسیون و موردهای آزمون را ارائه می دهد. سپس درک و اندازه گیری نرم افزار در بخش چهارم مورد بحث قرار می گیرد. آزمون جامع کامل، آزمون رگرسیون انتخابی، اولویت دهی موردهای آزمون، کاهش مجموعه آزمون و افزایش مجموعه آزمون موضوع بخش های پنجم تا نهم را تشکیل می دهند. در بخش دهم رویکرد کلی برای طراحی فنون کاهش و اولویت دهی مورد بحث قرار می گیرد. بخش یازدهم هم به نتیجه گیری اختصاص دارد.

۱-۲- آزمون رگرسیون نرم افزار

چه در جریان توسعه ی نرم افزار تا مرحله ی تکمیل و تحویل آن و چه در جریان فرایند نگهداری نرم افزار، به دلیل مطرح شدن نیازمندی های 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 کاستی‌ها و خطاها و یا بهبود خصوصیات موجود انجام می‌شود، ممکن است خود تأثیرات مخربی بر کیفیت و قابلیت اطمینان نرم‌افزار داشته باشند به طوری که رفتار آزمون شده سیستم نرم‌افزاری تنزل کرده^۱ و پسرقت نماید. نتیجه این فرایند، ایجاد اشکالاتی موسوم به خطاهای پسرقت (خطاهای رگرسیونی^۲) خواهد بود [Mir08]. با وجود این که عملیات همه جانبه و دقیق توسعه امکان مجزاسازی تغییرات را فراهم می‌سازد، پیچیدگی ذاتی سیستم‌های نرم‌افزاری نوین، پیش‌بینی دقیق تأثیر یک تغییر را دشوار می‌سازد. از آن جا که این اصلاحات، در بخش‌های مختلف نرم‌افزار انجام می‌شود و نیز، اثر این تغییرات، تنها به بخش‌های اصلاح شده کد نرم‌افزار محدود نمی‌شود، انجام مکرر "آزمون جامع"^۳ برای اطمینان از صحت عملکرد تمام بخش‌های نرم‌افزار، اعم از تغییر یافته و بدون تغییر، ضروری است. اهمیت این موضوع زمانی آشکار می‌شود که بدانیم عدم افت^۴ کیفیت نرم‌افزار آزمون شده امری ضروری در چرخه‌ی نگهداری آن است. بنابراین، آزمون رگرسیون نرم‌افزاری را در اساس، می‌توان جستجو برای یافتن خطاهای رگرسیونی (تأثیرات نامطلوب تغییرات بر بخش‌های اصلاح نشده) دانست [KP02]. جایگاه آزمون رگرسیون در چرخه‌ی توسعه نرم‌افزار در شکل ۱-۱ نشان داده شده است.



شکل (۱-۱) جایگاه آزمون رگرسیون در چرخه‌ی توسعه‌ی نرم‌افزار [WO03]

¹ Regress

² Regression Bugs

³ Exhaustive Test

⁴ Nonregression

تعریف رسمی آزمون رگرسیون نرم‌افزاری به این صورت است [Hsu]:
 ”برای برنامه‌ی مفروض P ، نسخه‌ی اصلاح شده آن یعنی P' و مجموعه آزمون T ، که قبلاً برای آزمون P مورد استفاده قرار گرفته است؛ روشی بیابید که از T استفاده کند و اطمینان کافی در مورد درستی P' به دست دهد.“

آزمون رگرسیون نرم‌افزاری در استاندارد IEEE به این صورت تعریف شده است [HMD02]:
 ”اجرای مجدد انتخابی برای سیستم یا مؤلفه برای تشخیص این که اصلاحات منجر به تأثیرات ناخواسته نشده و سیستم مذکور هنوز نیازمندی‌های تعیین شده را برآورده می‌کند (IEEE610.12-90).“

تعریف مناسب دیگر برای آزمون رگرسیون نرم‌افزاری به این صورت است:
 ”آزمون رگرسیون نرم‌افزاری، از فعالیت‌های نگهداری نرم‌افزار برای برنامه‌ی اصلاح شده است؛ تا اطمینان حاصل شود که تغییرات صحیح هستند و تأثیر عکس (نامطلوب) بر روی بخش‌های تغییرنیافته‌ی برنامه نداشته‌اند [KPR05].“

آزمون رگرسیون نرم‌افزاری، دو فاز مجزا دارد: فاز اولیه و فاز بحرانی [RH97]. فاز اولیه‌ی آزمون رگرسیون نرم‌افزاری، پس از تحویل یک نسخه‌ی جدید از نرم‌افزار آغاز می‌شود؛ در خلال این فاز، برنامه‌نویسان به بهبود و تصحیح نرم‌افزار می‌پردازند. پس از تکمیل تصحیحات، فاز دوم یا بحرانی آزمون رگرسیون نرم‌افزاری آغاز می‌شود؛ در جریان این فاز، فرایند غالب، آزمون رگرسیون است که البته زمان آن به دلیل ضرب‌العجل‌های تحویل محصول نرم‌افزاری محدود است و از سوی دیگر به دلیل رقابت شدید در بازار نرم‌افزار، کمینه کردن هزینه‌ها در فاز بحرانی اهمیت فراوانی دارد. به‌عنوان مثال یک فن انجام آزمون رگرسیون ممکن است از سابقه‌ی آزمون^۱ و اطلاعات تحلیل برنامه که در جریان فاز اولیه جمع آوری شده است، برای دستیابی به کاهش هزینه‌ها و انتخاب موارد آزمون مناسب، بهره‌گیرد.

علاوه بر این، سبک‌های جدید توسعه‌ی نرم‌افزار چالش‌های بیشتری را نیز برای انجام کارآمد آزمون رگرسیون ایجاد می‌کند [YH12]. برای نمونه در توسعه نرم‌افزار مبتنی بر مؤلفه^۲ از تعداد بسیاری مؤلفه‌های از پیش تولید شده‌ی نرم‌افزاری به‌صورت آماده و جعبه سیاه استفاده می‌شود که

