آزمون رگرسیون نرم‌افزار: مفاهیم و روش‌ها

[*]علیرضا خلیلیان[1]، دانشجوی دکتری نرم‌افزار دانشگاه اصفهان و عضو آزمایشگاه آزمون و تأیید نرم‌افزار شهید بهشتی
[†]مجتبی وحیدی اصل و [‡]حسن حقیقی، استادیار دانشکده مهندسی برق و کامپیوتر شهید بهشتی و سرپرست آزمایشگاه آزمون و تأیید نرم‌افزار دانشگاه شهید بهشتی
[*]khalilian@eng.ui.ac.ir, {[†]mo_vahidi, [‡]h_haghighi}@sbu.ac.ir

۱-۱- مقدمه

آزمون رگرسیون[۲] نرم‌افزار، با مفهوم **رگرسیون نرم‌افزار** سروکار دارد. رگرسیون نرم‌افزار عبارت است از ایراد نرم‌افزاری، که سبب توقف عملکرد یک ویژگی می‌شود؛ به‌طوریکه بعد از یک رویداد به‌خصوص (به‌عنوان مثال بروزرسانی یا تولید وصله‌های نرم‌افزاری[۳]) باید این ویژگی انجام شود. به‌عبارت دیگر، بعد از اصلاح نرم‌افزار 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 می‌شود که خطای رگرسیونی اتفاق افتاده است. آزمون رگرسیون، به هر قسم از آزمون نرم‌افزاری اطلاق می‌شود که هدف آن، آشکارسازی رگرسیون‌های نرم‌افزاری باشد. این رگرسیون‌ها، هر زمان که عملکرد نرم‌افزار، که پیش‌تر به‌درستی کار می‌کرد، دچار توقف در این عملکرد مورد نظر شود، رخ می‌دهند. عموماً رگرسیون‌ها، نتیجه‌ی تأثیرات نامطلوب ناشی از تغییرات در نرم‌افزار 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 نرم‌افزار تشریح می‌شود. بخش سوم انواع آزمون رگرسیون و آزمایه‌ها[۴] را ارائه می‌دهد. سپس درک و اندازه‌گیری نرم‌افزار در بخش چهارم مورد بحث قرار می‌گیرد. آزمون جامع کامل، آزمون رگرسیون انتخابی، اولویت‌دهی آزمایه‌ها، کاهش مجموعه آزمون و افزایش مجموعه آزمون موضوع بخش‌های پنجم تا نهم را تشکیل می‌دهند. در بخش دهم رویکرد کلی برای طراحی فنون کاهش و اولویت‌دهی مورد بحث قرار می‌گیرد. بخش یازدهم هم به نتیجه گیری اختصاص دارد.

۱-۲- آزمون رگرسیون نرم‌افزار

چه در جریان توسعه‌ی 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 نیازمندی‌های محصول برای تطبیق با

---

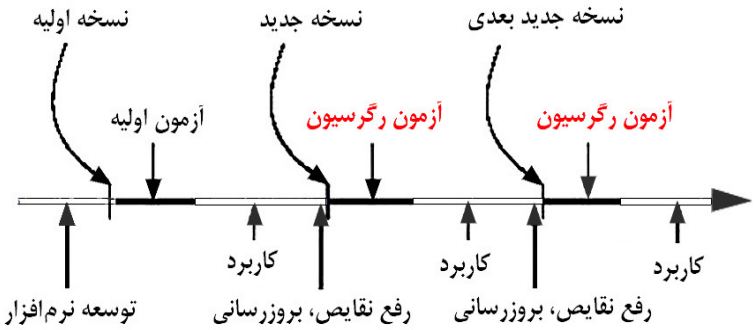
[1] www.khalilian.net

[۲] پس‌نمایی

[3] System Patching

[4] Test case

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 به‌طوری که رفتار آزمون شده سیستم نرم‌افزاری تنزل کرده[1] و پسرفت نماید. نتیجه این فرایند، ایجاد اشکالاتی موسوم به خطاهای پسرفت (خطاهای رگرسیونی[2]) خواهد بود. با وجود این‌که 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 آن‌جا 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[3]" برای اطمینان از صحت عملکرد تمام بخش‌های نرم‌افزار، اعم از تغییریافته و بدون تغییر، ضروری است. اهمیت این موضوع زمانی آشکار می‌شود که بدانیم عدم افت[4] کیفیت نرم‌افزارِ آزمون‌شده 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. جایگاه آزمون رگرسیون در چرخه‌ی توسعه نرم‌افزار در شکل ۱ نشان داده شده است.



