تاریخ دریافت مقاله: ۹۶/۰۱/۰۱
تاریخ پذیرش مقاله: ۹۶/۰۲/۱۵

ارزیابی و مقایسه تازه‌ترین فنون آزمون پس‌نمایی نرم‌افزار

محمّدرضا دهقانی تفتی
دانشجوی کارشناسی ارشد نرم‌افزار، دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی
پست الکترونیکی: mo.dehghani@mail.sbu.ac.ir

علیرضا خلیلیان
دانشجوی دکتری نرم‌افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان
پست الکترونیکی: khalilian@eng.ui.ac.ir

مجتبی وحیدی اصل*
استادیار مهندسی نرم‌افزار، دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی
پست الکترونیکی: mo_vahidi@sbu.ac.ir

چکیده

رواج و ضرورت استفاده از آزمون پس‌نمایی[1] در مرحله نگهداری نرم‌افزار، منجر به تحقیقات گسترده و توسعه فنون متعددی شده است. اکنون به پاسخ دو سؤال نیاز داریم: روند تحقیقاتی چگونه بوده و چطور باید ادامه یابد؟ فنون را چگونه با هم مقایسه کنیم و فن بهتر کدام است؟ این مقاله مهم‌ترین دستاوردهای آزمون پس‌نمایی نرم‌افزار را در خلال سال‌های ۲۰۱۰ تا ۲۰۱۶ گزارش می‌نماید. هدف این است که ضمن روشن کردن مفاهیم، معیارهایی نیز معرفی گردد که زمینه ارزیابی و مقایسه فنون آزمون پس‌نمایی را به‌طور نظام‌یافته فراهم سازد. با تکیه بر نتایج بررسی‌ها، مهم‌ترین دستاوردهای نظری و تجربی تحقیقات اخیر گزارش شده است.

واژه‌های کلیدی: آزمون نرم‌افزار، آزمون پس‌نمایی، مجموعه آزمون، آزمایه.

مقدمه

آزمون پس‌نمایی یکی از رایج‌ترین روش‌های مورد استفاده در صنعت و دانشگاه برای بهینه‌سازی فعالیت‌های تضمین کیفیت نرم‌افزار حین مرحله نگهداری است؛ اختصاصاً کاهش خطر تغییرات و هزینه‌های محاسباتی [۱]. در مرحله نگهداری، نرم‌افزار دائماً به سه دلیل در حال تغییر است: الف) اشکال‌زدایی و رفع خطاها؛ ب) اضافه

* نویسندهٔ مسئول

1- Regression testing

```
1:  read (x, y);
2:  if ( y < 0) then
3:      power = -y;
4:  else
5:      power = y;
6:  endif
7:  z = 1;
8:  while (power != 0)  do
9:      z = z - y;    ***
10:     power = power – 1;
11: endwhile
12: if (y < 0) then
13:     z = 1 / z;
14: endif
15: result = z;
16: write (result);
```

نسخه اولیه برنامه

```
1:  read (x, y);
2:  if ( y < 0) then
3:      power = -y;
4:  else
5:      power = y;
6:  endif
7:  z = 1;
8:  while (power != 0)  do
9:      z = z * y;    ***
10:     power = power – 1;
11: endwhile
12: if (y < 0) then
13:     z = 1 / z;
14: endif
15: result = z;
16: write (result);
```

نسخه دوم برنامه

مجموعه آزمایه‌های پس‌نمایی

آزمایه	خروجی منتظره	خروجی واقعی
t_1	½	½
t_2	1	1
t_3	27	-8

روی نسخه اولیه (بالا)

روی نسخه دوم (پایین)

آزمایه	خروجی منتظره	خروجی واقعی
t_1	½	-1
t_2	1	1
t_3	27	27

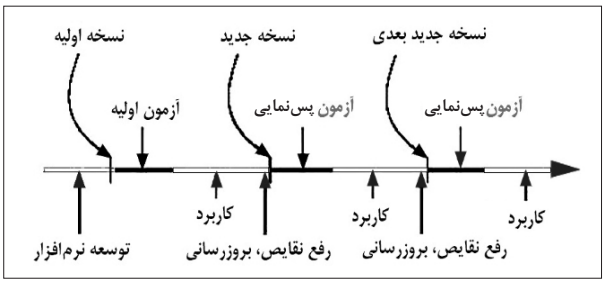
شکل ۱: مثال از عمکرد آزمون پس‌نمایی

شـــدن قابلیت‌های جدیـــد؛ پ) حذف یا تغییـــر ویژگی‌های موجـــود. مأموریت اصلی آزمون پس‌نمایی این اســـت که بررســـی کند نرم‌افزار تغییریافته هنوز درست کار می‌کند [۲]. به شکل ۱ توجه کنید.