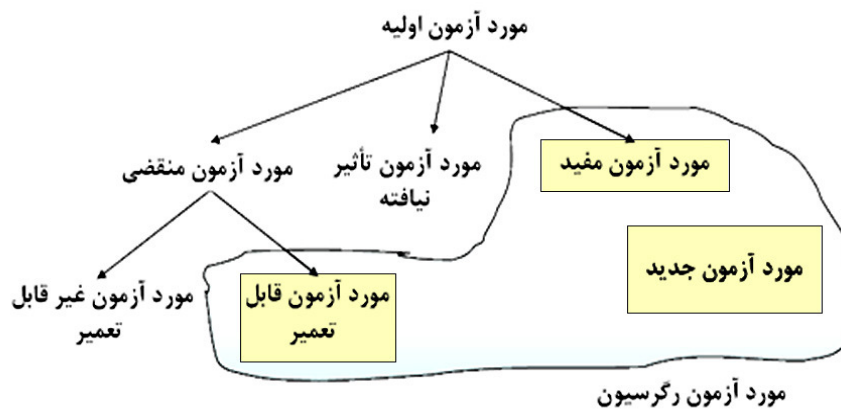
¹ Test history

² Component-based Software Engineering

اغلب از شرکت‌ها، منابع یا افراد خارجی گرفته می‌شوند. هر تغییری بر این مؤلفه‌های خارجی بر کل سیستم نرم‌افزاری تأثیر می‌گذارد ولی عملاً آزمون رگرسیون روی آن‌ها غیر ممکن است زیرا ساختار و عملکرد داخلی این اجزا برای برنامه‌نویسانی که از آن استفاده می‌کنند شناخته شده نیست. هر چه چرخه حیات توسعه نرم‌افزار کوتاهتر باشد (مثل توسعه نرم‌افزار با متدولوژی‌های چابک)، محدودیت‌ها و حساسیت‌های بیشتری بر چگونگی انجام آزمون رگرسیون با توجه به منابع محدود ایجاد می‌شود.

هزینه‌ی آزمون رگرسیون، 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 رگرسیون است.

با تحول نرم‌افزار و برای هر نسخه‌ی جدید، هر بار موارد آزمون جدیدی به مجموعه آزمون افزوده می‌شوند. به‌عنوان مثال، برای یک نرم‌افزار صنعتی، بیش از سه هفته طول می‌کشد تا تمام دنباله آزمون (مجموعه‌ی مرتب موارد آزمون برای اجرا) برای یک برنامه با ۲۰ هزار خط کد اجرا شوند [RUC01]. بنابراین آزمونگران در آزمون رگرسیون نرم‌افزار، با مشکل حجم بسیار زیاد و روبه رشد آزمون‌ها و از طرفی نیاز به تکرار فرایند آزمون کل، پس از هر تغییر جزئی برای اعتبارسنجی نسخه‌ی جدید نرم‌افزار، مواجهند. (شکل ۱-۲). بدیهی است که کنار گذاشتن تصادفی موارد آزمون، ریسک فراوانی دارد و قابلیت اطمینان نسخه‌ی جدید را شدیداً در مخاطره قرار می‌دهد. لذا یافتن مجموعه‌ی کوچکتر و غیرافزونه از موارد آزمون، که بتواند به‌طور مؤثر و کارا نرم‌افزار را بیازماید، از اهمیت زیادی برخوردار است [RUC01][EMR02][REM04]. به‌دلیل تکرارهای زیاد، آزمون رگرسیون نرم‌افزار، بخش زیادی از هزینه‌های نگهداری نرم‌افزار را به خود اختصاص می‌دهد [Bzr79][LWh90][RH96][MRE02]. به‌طوری‌که، تخمین زده می‌شود که آزمون رگرسیون ممکن است در اثر اجراهای مکرر، تقریباً نیمی از کل هزینه‌ی نگهداری نرم‌افزار را به خود اختصاص دهد [RUC01]. بنابراین، مهم‌ترین مسأله‌ای که آزمونگران نرم‌افزار در این مرحله با آن مواجهند، این است که چگونه در هر مرحله از آزمون رگرسیون، زیرمجموعه‌ای مناسب از موارد آزمون را برای آزمون مؤثر بر روی نسخه‌ی جدید انتخاب کنند؟



شکل (۲-۱) تأثیر اصلاحات نرم افزار بر موارد آزمون در آزمون رگرسیون نرم افزار [Hsu]

برای به صرفه بودن آزمون رگرسیون، آزمونگران سعی می کنند که دنباله آزمون ها را ذخیره کنند که برای اجراهای مجدد در مراحل بعدی آزمون رگرسیون از آن ها استفاده کنند. در این زمینه فنون و متدولوژی های گوناگونی در دست مطالعه قرار گرفته اند [OTP98][REM02][MRE02]. یک راه کار ممکن، عبارت است از تولید و نگهداری مجموعه آزمون های با کیفیت که در خلال توسعه ی نسخ متعدد نرم افزار صورت می گیرد [You08]. چنین مجموعه ای را می توان با حذف موارد آزمون منسوخ^۱ یا شناسایی و علامت گذاری موارد آزمون افزونه به دست آورد. موارد آزمون افزونه با انواع منسوخ آن کاملاً متفاوت هستند. این تفاوت از آن جایی ناشی می شود که آزمون های افزونه اجرایی هستند اما با در نظر گرفتن یک معیار آزمون به خصوص اهمیت چندانی ندارند. مثلاً موارد آزمونی که مسیرهای یکسانی را می پوشانند با توجه به معیارهای ساختاری افزونه هستند اما موارد آزمونی که با بخش های یکسانی تطابق دارند، از لحاظ عملکردی افزونه هستند. موارد آزمون افزونه ممکن است در اثر فعالیت همزمان دو طراح آزمون یا تغییرات همزمان در کد ایجاد شوند. این گونه آزمون ها کارایی کلی فرایند آزمون را تحت تأثیر قرار نمی دهند، اما تأثیر شدیدی بر موازنه ی میان هزینه و منفعت آزمون می گذارند. موارد آزمون افزونه نه تنها خطایی آشکار نمی سازند، بلکه هزینه های فرایند آزمون را نیز افزایش می دهند. با این وجود، این گونه موارد آزمون دور انداخته نمی شوند زیرا ممکن است در آزمون نسخ آتی نرم افزار مفید باشند.

¹ Obsolete

برای افزایش کارایی آزمون رگرسیون از فنون یا متدولوژی‌هایی استفاده می‌شود. این فنون را به دو دسته می‌توان تقسیم کرد:

- فنونی که از مجموعه آزمون‌های موجود به شکلی مجدداً استفاده می‌کنند.
 - فنونی که موردهای آزمون جدید ساخته و به مجموعه آزمون فعلی اضافه می‌کنند.
- چهار متدولوژی که بر پایه‌ی استفاده‌ی مجدد از دنباله آزمون‌های موجود، از نسخه‌ی اصلی نرم‌افزار هستند، عبارتند از: آزمون مجدد کامل^۱، آزمون رگرسیون انتخابی^۲، کاهش (کمینه‌سازی) مجموعه آزمون^۳ و اولویت‌دهی موارد آزمون^۴. خانواده‌ای از فنون که با رویکرد ایجاد موردهای آزمون جدید کار می‌کنند، افزایش مجموعه آزمون^۵ نام دارند.
- در ادامه بحث، ابتدا انواع آزمون رگرسیون و موردهای آزمون معرفی می‌شوند و در بخش بعدی تا حدی که به این تحقیق مربوط می‌شود، گذری مختصر خواهیم داشت بر بحث درک و اندازه‌گیری نرم‌افزار. در نهایت متدولوژی‌های اصلی آزمون رگرسیون تشریح خواهند شد.