شکل ۱: جایگاه آزمون رگرسیون در چرخه‌ی توسعه‌ی نرم‌افزار

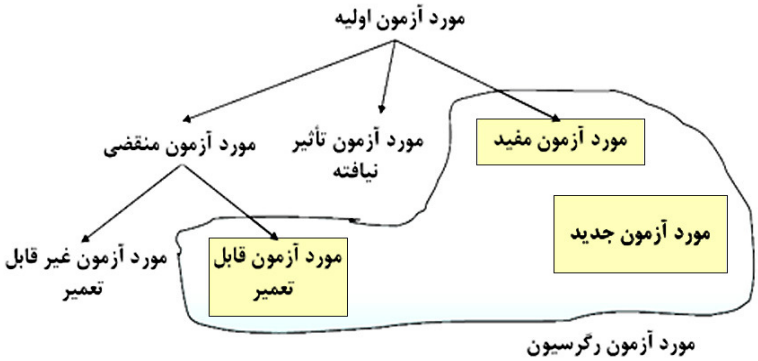
آزمون رگرسیون نرم‌افزاری، دو فاز مجزا دارد: فاز اولیه و فاز بحرانی. فاز اولیه‌ی آزمون رگرسیون نرم‌افزاری، پس از تحویل یک نسخه‌ی جدید از نرم‌افزار آغاز می‌شود؛ در خلال این فاز، برنامه‌نویسان به بهبود و تصحیح نرم‌افزار می‌پردازند. پس از تکمیل تصحیحات، فاز دوم یا بحرانی آزمون رگرسیون نرم‌افزاری آغاز می‌شود؛ 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 به‌دلیل 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[5] و اطلاعات تحلیل برنامه که در جریان فاز اولیه جمع آوری شده است، برای دستیابی به کاهش هزینه‌ها و انتخاب آزمایه‌های مناسب، بهره گیرد.

---

[1] Regress
[2] Regression Bugs
[3] Exhaustive Test
[4] Nonregression
[5] Test history

علاوه بر این، سبک‌های جدید توسعه‌ی نرم‌افزار چالش‌های بیشتری را نیز برای انجام کارامد آزمون رگرسیون ایجاد می‌کند. برای نمونه در توسعه نرم‌افزار مبتنی بر مؤلفه[1] از تعداد بسیاری مؤلفه‌های از پیش تولید شده‌ی نرم‌افزاری به‌صورت آماده و جعبه سیاه استفاده می‌شود که 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 آزمایه‌ها بر روی نرم‌افزار و نیز هزینه‌های منابع و نیروی انسانی صرف شده برای اجرای آزمایه‌ها در هر بار اجرای آزمون رگرسیون است.

با تحول نرم‌افزار و برای هر نسخه‌ی جدید، هر بار آزمایه جدیدی به مجموعه آزمون افزوده می‌شوند. به‌عنوان مثال، برای یک نرم‌افزار صنعتی، بیش از سه هفته طول می‌کشد تا تمام دنباله آزمون (مجموعه‌ی مرتب آزمایه‌ها برای اجرا) برای یک برنامه با ۲۰ هزار خط کد اجرا شوند. 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 ۲). بدیهی است که کنار گذاشتن تصادفی آزمایه‌ها، ریسک فراوانی دارد و قابلیت اطمینان نسخه‌ی جدید را شدیداً در مخاطره قرار می‌دهد. لذا یافتن مجموعه‌ی کوچکتر و غیرافزونه از آزمایه‌ها، که بتواند به‌طور مؤثر و کارا نرم‌افزار را بیازماید، از اهمیت زیادی برخوردار است. به‌دلیل تکرارهای زیاد، آزمون رگرسیون نرم‌افزار، بخش زیادی از هزینه‌های نگهداری نرم‌افزار را به خود اختصاص می‌دهد. به‌طوری‌که، تخمین زده می‌شود که آزمون رگرسیون ممکن است در اثر اجراهای مکرر، تقریباً نیمی از کل هزینه‌ی نگهداری نرم‌افزار را به‌خود اختصاص دهد. 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 آزمایه‌ها را برای آزمون مؤثر بر روی نسخه‌ی جدید انتخاب کنند؟



شکل ۲: تأثیر اصلاحات نرم‌افزار بر آزمایه‌ها در آزمون رگرسیون نرم‌افزار

برای به‌صرفه بودن آزمون رگرسیون، آزمونگران سعی می‌کنند که دنباله آزمون‌ها را ذخیره کنند که برای اجراهای