برنامـــه دو ورودی x و y دریافت می‌کند و مقدار x_y را محاســـبه می‌نماید. برای آزمایش درستی آن از دو آزمایه t_1 و t_2 استفاده می‌کنیم که هر دو دستور شرطی (انشعاب، انتخـــاب، تصمیم‌گیری) را اجرا کند (پوشـــش دهد). جدول سمت راست در بالا خروجی‌هایی که از اجرای این دو آزمایه روی نســـخه اولیه دریافت می‌شود (واقعی) و خروجی که واقعاً باید دریافت شـــود (منتظره) را نشان می‌دهد. این دو آزمایه اجرای موفق[2] دارند چون خروجی واقعی و منتظره برابرند. اکنون سومین آزمایه t_3 روی برنامه اجرا می‌شود. متأســـفانه این آزمایه اجرای ناموفـــق[3] دارد چون به‌جای خروجی درســـت ۲۷ مقدار نادرســـت ۸- داده می‌شـــود. برای تصحیح خطا، خط شـــماره ۹ تغییر داده می‌شـــود و ســـه آزمایه دوباره روی نسخه دوم اجرا می‌شوند (سمت راســـت پایین). این‌بار آزمایه سوم اجرای موفق دارد ولی آزمایه دوم ناموفق. این معنای آزمون پس‌نمایی اســـت که می‌خواهد مطمئن شود تغییرات، بخش‌های درست را خراب نکرده باشند.

مسئله اینجاست که آزمون پس‌نمایی مهم اما پرهزینه

2- Pass
3- Fail



شکل ۲: جایگاه آزمون پس‌نمایی

اســـت، به‌طوری‌که بخـــش عظیمی از هزینه‌های توســـعه نرم‌افـــزار را به خود اختصاص می‌دهـــد. برای مثال طبق گزارش‌هـــای رســـمی، یکـــی از شـــرکت‌های نرم‌افزاری، محصولـــی با مجموعـــه آزمونی بالغ بـــر ۳۰۰۰۰ آزمایه[4] داشته اســـت که به بیش از ۱۰۰۰ ماشین‌ساعت برای اجرا شـــدن نیاز دارند [۶۲]. همچنین به صدها ساعت زمان نیاز است تا مهندســـان این فرآیند آزمون پس‌نمایی را نظارت کننـــد. پس هزینه آزمون پس‌نمایی شـــامل نیـــاز به زمان بسیار و منابع ســـخت‌افزاری، نرم‌افزاری و انسانی است. دو دلیل عمده ریشـــه هزینه بالای آزمون پس‌نمایی است که به ماهیت آزمون برمی‌گردد: یکی دفعات بســـیار انجام آزمون و دیگری حجـــم رو به افزایش آزمایه‌ها [۱, ۶, ۵۰, ۵۱, ۵۵, ۶۰]. شکل ۲ عملکرد آزمون پس‌نمایی (رگرسیون) را در چرخـــه حیات نرم‌افزار به تصویر می‌کشـــد. هر چه چرخه حیات توســـعه نرم‌افزار کوتاهتر باشد (مثل توسعه نرم‌افزار با روشگان چابک)، محدودیت‌ها و حساسیت‌های

4- Test case

بیشـــتری بر چگونگی انجام آزمـــون پس‌نمایی با توجه به منابع محدود ایجاد می‌شود [۶۱].