۱-۳- انواع آزمون رگرسیون و موردهای آزمون

آزمون رگرسیون را می‌توان به دو دسته پیشرو^۶ و اصلاحی^۷ تقسیم کرد [YH12]. آزمون رگرسیون پیشرو آزمونی است که حاوی تغییر مشخصات برنامه تغییر کرده است، یعنی برنامه تغییر کرده باید با توجه به مشخصات جدید آن آزموده شود. پس این نوع آزمون به فنون تولید مجموعه آزمون نیاز دارند. آزمون رگرسیون اصلاحی حاوی تغییرات در تصمیمات طراحی و کدهای واقعی است و مشخصات تغییر نمی‌کند. پس بدون تغییر موردهای آزمون موجود می‌توان از آن‌ها به‌منظور آزمون برنامه تغییر کرده استفاده کرد.

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

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 باید تولید شوند.

۱. **قابل استفاده مجدد^۱**: 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 را در مجموعه آزمون نگه می‌دارند زیرا برای

آزمون رگرسیون نسخه‌های بعدی برنامه لازم می‌شوند.

۲. **قابل آزمون مجدد^۲**: موردهای آزمونی هستند که بخش‌های تغییر کرده از برنامه فعلی

در نسخه جدید آن‌را می‌آزمایند. بنابراین برای آزمون نسخه تغییر کرده، این موردهای

آزمون حتماً باید اجرا شوند.

۳. **منسوخ^۳**: موردهای آزمونی هستند که:

أ. ارتباط بین ورودی/خروجی‌های آن‌ها به‌خاطر تغییر مشخصات دیگر معتبر نیست.

ب. به‌خاطر تغییرات برنامه، بخش‌هایی که برای آزمون آن‌ها طراحی شده بودند را

دیگر نمی‌آزمایند.

ت. موردهای آزمون ساختاری هستند که دیگر سهمی در پوشش ساختاری برنامه

ندارند.

۴. **ساختاری جدید^۴**: ساختارهای برنامه تغییر یافته را می‌آزمایند و پوشش ساختاری در

بخش‌های تغییر یافته نسخه جدید برنامه ایجاد می‌کنند.

۵. **مشخصات جدید^۵**: موردهای آزمون جدیدی هستند که مشخصات برنامه تغییر یافته را

می‌آزمایند؛ یعنی کدهای جدید تولید شده حاصل از بخش‌هایی تغییر یافته مشخصات

نسخه جدید.

فنون آزمون رگرسیون انتخابی موردهای آزمون قابل آزمون مجدد را شناسایی می‌کنند. فنون

کمینه‌سازی مجموعه آزمون موردهای آزمون منسوخ را شناسایی می‌نمایند. فنون اولویت‌دهی هم

در حقیقت روشی پیشرفته برای ساخت یک طرح آزمون هستند.

¹ Reusable

² Retestable

³ Obsolete

⁴ New-structural

⁵ New-specification

۱-۴- درک و اندازه گیری نرم افزار

اندازه گیری و درک نرم افزار، دو جنبه‌ی مهندسی نرم افزار هستند که اخیراً تأکید زیادی بر آن‌ها شده و پیشرفت‌های چشمگیری داشته‌اند. بسیاری از ایده‌های این دو زمینه‌ی تحقیقاتی با موضوع تحقیق حاضر مرتبط هستند. از این رو در بخش جاری در حد ضرورت، اطلاعاتی را پیرامون این دو جنبه که با این تحقیق ارتباط دارند، بیان می‌نماییم.

یکی از طرقی که مهندسی می‌توانند از اندازه گیری بهره ببرند، اندازه گیری خصوصیات محصول فعلی و فرایندهای کنونی به منظور پیش‌بینی خصوصیات محصول آتی است [Mir08]. مدل‌های پیش‌بینی حاصل، کمک بسیاری در برطرف نمودن مسایل آزمون رگرسیون خواهد داشت. از این مدل‌ها می‌توان جهت تخمین خصوصیات که به خروجی آزمون رگرسیون مرتبط است، بهره برد و نیز می‌توان در برنامه‌ریزی این آزمون از آن‌ها استفاده کرد.

انتخاب فنون اندازه گیری مورد استفاده و نیز متریک‌های محاسبه، یک چالش اساسی در این زمینه است. این موضوع کاملاً به ماهیت اطلاعات مرتبط به مسأله آزمون رگرسیون و نیز آنچه (به لحاظ هزینه‌ای با صرفه) در دسترس قرار دارد، وابسته است. بر اساس آنچه از تعریف آزمون رگرسیون ارائه شد، می‌توانیم چند فن اندازه گیری مرتبط را مشخص نماییم [Mir08].

۱. تغییرات مصنوعات نرم افزاری: رویداد مهمی است که منجر به ایجاد خطاهای رگرسیونی می‌گردد. بنابراین، طبیعی است که تاریخچه تکامل محصول را در نظر گرفته و بخش‌های تغییر یافته‌ای را که مورد توجه آزمون رگرسیون است، شناسایی نماییم.

۲. اندازه گیری کیفیت نرم افزار: زمینه‌ی تحقیقاتی دیگری است که از جنبه‌ی مفهومی به آزمون رگرسیون مربوط می‌شود. متریک‌های کیفیت را می‌توان به منظور تشخیص نواحی از کد که خطاخیز هستند^۱، بکار برد. در این صورت چنین نواحی را به طور وسیع‌تر در مرحله‌ی آزمون رگرسیون، تحت آزمون مجدد قرار می‌دهیم.

۳. اندازه گیری اطلاعات پوشش آزمون‌ها: مهم‌ترین جنبه‌ایست که اطلاعات دقیق و قاطعی را برای آزمون رگرسیون به دست می‌دهد. این اطلاعات در برنامه‌ریزی آزمون رگرسیون کمک می‌کنند که به پوشش مطلوب از مصنوعات نرم افزاری دست یابیم. استفاده از فنون تحلیل

¹ Fault-prone

زمان اجرا همچون فراکدگذاری کد، ما را قادر می‌سازد تا میزان پوشش کد توسط هر مورد آزمون را اندازه‌گیری نماییم. اطلاعات پوشش کد، اساس اغلب فنون آزمون رگرسیون فعلی را تشکیل می‌دهند.

به‌واسطه‌ی اهمیت اندازه‌گیری اطلاعات پوشش آزمون‌ها و نیز کاربرد آن‌ها در فنون فعلی آزمون رگرسیون در ادامه آنرا به تفصیل بیشتری مورد بحث قرار می‌دهیم.