[1] Component-based Software Engineering

مجدد در مراحل بعدی آزمون رگرسیون از آن‌ها استفاده کنند. در این زمینه فنون و متدولوژی‌های گوناگونی در دست مطالعه قرار گرفته‌اند. یک راه‌کار ممکن، عبارت است از تولید و نگهداری مجموعه آزمون‌های باکیفیت که در خلال توسعه‌ی نسخ متعدد نرم‌افزار صورت می‌گیرد. چنین مجموعه‌ای را می‌توان با حذف آزمایه‌های منسوخ[1] یا شناسایی و علامت‌گذاری آزمایه‌های افزونه به‌دست آورد. آزمایه‌های افزونه با انواع منسوخ آن کاملاً متفاوت هستند. این تفاوت از آن‌جایی ناشی می‌شود که آزمون‌های افزونه اجرایی هستند اما با درنظر گرفتن یک معیار آزمون به‌خصوص اهمیت چندانی ندارند. مثلاً آزمایه‌هایی که مسیرهای یکسانی را می‌پوشانند با توجه به معیارهای ساختاری افزونه هستند اما آزمایه‌هایی که با بخش‌های یکسانی تطابق دارند، از لحاظ عملکردی افزونه هستند. آزمایه‌های افزونه ممکن است در اثر فعالیت همزمان دو طراح آزمون یا تغییرات همزمان در کد ایجاد شوند. این‌گونه آزمون‌ها کارایی کلی فرایند آزمون را تحت تأثیر قرار نمی‌دهند، اما تأثیر شدیدی بر موازنه‌ی میان هزینه و منفعت آزمون می‌گذارند. آزمایه‌های افزونه نه تنها خطایی آشکار نمی‌سازند، بلکه هزینه‌های فرایند آزمون را نیز افزایش می‌دهند. با این وجود، این‌گونه آزمایه‌ها دور انداخته نمی‌شوند زیرا ممکن است در آزمون نسخ آتی نرم‌افزار مفید باشند. 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 آزمایه‌های 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**[2]، **آزمون رگرسیون انتخابی**[3]، **کاهش (کمینه‌سازی) مجموعه آزمون**[4] و **اولویت‌دهی آزمایه‌ها**[5]. خانواده‌ای از فنون که با رویکرد ایجاد آزمایه‌های جدید کار می‌کنند، **افزایش مجموعه آزمون**[6] نام دارند.

۱-۳- انواع آزمون رگرسیون و آزمایه‌ها

آزمون رگرسیون را می‌توان به دو دسته پیشرو[7] و اصلاحی[8] تقسیم کرد. 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 آزمایه‌های موجود می‌توان از آن‌ها به‌منظور آزمون برنامه تغییر کرده استفاده کرد.

---

[1] Obsolete
[2] Retest All
[3] Regression Test Selection
[4] Test Suite Reduction (Minimization)
[5] Test Case Prioritization
[6] Test Suite Augmentation
[7] Progressive
[8] Corrective

آزمایه‌ها را می‌توان به پنج دسته تقسیم کرد که دسته‌های اول تا سوم حاوی آزمایه‌های داخل مجموعه آزمون فعلی هستند. دسته‌های چهارم و پنجم آزمایه‌هایی هستند که برای آزمون برنامه‌ی تغییر یافته باید تولید شوند.

۱. **قابل استفاده مجدد**[1]: آزمایه‌هایی هستند که فقط بخش‌هایی از برنامه را که بین دو نسخه تغییر نکرده‌اند، می‌آزمایند. اجرای این آزمایه‌ها برای آزمون نسخه تغییریافته برنامه ضروری نیست ولی آن‌ها را در مجموعه آزمون نگه می‌دارند زیرا برای آزمون رگرسیون نسخه‌های بعدی برنامه لازم می‌شوند.

۲. **قابل آزمون مجدد**[2]: آزمایه‌هایی هستند که بخش‌های تغییر کرده از برنامه فعلی در نسخه جدید آن‌را می‌آزمایند. بنابراین برای آزمون نسخه تغییر کرده، این آزمایه‌ها حتماً باید اجرا شوند.

۳. **منسوخ**[3]: آزمایه‌هایی 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. آزمایه‌هایی ساختاری هستند که دیگر سهمی در پوشش ساختاری برنامه ندارند.

۴. **ساختاری جدید**[4]: ساختارهای برنامه تغییریافته را می‌آزمایند و پوشش ساختاری در بخش‌های تغییر یافته نسخه جدید برنامه ایجاد می‌کنند.

۵. **مشخصات جدید**[5]: آزمایه‌های جدیدی هستند که مشخصات برنامه تغییریافته را می‌آزمایند؛ یعنی کدهای جدید تولید شده حاصل از بخش‌هاتی تغییریافته مشخصات نسخه جدید.

فنون آزمون رگرسیون انتخابی آزمایه‌های قابل آزمون مجدد را شناسایی می‌کنند. فنون کمینه‌سازی مجموعه آزمون آزمایه‌های منسوخ را شناسایی می‌نمایند. فنون اولویت‌دهی هم در حقیقت روشی پیشرفته برای ساخت یک طرح آزمون هستند.

۱-۴- درک و اندازه‌گیری نرم‌افزار

اندازه‌گیری و درک نرم‌افزار، دو جنبه‌ی مهندسی نرم‌افزار 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 این‌رو در بخش جاری در حد ضرورت، اطلاعاتی را پیرامون این دو جنبه که با این تحقیق ارتباط دارند، بیان می‌نماییم.