برای پشـــتیبانی و کاهش هزینه‌های آزمون پس‌نمایی تاکنون چهار دســـته از فنون در نظرگرفته شده‌اند [۱, ۲]: الف) اولویت‌دهی[5]، ب) انتخاب[6]، پ) کاهش/کمینه‌ســـازی[7]؛ ت) افزایش[8]. در ســـال‌های اولیه پیدایـــش فنون پس‌نمایی که از حدود ۱۹۹۵ شـــروع شـــده است، محققان بر طراحی فنون مختلف در هر رویکرد تمرکز داشـــته‌اند و این رویه تقریباً تا سال ۲۰۱۰ ادامه داشـــته است. پس از این دوره، فنون آزمون پس‌نمایی آن‌قدر بلوغ یافتند که به‌تدریج وارد صنعت شـــدند. در دوره پس از بلوغ، تمرکز تحقیقاتی دو شـــاخه پیدا کرد: الف) مطالعه‌های تجربی گسترده با فنون موجـــود روی برنامه‌هایی با مقیاس بـــزرگ و واقعی؛ ب) طراحی فنون پیچیده.

در مطالعه‌های تجربی گسترده، محققان به‌دنبال شناخت بیشتر عملکرد فنون موجود در موقعیت‌های عملی هستند و سعی می‌کنند از جنبه‌های مختلف آن‌ها را مورد سنجش قرار دهند. نشـــان داده شـــده که بـــرای حداکثر بهره‌وری از هـــر فن باید عوامـــل زمینه‌ای مؤثـــر در موفقیت آن را بشناسیم [۵۳]. گاهی انتخاب نادرست یک فن خوب در یک موقعیت عملی بخصوص، باعث تنزل شدید سودمندی آن فن می‌گردد. فنون پیچیـــده با تکیه بر تجربه طراحی فنون گذشـــته و نتایج حاصل از ارزیابی آن‌ها و نیز بهره‌گیری از دانش نظری قوی‌تر طراحی می‌شوند که آگاهانه‌تر عمل کنند و سودمندی بیشتری در عمل داشته باشند.

برای بهره‌مندی از یافته‌های دو شاخه از تحقیقات ذکر شـــده، لازم اســـت فنون در یک مطالعه نظام‌یافته بررسی شوند. یو و هارمان [۲] مروری بر فنون آزمون پس‌نمایی کرده‌انـــد کـــه در آن مهم‌ترین مقاله‌هـــا از ۱۹۷۷ تا ۲۰۱۰ مورد بررسی قرار گرفته‌اند؛ بررسی فنون به‌تنهایی، بدون هیچ معیار و بدون مقایســـه‌ای صورت گرفته است. کی و همـــکاران او [۴۹] با تمرکز بر فنون آزمون پس‌نمایی وب، مطالعه نقشه‌برداری نظام‌یافته‌ای روی مهم‌ترین مقاله‌های ســـال ۲۰۰۰ تا ۲۰۱۳ ارائه کرده‌اند. اِنگستروم و همکاران [۵۱] ۲۷ مقاله در روش‌های انتخاب آزمون پس‌نمایی ارائه شـــده بین ۱۹۹۷ تا ۲۰۰۶ را مورد بررسی قرار داده‌اند که آن‌ها را چند معیار ســـاده ارزیابی کرده‌اند. کاتال و میشرا در مطالعـــه نقشـــه‌برداری نظام‌یافتـــه [۵۰] ۱۲۰ مقاله بین ۲۰۰۱ تا ۲۰۱۰ در اولویت‌دهی آزمایه‌ها را با در نظر گرفتن چند معیار ســـاده مورد بررســـی قرار داده‌اند. هیونسوک دو [۱] مقاله‌هـــای اخیر در آزمـــون پس‌نمایی بین ۲۰۱۰ تا ۲۰۱۵ را بدون معیار خاصی مورد بررسی قرار داده است. مطالعه‌های مذکور یک یا چند مورد از مشـــکل‌های زیر را دارند: الف) عدم پوشش مقالات تا ۲۰۱۶؛ ب) عدم بررسی فنون هـــر چهار رویکرد به‌طور یکجـــا بخصوص افزایش آزمایه‌ها؛ پ) عدم معرفی معیارهایی برای مقایســـه فنون؛ ت) عـــدم تمرکز بـــر مطالعه‌های تجربی گســـترده و فنون اختصاصی به‌طور همزمان.