مفهوم پوشش آزمون، جزء 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 معیار پوشش به‌خصوص به‌دست آمده‌اند. از منظر تولید آزمون، نیاز به مجموعه آزمون‌هایی داریم که کفایت پوشش را نسبت به یک معیار پوشش خاص تأمین می‌نمایند. با این وجود، مفهوم پوشش تنها با تولید موارد آزمون محدود نمی‌شود. فنون آزمون رگرسیون نیز عموماً اطلاعات پوشش را به‌کار می‌گیرند. در این حوزه، اندازه‌گیری پوشش موارد آزمونی که هم‌اکنون موجود هستند، مورد نظر است. با وجودی که پوشش را می‌توان برحسب مصنوعات گوناگونی که در چرخه‌ی حیات نرم‌افزار موجود هستند (مثل نیازمندی‌ها) تعریف کرد، اما معمولاً برحسب ساختارهای کد تعریف می‌شود.

اندازه‌گیری پوشش کد، عبارت است از شناسایی ساختارهایی از کد که در خلال اجرای یک مورد آزمون به‌کار می‌روند (اجرا می‌شوند) [Mir08]. استفاده از اندازه‌ی پوشش کد، نویدبخش این موضوع است که هرچه یک مجموعه آزمون کد بیشتری را بیوشاند، باعث ایجاد قابلیت اطمینان بالاتری در نرم‌افزار تحت آزمون می‌گردد. اگر مدل رفتاری نرم‌افزار را قطعی در نظر بگیریم، که یک ورودی مشخص همواره مسیر اجرایی یکسانی را طی می‌کند، در این صورت پوشش مورد آزمون از یک اجرا تا اجرای دیگر بلا تغییر می‌ماند. چنین داده‌هایی خصوصیت اتکاپذیری از مورد آزمون را به‌دست می‌دهد که در عملیات مختلف آزمون به‌منظور درک بهتر آن، می‌تواند به‌کار رود.

در نظر گرفتن ساختارهای مشخصی از کد، متریک‌های پوشش مختلفی را معرفی می‌کند. در ادبیات موضوع، معیارهای پوشش بسیاری مانند پوشش انشعاب، بلاک‌های اولیه^۱، تعریف و استفاده و غیره معرفی شده است. به‌عنوان نمونه، پوشش دستورات مشخص می‌کند که آیا یک

¹ Basic block

دستور مشخص از کد (یا نمایش خاصی از آن مثل بایت کد) توسط یک مورد آزمون به اجرا درآمده است یا خیر.

جهت اندازه گیری پوشش یک مورد آزمون، می توانیم مسیر اجرایی آنرا در برنامه هنگام اجرای مورد آزمون ردیابی^۱ نماییم که معمولاً به آن الگوبرداری از برنامه^۲ می گویند. این کار را می توان از طریق تغییر کد، با اضافه کردن کد اضافی که رویدادها را در ضمن اجرا ثبت می کند محقق نمود. این کد اضافی در فرایندی به نام فراکدگذاری درج می شود. فراکدگذاری، عملیات خودکاری است که می تواند در مراحل مختلفی از جمله زمان کامپایل برنامه انجام شود. مزیت استفاده از فراکدگذاری مستقیم فایل های اجرایی، عدم تأثیر بر ابزارهای سیستمی همچون کامپایلرها و پیوند دهنده ها^۳ است. ضمن این که باعث تسهیل اندازه گیری مفصل تر شده و به دلیل عدم نیاز به کامپایل مجدد، کل فرایند را ساده تر می نمایند. به طور کلی فراکدگذاری سه مرحله دارد:

۱. تحلیل برنامه ی اصلی جهت تشخیص ساختار کد.
 ۲. حاصل تحلیل فوق برای فراکدگذاری و یافتن محل هایی که باید آن ها را درج نمود، مورد استفاده قرار می گیرد.
 ۳. کد مورد نظر با فراکدگذاری تغییر داده می شود به گونه ای که رفتار برنامه حفظ شود.
- پس از فراکدگذاری، موارد آزمون را می توان روی آن اجرا کرد و نشانه های ثبت شده از رویدادها را برای استخراج پوشش هر مورد آزمون به کار برد.
- در حوزه ی آزمون رگرسیون، اطلاعات پوشش عموماً از نسخه ی پیشین نرم افزار مورد استفاده قرار می گیرد. به منظور اندازه گیری پوشش هر مورد آزمون، لازم است تا آنرا فراکدگذاری کرده و سپس اجرا نماییم. اما این کار هدف بهینه سازی آزمون رگرسیون را که جلوگیری از اجرای مجدد موارد آزمون غیر ضروری است، به مخاطره می اندازد. برای حل این مشکل، محققان بررسی و ارزیابی کرده اند که داده های پوشش حاصل از نسخه ی قبلی، اگر در دسترس باشند، تخمین منطقی از داده های پوشش کنونی هستند. بنابراین فنون آزمون رگرسیون مبتنی بر پوشش، فرض می کنند که در هنگام اجرای موارد آزمون روی نسخه ی قبلی، از پیش کد فراکدگذاری شده و

¹ Trace

² Program profiling

³ Linker

داده‌های پوشش جمع‌آوری شده‌اند. داده‌های پوشش موجود را می‌توان برای هدایت بهینه‌سازی نسخه‌ی کنونی بکار گرفت.

۱-۵- آزمون مجدد کامل

این متدولوژی، به بیان ساده، به معنی استفاده‌ی مجدد تمام موارد آزمون تولید شده در گذشته است [REM02]؛ یعنی تمام موارد آزمون مورد نیاز برای آزمون نسخه‌ی اولیه، به‌همراه تمام موارد آزمونی که در هر مرتبه تغییر یافتن نسخه‌ی نرم‌افزار تولید شده‌اند، در هر تکرار آزمون رگرسیون، مجدداً اجرا می‌شوند. روشن است که در جریان تغییرات نسخ، بسیاری از موارد آزمون منسوخ^۱ خواهند شد. لذا همان‌طور که گفته شد، در جریان انجام مکرر آزمون رگرسیون، حجم مخزن آزمون بسیار زیاد خواهد شد؛ به‌طوری که علی‌رغم سادگی و اطمینان بالایی که حاصل می‌شود، باعث می‌شود آزمون رگرسیون، به دلیل نیاز به میزان توجه ناپذیر زمان و کار انسانی، غیرعملی باشد در واقع به‌دلیل هدر رفتن جدی منابع آزمون در اثر هزینه‌ی زیاد اجرای غیر لازم آزمون‌ها در این فن، آزمونگران سیستم‌های نرم‌افزاری بزرگ، از اجرای آزمون جامع در هر مرتبه خودداری می‌کنند.