یکی از طرقی که مهندسین می‌توانند از اندازه‌گیری بهره ببرند، اندازه‌گیری خصوصیات محصول فعلی و فرایندهای کنونی به‌منظور پیش‌بینی خصوصیات محصول آتی است. مدل‌های پیش‌بینی حاصل، کمک بسیاری در برطرف نمودن مسایل آزمون رگرسیون خواهد داشت. از این مدل‌ها می‌توان جهت تخمین خصوصیاتی که به خروجی

---

[1] Reusable
[2] Retestable
[3] Obsolete
[4] New-structural
[5] New-specification

آزمون رگرسیون مرتبط است، بهره برد و نیز می‌توان در برنامه‌ریزی این آزمون از آن‌ها استفاده کرد.
انتخاب فنون اندازه‌گیری 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 باصرفه) دردسترس قرار دارد، وابسته است. بر اساس آن‌چه از تعریف آزمون رگرسیون ارائه شد، می‌توانیم چند فن اندازه‌گیری مرتبط را مشخص نماییم.

۱. **تغییرات مصنوعات نرم‌افزاری:** رویداد مهمی است که منجر به ایجاد خطاهای رگرسیونی می‌گردد. بنابراین، طبیعی است که تاریخچه تکامل محصول را در نظرگرفته و بخش‌های تغییر یافته‌ای را که مورد توجه آزمون رگرسیون است، شناسایی نماییم.
۲. **اندازه‌گیری کیفیت نرم‌افزار:** زمینه‌ی تحقیقاتی دیگری است که از جنبه‌ی مفهومی به آزمون رگرسیون مربوط می‌شود. متریک‌های کیفیت را می‌توان به‌منظور تشخیص نواحی از کد که خطاخیر هستند[1]، بکار برد. در این‌صورت چنین نواحی را به‌طور وسیع‌تر در مرحله‌ی آزمون رگرسیون، تحت آزمون مجدد قرار می‌دهیم.
۳. **اندازه‌گیری اطلاعات پوشش آزمون‌ها:** مهم‌ترین جنبه‌ایست که اطلاعات دقیق و قاطعی را برای آزمون رگرسیون به‌دست می‌دهد. این اطلاعات در برنامه‌ریزی آزمون رگرسیون کمک می‌کنند که به پوشش مطلوب از مصنوعات نرم‌افزاری دست یابیم. استفاده از فنون تحلیل زمان اجرا همچون فراکدگذاری کد، ما را قادر می‌سازد تا میزان پوشش کد توسط هر آزمایه را اندازه‌گیری نماییم. اطلاعات پوشش کد، اساس اغلب فنون آزمون رگرسیون فعلی را تشکیل می‌دهند.

به‌واسطه‌ی اهمیتِ 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 یک معیار پوشش خاص تأمین می‌نمایند. با این وجود، مفهوم پوشش تنها با تولید آزمایه‌ها محدود نمی‌شود. فنون آزمون رگرسیون نیز عموماً اطلاعات پوشش را به‌کار می‌گیرند. در این حوزه، اندازه‌گیری پوشش آزمایه‌هایی که هم‌اکنون موجود هستند، مورد نظر است. با وجودی‌که 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 آزمایه به‌کار می‌روند (اجرا می‌شوند). استفاده از اندازه‌ی پوشش کد، نویدبخش این موضوع است که هرچه یک مجموعه آزمون کد بیشتری را بپوشاند، 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 این‌صورت پوشش آزمایه از یک اجرا تا اجرای دیگر بلا تغییر می‌ماند. چنین داده‌هایی خصوصیتِ اتکاپذیری از

[1] Fault-prone

آزمایه 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[1]، تعریف و استفاده و غیره معرفی شده است. به‌عنوان نمونه، پوشش دستورات مشخص می‌کند که آیا یک دستور مشخص از کد (یا نمایش خاصی از آن مثل بایت کد) توسط یک آزمایه به اجرا درآمده است یا خیر.

جهت اندازه‌گیری پوشش یک آزمایه، می‌توانیم مسیر اجرایی آنرا در برنامه هنگام اجرای آزمایه ردیابی[2] نماییم که معمولاً به آن الگوبرداری از برنامه[3] می‌گویند. این‌کار را می‌توان از طریق تغییر کد، با اضافه کردن کد اضافی که رویدادها را در ضمن اجرا ثبت می‌کند محقق نمود. این کد اضافی در فرایندی 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 مستقیمِ فایلهای اجرایی، عدم تأثیر بر ابزارهای سیستمی همچون کامپایلرها و پیوند دهنده‌ها[4] است. ضمن این‌که باعث تسهیل اندازه‌گیری مفصل‌تر شده و به‌دلیل عدم نیاز به کامپایل مجدد، کل فرایند را ساده‌تر می‌نمایند. به‌طور کلی فراکدگذاری سه مرحله دارد:

۱. تحلیل برنامه‌ی اصلی جهت تشخیص ساختار کد.
۲. حاصل تحلیل فوق برای فراکدگذاری و یافتن محل‌هایی که باید آن‌ها را درج نمود، مورد استفاده قرار می‌گیرد.
۳. کد مورد نظر با فراکدگذاری تغییر داده می‌شود به‌گونه‌ای که رفتار برنامه حفظ شود.