برای پـــر کردن خلأ تحقیقاتی یاد شـــده، مقاله حاضر دانش و تجربه‌های حاصل از مطالعه‌های اخیر را از دو جنبه مطرح شده مرور می‌کند و فنون را با هم مقایسه می‌نماید. از آنجائی‌که مقایســـه باید ضابطه‌مند باشـــد، معیارهایی معرفی می‌شود که بر اساس آن‌ها امکان ارزیابی و مقایسه میسر گردد. مقاله حاضر از آن جهت جدید است که فنون و مطالعه‌های جدید آزمون پس‌نمایی را به‌طور خلاصه مرور می‌کند و نکات مهم آن‌ها را برجســـته می‌ســـازد. همچنین چارچوبی برای ارزیابی و مقایســـه فنون آزمون پسنمایی نرم‌افزار ارائه می‌کند. حاصل مقاله حاضر دســـتکم به دو شکل می‌تواند مورد استفاده قرار گیرد:

- برای پژوهشگرانی که قصد تحقیق در این زمینه دارند، مروری از تازه‌تریـــن یافته‌ها عرضه می‌کند و با جمع‌بندی نتایج، آن‌ها را به سمت خلق فنون سودمندتر هدایت می‌نماید.
- برای کارورزان صنعت که به‌دنبال پیاده‌ســـازی فنی مؤثر و مناســـب با پروژه‌های عملی خود هســـتند، امکان

5- Prioritization
6- Selection
7- Reduction/Minimization
8- Augmentation

شناخت و ارزیابی فنون موجود را میسر می‌نماید.

به‌طور خلاصه، نوآوری‌های این مقاله عبارتند از:

- معرفی فنون اخیر آزمون پس‌نمایی نرم‌افزار و ارائه خلاصه‌ای از عملکرد هر یک
- ارائه مجموعه‌ای از معیارها جهت ارزیابی و مقایسه فنون آزمون پس‌نمایی نرم‌افزار
- بحث در یافته‌ها و نتایج حاصل از ارزیابی و مقایسه آن‌ها و ارائه رهنمودهایی برای کاربردهای عملی و پژوهش‌های آتی

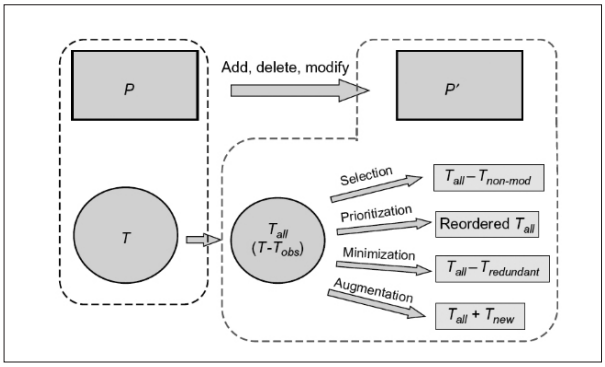
ساختار ادامه مقاله به این شرح است: بخش دوم ادبیات موضوع را با ارائه تعریف‌های لازم مرور می‌کند. روشگان جمع‌آوری مقالات و معیارهای ارزیابی در بخش سوم بیان می‌شوند. در بخش چهارم خلاصه هر روش بیان می‌شود و با معیارها ارزیابی می‌گردد. بخش پنجم هم مقاله را جمع‌بندی می‌نماید.

۲- مروری بر ادبیات

معمولاً آزمون پس‌نمایی با استفادهٔ مجدد از آزمایه‌های قبلی و با ایجاد آزمایه‌های جدید که برای آزمودن خصوصیات جدید استفاده می‌شود، انجام می‌گیرد. آزمون پس‌نمایی به‌طور غیررسمی این‌چنین تعریف می‌شود:

تعریف ۱: فرض کنید P برنامه‌ای باشد که برای ایجاد نسخه‌ای جدید از آن، اصلاح شده و P' به وجود آمده و T مجموعه آزمونی باشد که برای P ساخته شده است. در گذار از P به P'، برنامه ممکن است پس‌رفت نماید؛ رفتاری که قبلاً در P به‌درستی عمل می‌کرده ممکن است در P' دچار مشکل شود. آزمون پس‌نمایی برای تشخیص این‌که آیا P' پس‌رفت داشته است یا خیر صورت می‌گیرد [۱].