۱-۶- آزمون رگرسیون انتخابی

متدهای انتخاب در آزمون رگرسیون که به اختصار RTS گفته می‌شوند، هر بار، با انتخاب زیرمجموعه‌ای از کل موارد آزمون موجود، بر اساس عواملی مانند تغییرات رخ داده در کد و رفتار اجرایی آزمون‌ها برای اجرا انتخاب می‌کنند و به این ترتیب، هزینه‌ی آزمون رگرسیون را به میزان قابل توجهی کاهش می‌دهند [WL92][RH97][MRE02][REM04]. در عین حال فنون انتخاب در آزمون رگرسیون، مجموعه آزمون کل را کاهش دائمی نمی‌دهند و موارد آزمون را حذف نمی‌کنند. مسأله‌ی اجرای مجدد انتخابی موارد آزمون را می‌توان به‌صورت زیر بیان کرد [RH97]:

^۱ Obsolete - مورد آزمون t مهجور نامیده می‌شود هرگاه در نسخه‌ی جدید، ویژگی تحت آزمون توسط t وجود نداشته باشد یا دچار تغییر شده باشد، به‌طوری که اجرای مورد آزمون فوق در آزمون رگرسیون برای نسخه جدید، زاید باشد. می‌توان گفت مورد آزمون t مهجور است اگر و تنها اگر حاوی ورودی آزمونی باشد که برای نسخه جدید برنامه، نامعتبر باشد و یا با توجه به حالت جدید مشخصه‌های برنامه، رابطه ورودی-خروجی نامعتبری برای نسخه‌ی جدید برنامه داشته باشد [RH97].

فرض: اگر T مجموعه‌ی کل موارد آزمون موجود، P نسخه‌ی اولیه‌ی برنامه و P' نسخه‌ی جدید آن باشد؛

۱. زیرمجموعه‌ی $T' \subset T$ را مجموعه‌ای از موارد آزمون برای اجرا روی P' در نظر می‌گیریم.
۲. با آزمودن P' توسط T' ، تصحیحات مورد نیاز P' نسبت به T' مشخص می‌شود.
۳. در صورت نیاز، T'' ایجاد می‌شود، که مجموعه‌ای از آزمون‌های عملکردی یا ساختاری جدید برای P' هستند.
۴. با آزمودن P' توسط T'' ، تصحیحات مورد نیاز P' نسبت به T'' مشخص می‌شود.
۵. سابقه‌ی آزمون و نیز T''' که دنباله آزمون جدیدی برای P' است، از T' ، T'' و T''' برای آن ایجاد می‌شوند.

در توضیح گام‌های بالا، می‌توان گفت که اجرای مجدد انتخابی موارد آزمون، پاره‌ای مسایل را مد نظر قرار می‌دهد: در گام (۱) که شامل مسأله‌ی انتخاب موارد آزمون رگرسیونی (موارد آزمون برای بخش‌های دگرگونی یافته) است؛ که به صورت انتخاب زیرمجموعه‌ی T' از T برای آزمودن نسخه‌ی جدید P' بیان می‌شود. این مسأله، همچنین شامل زیر مسأله‌های تعیین آزمون‌هایی در T است که منسوخ شده‌اند. گام (۳) نیز به بیان مسأله‌ی تعیین پوشش می‌پردازد؛ که مسأله‌ی تعیین بخش‌هایی از نسخه‌ی جدید P' یا مجموعه‌ی مشخصه‌های جدید S' است که نیاز به آزمون بیشتر دارند. گام (۲) و (۴)، بر مسأله‌ی اجرای دنباله آزمون دلالت دارند؛ که مسأله‌ی اجرای کارای آزمون‌ها و بررسی نتایج آزمون‌ها برای تصحیح برنامه است. گام (۵) مسأله‌ی نگهداری دنباله آزمون را بیان می‌کند؛ که مسأله‌ی بروزرسانی و مرتب‌سازی اطلاعات آزمون است.

به بیان ساده‌تر، فنون انتخاب در آزمون رگرسیون در هر مرحله، به انتخاب یک زیر مجموعه 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 اگر ملاک انتخاب [RH97]، رشته آزمونی با پوشش کامل کد باشد، این انتخاب را انتخاب امن در آزمون رگرسیون^۱ گویند. مسأله‌ی بسیار مهم در مورد این فنون، موازنه‌ی بین ایمنی و کارایی^۲ است. فنون RTS ایمن، تحت شرایط به خصوص، تضمین می‌کند که موارد آزمون انتخاب نشده، در اجرا بر روی نسخه‌ی جدید نمی‌توانستند خطایی آشکار کنند. استراتژی اکثر

¹ Safe Regression Test Selection

² Efficiency

فنون RTS امن، این است که با هر بار تغییر در نسخه و نیاز به تکرار آزمون رگرسیون، با بررسی و مقایسه‌ی گراف جریان کنترلی نسخه‌ی جدید و نسخه‌ی قبل، موارد آزمون انتخاب می‌شوند که بخش‌های اصلاح شده در نسخه‌ی جدید را می‌آزمایند [RH97]. مطالعات نشان داده‌اند که میزان تغییرات بین دو نسخه، به شدت کارایی فنون آزمون رگرسیون انتخابی را تحت تأثیر قرار می‌دهد [KPR05].

معمولاً فنون انتخاب موارد آزمون براساس شناسایی تغییرات و میزان تأثیر آن‌ها می‌باشد که با استفاده از فنونی همچون تفاوت معنایی [Bnk97]، تفاوت متنی [VFr97]، عملیات برش‌بندی [GHS96]، فوق داده‌های مؤلفه‌ای [ODR07] و گراف‌های رابطه‌ای بین کلاسی [OSH04] صورت می‌پذیرد.

از آن جا که انتخاب دقیق موارد آزمون آشکار کننده‌ی خطا، که ایده‌آل آزمون رگرسیون انتخابی است، ممکن نیست، چالش اصلی که آزمونگران در تمام فنون آزمون رگرسیون انتخابی با آن مواجهند، موازنه‌ی بین منفعت- هزینه در مورد این فنون است [KP02].

۱-۷- اولویت‌دهی موارد آزمون

ترتیب‌دهی موارد آزمون بر اساس یک معیار شایستگی، برای دستیابی سریع‌تر به هدف آزمون که عموماً، کشف سریع‌تر خطاها و یا دستیابی به پوشش سریع‌تر کد نرم‌افزار است، اولویت‌دهی موارد آزمون نامیده می‌شود [RUC99][RUC01][EMR01]. به بیان ساده، فن اولویت‌دهی موارد آزمون در حالت کلی، کل موارد آزمون موجود در مجموعه آزمون را نگه می‌دارد؛ در عین حال پیش از اجرای موارد آزمون، آن‌ها را مرتب می‌کند تا به این صورت در دستیابی سریع‌تر به اهداف آزمون، به آزمونگران کمک کند.