پس از فراکدگذاری، آزمایه را می‌توان روی آن اجرا کرد و نشانه‌های ثبت‌شده از رویدادها را برای استخراج پوشش هر آزمایه به‌کار برد.

در حوزه‌ی آزمون رگرسیون، اطلاعات پوشش عموماً از نسخه‌ی پیشین نرم‌افزار مورد استفاده قرار می‌گیرد. به‌منظور اندازه‌گیری پوشش هر آزمایه، لازم است تا آنرا فراکدگذاری کرده و سپس اجرا نماییم. اما این‌کار هدف بهینه‌سازی آزمون رگرسیون را که جلوگیری از اجرای مجدد آزمایه‌ها غیر ضروری است، به مخاطره می‌اندازد. برای حل این مشکل، محققان بررسی و ارزیابی کرده‌اند که داده‌های پوشش حاصل از نسخه‌ی قبلی، اگر در دسترس باشند، تخمین منطقی از داده‌های پوشش کنونی هستند. بنابراین فنون آزمون رگرسیون مبتنی بر پوشش، فرض می‌کنند که در هنگام اجرای آزمایه‌ها روی نسخه‌ی قبلی، از پیش کد فراکدگذاری شده و داده‌های پوشش جمع‌آوری شده‌اند. داده‌های پوشش موجود را می‌توان برای هدایتِ بهینه‌سازی نسخه‌ی کنونی بکار گرفت.

۱-۵- آزمون مجدد کامل

این متدولوژی، به بیان ساده، به معنی استفاده‌ی مجدد *تمام* آزمایه‌های تولید شده در گذشته است؛ یعنی تمام آزمایه‌های مورد نیاز برای آزمودن نسخه‌ی اولیه، به‌همراه تمام آزمایه‌هایی که در هر مرتبه تغییر یافتن نسخه‌ی

---

[1] Basic block
[2] Trace
[3] Program profiling
[4] Linker

نرم‌افزار تولید شده‌اند، در هر تکرار آزمون رگرسیون، مجدداً اجرا می‌شوند. روشن است که در جریان تغییرات نسخ، بسیاری از آزمایه‌ها منسوخ 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 توجیه 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 آزمایه‌های 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. در عین حال فنون انتخاب در آزمون رگرسیون، مجموعه آزمون کل را کاهش دائمی نمی‌دهند و آزمایه‌ها را حذف نمی‌کنند. مسأله‌ی اجرای مجدد انتخابی آزمایه‌ها را می‌توان به‌صورت زیر بیان کرد:

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

۱. زیرمجموعه‌ی $T' \subset T$ را مجموعه‌ای از آزمایه‌ها 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 انتخابیِ آزمایه‌ها، پاره‌ای مسایل را مد نظر قرار می‌دهد: در گام (۱) که شامل *مسأله‌ی انتخاب آزمایه‌های رگرسیونی* (آزمایه‌ها برای بخش‌های دگرگونی یافته) است؛ که به‌صورت 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، رشته آزمونی با پوشش کامل کد باشد، این انتخاب را انتخاب امن در آزمون رگرسیون[1] گویند. مسأله‌ی بسیار مهم در مورد این

[1] Safe Regression Test Selection

فنون، موازنه‌ی بین ایمنی و کارایی[1] است. فنون RTS ایمن، تحت شرایط به‌خصوص، تضمین می‌کند که آزمایه-های انتخاب نشده، در اجرا بر روی نسخه‌ی جدید نمی‌توانستند خطایی آشکار کنند. استراتژی اکثر 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، آزمایه‌هایی انتخاب می‌شوند که بخش‌های اصلاح شده در نسخه‌ی جدید را می‌آزمایند. مطالعات نشان داده‌اند که میزان تغییرات بین دو نسخه، به‌شدت کارایی فنون آزمون رگرسیون انتخابی را تحت تأثیر قرار می‌دهد.

معمولاً فنون انتخاب آزمایه‌ها براساس شناسایی تغییرات و میزان تأثیر آن‌ها می‌باشد که با استفاده از فنونی همچون تفاوت معنایی، تفاوت متنی، عملیات برش‌بندی، فوق داده‌های مؤلفه‌ای و گراف‌های رابطه‌ای بین کلاسی صورت می‌پذیرد. از آن‌جا که انتخاب دقیق آزمایه‌های 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.