شکل ۳ خلاصه فنون آزمون پس‌نمایی را نشان می‌دهد. در سمت راست شکل ۱، آزمایه‌هایی که حاصل از اعمال فنون هستند، نشان داده شده‌اند. آزمون پس‌نمایی را می‌توان به دو دسته کلی اصلاحی[9] و پیشرو[10] تقسیم



شکل ۳: نمای کلی فنون آزمون پس‌نمایی نرم‌افزار [۱]

کرد: اولی 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. اما دومی 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 آزمایه جدید نیاز دارند. سه فن انتخاب، اولویت‌دهی و کاهش/کمینه‌سازی بر کاهش هزینه‌های آزمون پس‌نمایی از طریق استفادهٔ مجدد از آزمایه‌های موجود تمرکز دارند. اما همیشه استفادهٔ مجدد از آزمایه‌های موجود، کافی نیست؛ برای آزمودن قابلیت‌های جدید، آزمایه‌های جدید نیز ممکن است مورد نیاز باشند. فن افزایش، آن نواحی از کد که جدیداً اضافه شده‌اند را تشخیص می‌دهد و آزمایه‌های جدیدی برای آن‌ها می‌سازد [۱]. برای کامل بودن رویکردها گاهی به آزمون مجدد کامل[11] هم به‌عنوان روشی اشاره می‌شود که در آن بی قید و شرط همه آزمایه‌هایی که داریم را مورد استفاده قرار می‌دهیم. آزمایه‌ها را می‌توان به پنج دسته تقسیم کرد [۱] که دسته‌های اول تا سوم حاوی آزمایه‌های داخل مجموعه آزمون فعلی هستند. دسته‌های چهارم و پنجم آزمایه‌هایی هستند که برای آزمون برنامه تغییر یافته باید تولید شوند.

تعریف ۲: آزمایه‌های قابل استفاده مجدد[12] آزمایه‌هایی هستند که فقط بخش‌هایی از برنامه را که بین دو نسخه

9- Corrective
10- Progressive

11- Retest all
12- Reusable

تغییـــر نکرده‌انـــد، می‌آزمایند. اجرای ایـــن آزمایه‌ها برای آزمون نسخه تغییریافته برنامه ضروری نیست ولی آن‌ها را در مجموعـــه آزمـــون نگه می‌دارند زیـــرا برای آزمون پس‌نمایی نسخه‌های بعدی برنامه لازم می‌شوند.

تعریف ۳: آزمایه‌های قابل آزمون مجدد[13] آزمایه‌هایی هســـتند که بخش‌های تغییر کرده از برنامه فعلی در نسخه جدید آن‌را می‌آزمایند. بنابراین برای آزمون نســـخه تغییر کرده، این آزمایه‌ها حتماً باید اجرا شوند.

تعریف ۴: آزمایه‌های منسوخ[14] آزمایه‌هایی هستند که: الف) ارتباط بین ورودی/خروجی‌های آن‌ها به‌خاطر تغییر مشخصات دیگر معتبر نیست؛ ب) به‌خاطر تغییرات برنامه، بخش‌های که بـــرای آزمون آن‌ها طراحی شـــده بودند را دیگر نمی‌آزمایند؛ ج) آزمایه‌هایی ساختاری هستند که دیگر سهمی در پوشش ساختاری برنامه ندارند.

تعریف ۵: آزمایه‌های ســـاختاری جدید[15] ساختارهای برنامه تغییریافته را می‌آزمایند و پوشـــش ســـاختاری در بخش‌های تغییر یافته نسخه جدید برنامه ایجاد می‌کنند.

تعریف ۶: آزمایه‌های مشـــخصات جدید[16] آزمایه‌های جدیـــدی هســـتند کـــه مشـــخصات برنامـــه تغییریافته را می‌آزماینـــد؛ یعنـــی کدهای جدیـــد تولید شـــده حاصل از بخش‌هاتی تغییریافته مشخصات نسخه جدید.

فنون انتخاب به دنبال کاهش انـــدازهٔ مجموعه آزمون هســـتند، اما این فنون، آگاه از تغییر هســـتند؛ یعنی این‌که انتخـــاب، موقتـــی اســـت و تمرکزش بر روی تشـــخیص قسمت‌های تغییریافتهٔ برنامه است. دلیل انتخاب آزمایه‌ها این اســـت که آن‌ها با قسمت‌های تغییریافتهٔ نرم‌افزار تحت آزمون مرتبط هستند [۲].