تعریف رسمی مسأله‌ی اولویت‌دهی موارد آزمون نخستین بار توسط رودرمل^۱ و گروهش در سال ۲۰۰۱ ارائه شد [RUC01]، به این ترتیب که اگر T را رشته آزمون، PT را مجموعه‌ی تمامی جایگشت‌های موارد آزمون موجود در T و f را تابعی از PT بر روی مجموعه‌ی اعداد حقیقی در نظر بگیریم، مسأله‌ی اولویت‌دهی موارد آزمون به صورت زیر تعریف می‌شود:

¹ Rothermel

مسأله: $T' \in PT$ را به گونه‌ای بیابید که:

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

که در اینجا PT بیانگر تمامی اولویت‌های (ترتیب‌ها) ممکن و f به هر کدام از این ترتیب‌های موجود اعمال شود و مقدار عددی به دست آمده، نشان‌دهنده‌ی اولویت آن ترتیب در میان سایر ترتیب‌ها (بر اساس معیار شایستگی^۱) در نظر گرفته شده در f است.

فضای جستجو در مسأله‌ی اولویت‌دهی موارد آزمون، در حقیقت جایگشتی از کلیه‌ی موارد آزمون موجود در رشته آزمون است؛ به گونه‌ای که این ترتیب اجرا، آشکارکننده‌ی حداکثر خطاهای ممکن باشد. بنابراین مسأله‌ی اولویت‌دهی موارد آزمون، در حقیقت یک مسأله‌ی جستجو است که در برخی مراجع این مسأله را معادل با مسأله‌ی کوله‌پشتی صفر و یک دانسته‌اند، که مسأله‌ی فوق، NP سخت و بدون راه‌حل قطعی است. لذا فنون ارائه شده برای مسأله‌ی اولویت‌دهی موارد آزمون مکاشفه‌ای بوده و با استفاده از فنون وزن‌دهی و معیارهای پوشش^۲ گوناگون، برای بهبود سرعت کشف خطا و کارایی بیشتر آزمون با هم رقابت می‌کنند و هیچ‌یک در حالت کلی بر دیگری برتری ندارند [LHH07][KKT07][ERK04][KP02][EMR02][RUC01].

۱-۸- 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 بی‌رویه‌ی مجموعه آزمون می‌تواند مشکل‌ساز 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 کماکان همه‌ی نیازمندی‌های آزمون را که مجموعه‌ی اولیه تأمین می‌کرد، برآورده می‌نماید. این مسأله به "کاهش مجموعه آزمون" (کمینه‌سازی مجموعه آزمون یا پالایش موارد آزمون) مشهور بوده و زیرمجموعه‌ی حاصل،

¹ Fitness Metric

² Coverage Criteria

مجموعه‌ی نماینده^۱ یا شاخص نامیده می‌شود. اگر هیچ زیرمجموعه‌ای از مجموعه‌ی نماینده، تمام نیازمندی‌ها را تأمین نکند، آنرا مجموعه‌ی بهینه یا مجموعه‌ی کمینه می‌نامیم [RHR02][WHL97].

تعریف رسمی مسأله‌ی کمینه‌سازی مجموعه آزمون به صورت زیر است [HGS93]:

فرض: یک مجموعه آزمون T از موارد آزمون $\{t_1, t_2, \dots, t_m\}$ ، یک مجموعه از نیازمندی‌های آزمون $\{r_1, r_2, \dots, r_n\}$ ، که باید برآورده شوند تا پوشش مطلوب از برنامه حاصل شود، و زیرمجموعه‌های $\{t_1, t_2, \dots, t_m\}$ از T که هر کدام به یکی از r_i ها منتسب است به قسمی که هر یک از موارد آزمون t_j متعلق به T_i ، r_i را تأمین نماید.

مسأله: زیرمجموعه‌ی کمینه‌ای از T پیدا کنید که همه‌ی r_i هایی که مجموعه‌ی غیر کمینه‌ی T تأمین می‌کند را برآورده نماید.

فنون کاهش مجموعه آزمون، با تشخیص و حذف موارد آزمون افزونه، دنباله آزمون را به طور دائمی کاهش می‌دهند (برخلاف فنون آزمون رگرسیون انتخابی). اما این کاهش ممکن است در بلند مدت بر کیفیت نرم‌افزار تأثیر سوء داشته باشد. از سوی دیگر، کاهش تعداد موارد آزمون به گونه‌ای که همه‌ی نیازمندی‌های آزمون را برآورده سازد، سبب افزایش کارایی فرایند آزمون می‌شود. علت این مسأله این است که زمان اجرای موارد آزمون کمتر شده و نهایتاً منجر به کاهش زمان کل آزمون می‌شود. بنابراین مسأله‌ی کاهش مجموعه آزمون در واقع به مسأله‌ی یافتن یک زیرمجموعه از مجموعه‌ی اصلی که بتواند تمام نیازمندی‌های آزمون را همانند مجموعه‌ی اصلی برآورده سازد، تبدیل می‌شود؛ این کاهش باید به گونه‌ای انجام شود که قابلیت آشکارکنندگی خطای مجموعه‌ی حاصل نسبت به مجموعه آزمون اولیه، بدون تغییر یا با کاهشی قابل اغماض (نسبت به میزان کاهش به دست آمده در زمان و منابع آزمون ناشی از کاهش مجموعه آزمون) همراه باشد. برقراری توازن بین کاهش توان آشکارسازی خطای مجموعه آزمون حاصل نسبت به کاهش به دست آمده در زمان و منابع مصرفی به دلیل کاهش حجم آزمون‌ها، همواره موضوعی چالشی بوده است. اصولاً در مورد کاهش مجموعه آزمون، دو هزینه اهمیت دارد [RHR02]:

۱. هزینه‌ی اجرای فرایند (ابزار) کاهش مجموعه آزمون برای تولید مجموعه‌ی کاهش یافته: یک ابزار کاهش مجموعه آزمون را می‌توان در پی ارائه نسخه‌های نرم‌افزار اجرا کرد و اجرای آن می‌تواند