۱-۷- اولویت‌دهی آزمایه‌ها

ترتیب‌دهی آزمایه‌ها 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 آزمایه‌ها نامیده می‌شود. به بیان ساده، فن اولویت‌دهی آزمایه‌ها در حالت کلی، کل آزمایه‌های موجود در مجموعه آزمون را نگه می‌دارد؛ در عین حال پیش از اجرای آزمایه‌ها، آن‌ها را مرتب می‌کند تا به این صورت در دستیابی سریع‌تر به اهداف آزمون، به آزمونگران کمک کند.

فضای جستجو در مسأله‌ی اولویت‌دهی آزمایه‌ها، در حقیقت جایگشتی از کلیه‌ی آزمایه‌های موجود در رشته آزمون است؛ به‌گونه‌ای که این ترتیب اجرا، آشکارکننده‌ی حداکثر خطاهای ممکن باشد. بنابراین مسأله‌ی اولویت‌دهی آزمایه‌ها، 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 آزمایه‌ها مکاشفه‌ای بوده و با استفاده از فنون وزن‌دهی و معیارهای پوشش[2] گوناگون، برای بهبود سرعت کشف خطا و کارایی بیشتر آزمون با هم رقابت می‌کنند و هیچ‌یک در حالت کلی بر دیگری برتری ندارند.

۱-۸- کاهش مجموعه آزمون

روشن است که هر چه تعداد آزمایه‌ها در فرایند آزمون بیشتر باشد، احتمال تأمین نیازمندی‌های آزمون بیشتر می‌گردد. اما مشکل این‌جا است که رشد بی‌رویه‌ی مجموعه آزمون می‌تواند مشکل‌ساز شود. به‌عبارت دیگر، آزمایه-های جدید ممکن است تولید و در مجموعه درج شود تا نیازمندی‌های بیشتری را تأمین نماید. در نتیجه آزمایه-های موجود در مجموعه آزمون، ممکن است بیشتر از حد نیاز برای تأمین نیازمندی‌های آزمون باشد. مسأله اینجاست که اگر تعدادی از آزمایه‌ها از مجموعه آزمون حذف شود، این مجموعه کماکان همه‌ی نیازمندی‌های

---

[1] Efficiency
[2] Coverage Criteria

آزمون را که مجموعه‌ی اولیه تأمین می‌کرد، برآورده می‌نماید. این مسأله به "کاهش مجموعه آزمون" (کمینه‌سازی مجموعه آزمون یا پالایش آزمایه‌ها) مشهور بوده و زیرمجموعه‌ی حاصل، مجموعه‌ی نماینده[1] یا شاخص نامیده می‌شود. اگر هیچ زیرمجموعه‌ای از مجموعه‌ی نماینده، تمام نیازمندی‌ها را تأمین نکند، آنرا مجموعه‌ی بهینه یا مجموعه‌ی کمینه می‌نامیم.

فنون کاهش مجموعه آزمون، با تشخیص و حذف آزمایه‌های افزونه، دنباله آزمون را به‌طور 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 آزمایه‌ها به‌گونه‌ای که همه‌ی نیازمندی‌های آزمون را برآورده سازد، سبب افزایش کارایی فرایند آزمون می‌شود. علت این مسأله این است که زمان اجرای آزمایه‌ها 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 به‌گونه‌ای انجام شود که قابلیت آشکارکنندگی خطای مجموعه‌ی حاصل نسبت به مجموعه آزمون اولیه، بدون تغییر یا با کاهشی قابل اغماض (نسبت به میزان کاهش به‌دست 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 به‌دست 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:

۱. **هزینه‌ي اجراي فرایند (ابزار) کاهش مجموعه آزمون براي تولید مجموعه‌ي کاهش‌یافته**: یک ابزار کاهش مجموعه آزمون را می‌توان درپی ارائه نسخه‌های نرم‌افزار اجرا کرد و اجرای آن می‌تواند به‌صورت خودکار و در ساعات خلوتی[2] انجام شود و در این صورت هزینه‌ی اجرای ابزار خیلی مشکل‌زا نخواهد بود. علاوه بر این هنگامی‌که با استفاده از یک ابزار، مجموعه آزمون کاهش داده شد، هزینه‌ی این کاهش به‌تدریج در اجراهای مکرر آزمون و استفاده از مجموعه‌ی کاهش‌یافته در نگارش‌های بعدی توزیع شده و در مقایسه با سایر هزینه‌ها کم اهمیت می‌گردد.