تعریـــف ۷: رودرمـــل و هارولد[17] مســـئله انتخاب را به صورت صوری به شکل زیر تعریف کرده‌اند [۵]:

ورودی: برنامه P، نسخهٔ تغییریافتهٔ P که 'P است و یک مجموعه آزمون T.

مســـئله: زیرمجموعه‌ای از T به نـــام 'T بیابید که برای آزمودن 'P استفاده می‌شود.

ایمنی[18] جنبهٔ مهم فنون انتخاب است. فنون انتخابِ ایمن تضمین می‌کنند آزمایه‌هایی که انتخاب نشده‌اند، نمی‌توانند خطایی را در 'P آشـــکار کنند. دو جنبهٔ دیگر فنون انتخاب، دقت[19] و بهره‌وری[20] هستند. دقت اشاره دارد به این‌که این فنون تا چه اندازه به درســـتی آزمایه‌ها را انتخاب می‌کنند، به‌طوری که دیگر نیاز نباشـــد آزمایه‌هـــای اضافه‌تری را دوباره اجرا نمـــود. دغدغهٔ بهـــره‌وری، هزینهٔ جمع‌آوری داده‌های مورد نیاز برای اجرای فن انتخاب و هزینهٔ اجرای فـــن انتخاب می‌باشـــد. مصالحه بین دقـــت و بهره‌وری و مقرون‌به‌صرفه بودنشان با توجه به خصوصیات برنامه، تغییرات و مجموعه‌های آزمون، متفاوت است [۱]. مطالعه‌ها [۵۳] نشان داده‌اند که میزان تغییرات بین دو نسخه، به‌شدت کارایی فنون انتخاب را تحت تأثیر قرار می‌دهد.

دغدغهٔ اولویت‌دهی، مرتـــب کردن آزمایه‌ها به گونه‌ای اســـت که معیار مورد نظر، مثلاً نرخ تشخیص خطا، زودتر بیشینه گردد. اولویت‌دهی به دنبال یافتن جایگشت بهینه از دنبالهٔ آزمایه‌ها است و فرض می‌کند تمامی آزمایه‌ها ممکن است به ترتیبی که تولید کرده، اجرا گردند، اما عمل آزمون ممکن اســـت در نقطهٔ دلخواهـــی از فرآیند آزمون، متوقف گـــردد. با این وجـــود اولویت‌دهی می‌تواند شـــانس این‌که آزمایه‌های مهم اجرا شوند را بهبود دهد.

تعریـــف ۸: مســـئله اولویت‌دهی به شـــکل زیر تعریف می‌گردد [۲]:

ورودی: یـــک مجموعـــه آزمـــون T، مجموعـــه‌ای از جایگشـــت‌های PT، T، و تابعـــی از PT بـــه اعـــداد حقیقی، $f = PT \rightarrow \mathbb{R}$.

مسئله: یافتن $T' \in PT$ به‌طوری که

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

مســـئلهٔ اولویت‌دهی آزمایه‌ها، یک مسئله جستجو است که در برخی مراجع آن‌را معادل با مســـئله کوله‌پشتی صفر و

13- Retestable
14- Obsolete
15- New-structural
16- New-specification
17- Harrold
18- Safety
19- Precision
20- Efficiency

یک دانسته‌اند. پس از آنجائی‌که اولویت‌دهی، NP سخت است، ممکن است تضمین بهینگی خیلی هزینه‌بر باشد. بر حسب انواع مختلف اطلاعات در دسترس که مرتبط با آزمایه‌ها هستند، فنون مختلف اولویت‌دهی را می‌توان طراحی و بهره‌برداری نمود [۶].

فنون کاهش یا کمینه‌سازی به دنبال کاهش اندازهٔ یک مجموعه آزمون از طریق حذف دائمی آزمایه‌های اضافی از مجموعه آزمون هستند [۲]. در نتیجه هزینهٔ اجرا، تأیید و مدیریت این آزمایه‌ها کاهش می‌یابد و بهره‌وری آزمون افزایش می‌یابد [۳].

تعریف ۹: رودرمل کاهش مجموعه آزمون را این‌چنین تعریف می‌کند [۴]:

ورودی: یک مجموعه آزمون T، مجموعه‌ای از نیازمندی‌های {r1,...,rm} که باید برآورده گردند و زیرمجموعه‌های T، که $T_1,...,T_n$ هستند به‌طوری که هر کدام از آزمایه‌های t_j که متعلق به T_i باشد، می‌تواند نیازمندی r_i را تأمین نماید.