¹ Representative Set

به صورت خودکار و در ساعات خلوتی^۱ انجام شود و در این صورت هزینه‌ی اجرای ابزار خیلی مشکل‌زا نخواهد بود. علاوه بر این هنگامی که با استفاده از یک ابزار، مجموعه آزمون کاهش داده شد، هزینه‌ی این کاهش به تدریج در اجراهای 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 اجرا منجر به آشکارسازی خطا می‌شوند: این هزینه از مورد قبلی مهم‌تر است. حذف چنین موارد آزمونی موجب کاهش درصد کشف خطای مجموعه آزمون شده و از کارایی آن می‌کاهد. اثر این کاهش به تدریج با بکارگیری مجدد آن در نسخه‌های بعدی نرم‌افزار افزایش یافته تا جایی که حجم خطاهای از دست‌رفته زیاد شده و فرایند آزمون دچار اختلال و عدم کارایی می‌گردد. دو روش برای بررسی هزینه‌های مربوط به حذف موارد آزمون آشکار کننده خطا وجود دارد:

أ. بر مبنای هر مورد آزمون^۲: با داشتن برنامه خطادار P و مجموعه آزمون T یکی از راه‌های اندازه‌گیری هزینه‌ی مجموعه آزمون کاهش یافته بر کشف خطا، شناسایی موارد آزمونی است که در T وجود دارند و خطاهایی را از P آشکار می‌کنند اما در T_{min} نیستند. این کمیت را می‌توان با تعداد موارد آزمون آشکار کننده خطا در T نرمال 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 خطا داشته باشند.

ب. بر مبنای هر مجموعه آزمون^۳: 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 خطا آشکار نمی‌کند. (۲) بعضی موارد آزمون هم در T و هم در T_{min} هستند که خطاهایی را آشکار می‌نمایند. (۳) بعضی از موارد آزمون در T آشکارکننده‌ی خطا هستند اما در T_{min} هیچ مورد آزمون آشکارکننده‌ی خطا وجود ندارد. مورد ۱ حالتی را نشان می‌دهد که T بی‌اثر است. حالت ۲ حالتی است که کاهش مجموعه موجب کاهش درصد کشف

^۱ Off-peak

^۲ On a per-test-case basis

^۳ On a per-test-suite basis

خطا نمی شود و حالت سوم حالتی است که کاهش مجموعه آزمون باعث می شود کشف خطا با اختلال مواجه شود. توجه کنید که خطاهایی که توسط اکثر موارد آزمون قابل کشف باشد، پس از کمینه سازی نیز می توان کشف نمود در حالی که خطاهایی که توسط تعداد معدودی از موارد آزمون قابل کشف باشد، پس از کمینه سازی در معرض عدم کشف قرار می گیرند.

عموماً کاهش مجموعه آزمون یک مسأله ی بهینه سازی تک هدفه^۱ است. اما در برخی بررسی های اخیر کاهش مجموعه آزمون چندهدفه^۲ نیز مورد توجه قرار گرفته است که برخی از این اهداف پوشش کد، تاریخچه ی کشف خطا در گذشته و هزینه ی اجرایی هستند [YHr07][BMK04][RHO98].

در کل مسأله کاهش مجموعه آزمون ها و انتخاب موارد آزمون از این جهت که زیر مجموعه ای از مجموعه اصلی را تولید می کنند شبیه یکدیگر می باشند. معیار کاهش مجموعه آزمون این است که آیا زیر مجموعه ی منتخب، همه نیازمندی های آزمون را تأمین می کند در حالی که تمرکز انتخاب موارد آزمون بر بخش های تغییر یافته سیستم نرم افزاری است. در بعضی از تقسیم بندی های اولیه [LWh90]، کاهش مجموعه آزمون در حوزه ی انتخاب موارد آزمون دسته بندی می شود. بعضی از روش هایی که ارائه شده سعی می نمایند تا مسایل را به صورت ترکیبی حل نمایند. مثلاً روش ارائه شده توسط والکات^۳ و همکاران [WSK06] از الگوریتم های ژنتیک استفاده می کند تا مسأله اولویت دهی و انتخاب موارد آزمون را همزمان حل نماید. یا تحقیقی که در [LHH07] انجام شده مجموعه ای روش های ابتکاری و الگوریتم های ژنتیک را برای حل مسأله ی کاهش مجموعه آزمون و اولویت دهی موارد آزمون ارائه نموده است.

از آن جایی که هر مورد آزمون می تواند یک یا چند نیازمندی آزمون را برآورده نماید، ارتباط میان آن ها را می توان با یک ماتریس بنام ماتریس مورد آزمون-نیازمندی نمایش داد. مثالی از آنرا در جدول ۱-۱ مشاهده می نمایید.

¹ Single Objective

² Multi Objective

³ Walcott

جدول (۱-۱) ماتریس مورد آزمون-نیازمندی

نیازمندی/مورد آزمون	نیازمندی ۱	نیازمندی ۲	نیازمندی ۳	نیازمندی ۴	نیازمندی ۵
مورد آزمون ۱	۰	۱	۱	۱	۱
مورد آزمون ۲	۱	۰	۰	۰	۰
مورد آزمون ۳	۰	۱	۰	۰	۱
مورد آزمون ۴	۱	۰	۰	۰	۰

مجموعه‌ی {مورد آزمون ۱، مورد آزمون ۲، مورد آزمون ۳، مورد آزمون ۴} معرف موارد آزمون و مجموعه‌ی {نیازمندی ۱، نیازمندی ۲، نیازمندی ۳، نیازمندی ۴} معرف نیازمندی‌های آزمون می‌باشد. عدد ۱ در سطر i و ستون j بیان می‌کند که مورد آزمون i می‌تواند نیازمندی j 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 می‌کنند که کماکان همه‌ی نیازمندی‌های آزمون را برآورده نماید. مسأله یافتن یک مجموعه نماینده‌ی بهینه در حالت کلی NP کامل است [Jhn74][AHU74] و روش‌های موجود مجموعه‌های نیمه بهینه تولید می‌کنند.

۱-۹- افزایش مجموعه آزمون

انواع رویکردهای مورد استفاده در آزمون رگرسیون (انتخاب، اولویت‌دهی و کاهش) از مجموعه آزمون موجود استفاده مجدد می‌نمایند. اما حین آزمون رگرسیون مسأله دیگری بروز می‌کند. کدی که تغییر می‌کند، ممکن است حاوی بخش‌های جدیدی باشد که مجموعه آزمون موجود آن بخش‌ها را نمی‌آزماید. بنابراین یکی از فعالیت‌های توسعه دهندگان هنگام آزمون رگرسیون سنجش کیفیت مجموعه آزمون بعد از تغییر برنامه است. هدف این است که رفتارهای جدید یا تغییر کرده که با مجموعه آزمون موجود آزموده نمی‌شوند و نیازمند موردهای آزمون جدید هستند، شناسایی شود. لذا برای افزایش سودمندی مجموعه آزمون در پاسخ به تغییرات، دو کار باید انجام شود [SCA08]:

- واریسی شود آیا مجموعه آزمون‌های موجود برای تغییرات ایجاد شده در برنامه کافی

هستند.