۲. **حذف برخي از آزمایه‌ها که در صورت اجرا منجر به آشکارسازي خطا مي‌شوند**: این هزینه از مورد قبلی مهم‌تر است. حذف چنین آزمایه‌هایی موجب کاهش درصد کشف خطای مجموعه آزمون شده و از کارایی آن می‌کاهد. اثر این کاهش به‌تدریج با بکارگیری مجدد آن در نسخه‌های بعدی نرم‌افزار افزایش یافته تا جایی‌که حجم خطاهای از دست‌رفته زیاد شده و فرایند آزمون دچار اختلال و عدم کارایی می‌گردد. دو روش برای بررسی هزینه‌های مربوط به حذف آزمایه‌های آشکار کننده خطا وجود دارد:

أ. **بر مبناي هر آزمایه**[3]: با داشتن برنامه خطادار P و مجموعه آزمون T یکی از راه‌های اندازه‌گیری هزینه‌ی مجموعه آزمون کاهش‌یافته بر کشف خطا، شناسایی آزمایه‌هایی است که در T وجود دارند و

[1] Representative Set
[2] Off-peak
[3] On a per-test-case basis

خطاهایی را از P آشکار می‌کنند اما در $T_{\min}$ نیستند. این کمیت را می‌توان با تعداد آزمایه‌های آشکار کننده خطا در T نرمال کرد. نکته مهم اینجاست که ممکن است چندین آزمایه یک خطای واحد را آشکار نمایند. در این‌حالت می‌توان بعضی از آن‌ها را حذف نمود بدون این‌که تأثیری در کشف خطا داشته باشند.

ب. **بر مبناي هر مجموعه آزمون**[1]: روش دیگر دسته‌بندی نتایج کاهش مجموعه آزمون بر حسب یک خطا در P به یکی از سه روش زیر است: ۱) هیچ آزمایه‌ای در T خطا آشکار نمی‌کند. ۲) بعضی آزمایه‌ها هم در T و هم در $T_{\min}$ هستند که خطاهایی را آشکار می‌نمایند. ۳) بعضی از آزمایه‌ها در T آشکارکننده‌ی خطا هستند اما در $T_{\min}$ هیچ آزمایه 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 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 آزمایه‌ها قابل کشف باشد، پس از کمینه‌سازی نیز می‌توان کشف نمود در حالی‌که خطاهایی که توسط تعداد معدودی از آزمایه‌ها قابل کشف باشد، پس از کمینه‌سازی در معرض عدم کشف قرار می‌گیرند.

عموماً کاهش مجموعه آزمون یک مسأله‌ی بهینه‌سازی تک‌هدفه[2] است. اما در برخی بررسی‌های اخیر کاهش مجموعه آزمون چندهدفه[3] نیز مورد توجه قرار گرفته است که برخی از این اهداف پوشش کد، تاریخچه‌ی کشف خطا در گذشته و هزینه‌ی اجرایی هستند.

در کل مسأله کاهش مجموعه آزمون‌ها و انتخاب آزمایه‌ها از این جهت که زیر مجموعه‌ای از مجموعه اصلی را تولید می‌کنند شبیه یکدیگر می‌باشند. معیار کاهش مجموعه آزمون این‌است که آیا زیر مجموعه‌ی منتخب، همه نیازمندی‌های آزمون را تأمین می‌کند در حالی‌که تمرکز انتخاب آزمایه‌ها بر بخش‌های تغییریافته سیستم نرم‌افزاری است. در بعضی از تقسیم‌بندی‌های اولیه، کاهش مجموعه آزمون در حوزه‌ی انتخاب آزمایه‌ها دسته‌بندی می‌شود. بعضی از روش‌هایی که ارائه شده سعی می‌نمایند تا مسایل را به‌صورت ترکیبی حل نمایند. مثلاً روش ارائه شده توسط والکات[4] و همکاران از الگوریتم‌های ژنتیک استفاده می‌کند تا مسأله اولویت‌دهی و انتخاب آزمایه‌ها را همزمان حل نماید. یا تحقیقی که در انجام شده مجموعه‌ای روش‌های ابتکاری و الگوریتم‌های ژنتیک را برای حل مسأله‌ی کاهش مجموعه آزمون و اولویت‌دهی آزمایه‌ها ارائه نموده است.

از آن‌جایی‌که هر آزمایه می‌تواند یک یا چند نیازمندی آزمون را برآورده نماید، ارتباط میان آن‌ها را می‌توان با یک ماتریس بنام ماتریس *آزمایه-نیازمندی* نمایش داد. مثالی از آنرا در جدول ۱ مشاهده می‌نمایید.

جدول ۱: ماتریس آزمایه-نیازمندی

نیازمندی/آزمایه	نیازمندی ۱	نیازمندی ۲	نیازمندی ۳	نیازمندی ۴	نیازمندی ۵

[1] On a per-test-suite basis
[2] Single Objective
[3] Multi Objective
[4] Walcott

آزمایه ۱	۰	۱	۱	۱	۱
آزمایه ۲	۱	۰	۰	۰	۰
آزمایه ۳	۰	۱	۰	۰	۱
آزمایه ۴	۱	۰	۰	۰	۰

مجموعه‌ی {آزمایه ۱، آزمایه ۲، آزمایه ۳، آزمایه ۴} معرف آزمایه‌ها 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 آزمایه iام می‌تواند نیازمندی jام 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 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 آزمایه های جدید هستند، شناسایی شود. لذا برای افزایش **سودمندی مجموعه آزمون در پاسخ به تغییرات**، دو کار باید انجام شود:

- وارسی شود آیا مجموعه آزمون‌های موجود برای تغییرات ایجاد شده در برنامه کافی هستند.
- اگر کافی نیستند، باید آزمایه‌های 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.

۱-۱۰- طراحی الگوریتم‌های کاهش مجموعه آزمون و اولویت‌دهی آزمایه‌ها

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

۱. همه الگوریتم‌ها در ساخت T' میزان سهم و مفید بودن[1] هر یک از آزمایه‌ها را بر اساس خصوصیات برنامه تحت آزمون P، خصوصیات آزمایه‌ها در T و هدف T' مورد ارزیابی قرار می‌دهند. با وجودی‌که خصوصیات متنوعی مطرح شده‌اند اما همگی را می‌توان به دو نوع **پوشش** و **هزینه** تقسیم‌بندی نمود. پوشش،

[1] Contribution and goodness

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 آزمایه، زمان بر پایی و آماده‌سازی برای اجرای یک آزمایه و هر هزینه دیگری در ارتباط با اجرای یک آزمایه (به‌خصوص در حالتی که از شبیه‌سازی استفاده شده باشد) می‌باشد. الگوریتم‌ها آزمایه‌های T را بر اساس ترکیبی از این خصوصیات ارزیابی می‌نمایند. مثلاً مراجعی انواع مختلفی از پوشش را جهت کاهش مجموعه آزمون به‌کار گرفته‌اند. فن آن‌ها هنگام تصمیم‌گیری درباره‌ی آزمایه‌هایی که باید در T' قرار گیرند، به ازاء هر یک از موجودیت‌های برنامه،E، آزمایه‌هایی را انتخاب می‌کند که E را بپوشاند. به‌عنوان مثالی دیگر، مرجع دیگری برای ایجاد ترتیبی از آزمایه‌ها در T'، انواع مختلفی از پوشش را همراه با میزان خطاخیز بودن پیمانه‌ها مورد استفاده قرار داده‌اند.

۲. الگوریتم‌های کاهش مجموعه آزمون و اولویت‌دهی آزمایه‌ها، در دفعاتی (فرکانسی) که سهم یک آزمایه 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 آزمایه‌ها در T' استفاده کند. این فرکانس را جمع کل[1] می‌نامند و از آن برای محاسبه‌ی اولویت‌دهی مبتنی بر جمع[2] استفاده می‌شود. روش دیگر، سهم هر آزمایه 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 و از آن برای محاسبه اولویت‌دهی مبتنی بر افزایش[3] استفاده می‌شود.

۳. یک الگوریتم می‌تواند از سهم آزمایه برای تعیین آزمایه‌ای که باید به T' اضافه شود استفاده کند (مورد استفاده‌ی کاهش مجموعه آزمون و اولویت دهی آزمایه‌ها) و یا از آن در جهت تعیین آزمایه‌ای که باید از T حذف شود، بهره ببرد (مورد استفاده‌ی کاهش مجموعه آزمون). اولی را فن **افزایشی**[4] و دومی را فن **تجزیه**[5] می‌نامند. فن افزایشی کار خود را با مجموعه‌ی خالی T' شروع می‌کند و آزمایه‌ها را یکی پس از دیگری به آن اضافه می‌کند. اما فن تجزیه با مجموعه T شروع کرده و به‌تدریج آزمایه‌ها را از آن حذف می‌کند تا T' به‌دست 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 آزمایه‌ها را می‌پوشاند. در مقابل الگوریتم افزایشی تنها زمانی که خاتمه یابد می‌توان تضمین کرد که پوشش کامل حاصل شده است. ترکیب این صفات یا به‌عبارتی محاسبه‌ی سهم هر آزمایه، فرکانس ارزیابی (مجدد) سهم هر آزمایه 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- نتیجه‌گیری

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. برای حل مشکل هزینه‌ی زیاد آزمون رگرسیون، متدولوژی‌های مختلفی ارائه شده‌اند که مهم‌ترین آن‌ها عبارتند از: آزمون رگرسیون انتخابی، کاهش مجموعه آزمون و اولویت‌دهی آزمایه‌ها. در مورد کاهش مجموعه آزمون، دو هزینه اهمیت دارد: ۱) هزینه اجرای فرایند (ابزار) کاهش مجموعه آزمون برای تولید مجموعه‌ی کاهش‌یافته و ۲) حذف برخی از آزمایه‌ها که در صورت اجرا منجر به آشکارسازی خطا می‌شوند. هزینه‌ی دوم اهمیت بیشتری دارد. پوشش و هزینه، دو خصوصیت مهمِ آزمایه‌ها است که الگوریتم‌های کاهش در ساخت مجموعه‌ی کاهش‌یافته میزان سهم و مفید بودن هر یک از آزمایه‌ها را بر اساس آن می‌سنجند.