- اگر کافی نیستند، باید موردهای آزمون جدیدی تولید شوند که هدفشان آزمون رفتار تغییر یافته برنامه باشد. پس نیازمندی روشی هستیم که آزمونگر را در تولید این مجموعه آزمون‌ها هدایت کند.
- این مسأله را افزایش مجموعه آزمون می‌نامیم.

۱۰-۱- رویکرد کلی برای طراحی الگوریتم‌های کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون

الگوریتم‌های کاهش مجموعه آزمون، به‌عنوان ورودی مجموعه آزمون T را گرفته و در خروجی یک مجموعه آزمون جدید تولید می‌کنند در حالی که الگوریتم‌های اولویت‌دهی مجموعه T را گرفته و یک توالی از موارد آزمون را در خروجی می‌سازند. برای راحتی در بحثی که در ادامه می‌آید، هر دو خروجی را T' می‌نامیم. الگوریتم‌های کمینه‌سازی مجموعه آزمون و اولویت‌دهی موارد آزمون صفات مشترکی دارند [JHr03].

۱. همه الگوریتم‌ها در ساخت T' میزان سهم و مفید بودن^۱ هر یک از موارد آزمون را بر اساس خصوصیات برنامه تحت آزمون P ، خصوصیات موارد آزمون در T و هدف T' مورد ارزیابی قرار می‌دهند. با وجودی که خصوصیات متنوعی مطرح شده‌اند اما همگی را می‌توان به دو نوع پوشش و هزینه تقسیم‌بندی نمود. پوشش، نیازمندی‌های برنامه P ، معیارهای مبتنی بر کد مثل دستور، انشعاب و جریان داده‌ها، خطرات، سالمندی و خطاخیز بودن مؤلفه‌های P را در بر می‌گیرد. هزینه نیز شامل زمان اجرای P به ازاء یک مورد آزمون، زمان بر پایی و آماده‌سازی برای اجرای یک مورد آزمون و هر هزینه دیگری در ارتباط با اجرای یک مورد آزمون (به‌خصوص در حالتی که از شبیه‌سازی استفاده شده باشد) می‌باشد. الگوریتم‌ها موارد آزمون T را بر اساس ترکیبی از این خصوصیات ارزیابی می‌نمایند. مثلاً مرجع [HGS93] و [RHO98] انواع مختلفی از پوشش را جهت کاهش مجموعه آزمون به کار گرفته‌اند. فن آن‌ها هنگام تصمیم‌گیری درباره‌ی موارد آزمونی که باید در T' قرار گیرند، به ازاء هر یک از

¹ Contribution and goodness

موجودیت‌های برنامه، E ، موارد آزمونی را انتخاب می‌کند که E را بپوشاند. به‌عنوان مثالی دیگر، مرجع [RUC01] برای ایجاد ترتیبی از موارد آزمون در T' ، انواع مختلفی از پوشش را همراه با میزان خطاخیز بودن پیمانها مورد استفاده قرار داده‌اند.

۲. الگوریتم‌های کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون، در دفعاتی (فرکانسی) که سهم یک مورد آزمون را مجدداً محاسبه می‌نمایند، با هم فرق می‌کنند. یک روش ممکن است این سهم را یکبار محاسبه کرده و از آن برای شمول یا عدم شمول موارد آزمون در T' استفاده کند. در مرجع [RUC01] این فرکانس را جمع کل^۱ نامیده شده و از آن برای محاسبه‌ی اولویت‌دهی مبتنی بر جمع^۲ استفاده شده است. روش دیگر، سهم هر مورد آزمون را بر اساس سهم افزوده‌ی آن نسبت به T' پس از شمول یا عدم شمول آن در T' محاسبه مجدد می‌نماید. این فرکانس، افزوده نامگذاری شده [RUC01] و از آن برای محاسبه اولویت‌دهی مبتنی بر افزایش^۳ استفاده می‌شود.

۳. یک الگوریتم می‌تواند از سهم مورد آزمون برای تعیین مورد آزمونی که باید به T' اضافه شود استفاده کند (مورد استفاده‌ی کاهش مجموعه آزمون و اولویت‌دهی موارد آزمون) و یا از آن در جهت تعیین مورد آزمونی که باید از T حذف شود، بهره‌برد (مورد استفاده‌ی کاهش مجموعه آزمون). اولی را فن افزایشی^۴ و دومی را فن تجزیه^۵ می‌نامند. فن افزایشی کار خود را با مجموعه‌ی خالی T' شروع می‌کند و موارد آزمون را یکی پس 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 T' به دست آید. مهم‌ترین مزیت روش تجزیه در کاهش مجموعه آزمون، این است که می‌توان در حین فرایند کاهش، در هر زمان الگوریتم را متوقف نمود و مجموعه‌ی باقیمانده، نیازمندی‌های موارد آزمون را می‌پوشاند. در مقابل الگوریتم افزایشی تنها زمانی که خاتمه یابد می‌توان تضمین کرد که پوشش کامل حاصل شده است.

ترکیب این صفات یا به عبارتی محاسبه‌ی سهم هر مورد آزمون، فرکانس ارزیابی (مجدد) سهم هر مورد آزمون و روش ساخت T' ، می‌تواند الگوریتم‌های متفاوتی تولید نماید.

¹ Total

² Total-based

³ Additional-based

⁴ Build-up

⁵ Break-down

۱-۱۱- نتیجه گیری

آزمون رگرسیون نرم افزاری، از فعالیت‌های نگهداری نرم افزار برای برنامه‌ی اصلاح شده است، تا اطمینان حاصل شود که تغییرات صحیح هستند و تأثیر عکس (نامطلوب) بر روی بخش‌های تغییرنیافته‌ی برنامه نداشته‌اند. برای حل مشکل هزینه‌ی زیاد آزمون رگرسیون، متدولوژی‌های مختلفی ارائه شده‌اند 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 فرایند (ابزار) کاهش مجموعه آزمون برای تولید مجموعه‌ی کاهش یافته و (۲) حذف برخی از موارد آزمون که در صورت اجرا منجر به آشکارسازی خطا می‌شوند. هزینه‌ی دوم اهمیت بیشتری دارد. پوشش و هزینه، دو خصوصیت مهم موارد آزمون است که الگوریتم‌های کاهش در ساخت مجموعه‌ی کاهش یافته میزان سهم و مفید بودن هر یک از موارد آزمون را بر اساس آن می‌سنجند.

همانگی فنون آزمون رگرسیون نرم افزار با تولید مورد آزمون، آزمون رگرسیون چند هدفه، مسأله اوراکل‌های آزمون و هزینه آن‌ها، سایر دامنه‌های کاربرد و آزمون رگرسیون غیر وظیفه‌مندی از مباحثی هستند که در مقاله بعدی مورد بررسی قرار می‌گیرند.