مسئله: مجموعه‌ای نماینده[۲۱] از آزمایه‌ها به نام 'T از T بیابید که تمامی r_iها را برآورده نماید.

معیار آزمون زمانی تأمین می‌شود که همه نیازمندی‌های آزمون تأمین گردند. مسئلهٔ کمینه‌سازی مجموعه آزمون NP کامل است که همزاد مسئلهٔ پوشش کمینهٔ مجموعه‌هاست [۲]. کاهش در مجموعه آزمون باید به‌گونه‌ای انجام شود که قابلیت آشکارکنندگی خطای مجموعهٔ 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؛ چون هدف اصلی آزمون آشکارسازی خطاهاست [۵۴]. 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، چالش اصلی است.

وقتی مهندسان نرم‌افزار از آزمون پس‌نمایی برای تأیید اعتبار نرم‌افزارهای در حال تحول استفاده می‌کنند، اغلب با اجرای آزمایه‌های موجود شروع می‌کنند. اما این کار ممکن است کافی نباشد؛ زیرا تغییرات کد می‌تواند عملکرد جدید اضافه کند و پوشش آزمون را عوض کند. فنون افزایش برای حل این مسئله، ابتدا تشخیص می‌دهند که کدام بخش‌های برنامه جدید هستند و برای آن‌ها آزمایه‌های جدید مورد نیاز است؛ سپس این آزمایه‌ها را تولید می‌کنند [۷]. چند نکته اهمیت دارد:

- الگوریتم‌های دقیق برای حل مسایل NP سخت از مرتبه نمایی هستند [۵۲] پس الگوریتم‌های مناسب بر مبنای روش‌های اکتشافی، فرااکتشافی، تقریبی و تصادفی طراحی می‌شوند. این روش‌ها جواب‌های نزدیک به بهینه را در زمان کم تولید می‌کنند که باعث کاربردپذیری عملی و مقیاس‌پذیری آن‌ها می‌گردد. فنون کاهش هزینه‌های آزمون پس‌نمایی این‌طورند.
- فنون آزمون پس‌نمایی در صدد انتخاب آزمایه‌هایی هستند که نیازمندی‌های خاصی را برآورده سازند. پوشش کد۲۲ و قدرت آشکارکنندگی خطا[۲۳] دو جنبه مهم هستند [۱].
- پوشش کد را بر اساس معیارهایی[۲۴] در سطح ریزدانه[۲۵] یا درشت‌دانه[۲۶] می‌توان محاسبه کرد که دستورهای انشعابی[۲۷]، زوج تعریف و استفاده[۲۸] و فراخوانی زیربرنامه‌ها از انواع رایج آن هستند [۵۴]. برای محاسبه پوشش کد نیاز به فراکدگذاری[۲۹] [۵۹] و اجرای آزمایه‌ها روی نسخه فراکدگذاری شده داریم. (منظور از فرا کدگذاری، افزودن کدهای اضافه‌ای درون کد است که اطلاعاتی از اجرای کدهای به دست آورد.) اغلب فنونی که از پوشش کد در بخشی از مراحل محاسبات خود استفاده می‌کنند، فقط یک نوع پوشش را مورد استفاده

21- Representative set

22- Code coverage
23- Fault-exposing potential
24- Criteria
25- Fine-grained
26- Coarse-grained
27- Branch, condition, or edge coverage
28- Def-use pair
29- Instrument

قرار می‌دهند. نشـــان داده شـــده [۵۶] که اســـتفاده از چند اطلاع پوشـــش کد از جنس‌های دیگر ســـودمندی فنون را بیشتر می‌کند ولی ملاحظاتی همچون سربار محاسباتی و سودمندی بیشتری که به‌دست می‌آید را باید در نظر گرفت. به‌طور تجربی نشان داده شده [۵۵] اگر دو پوشش استفاده شود که پوشش دوم از جنس متفاوتی باشد و ریزدانه‌تر، ســـودمندی قابل توجهی حاصل می‌شود. نشان داده شده [۲۵] که دانه‌بندی آزمایه‌ها تأثیر اساســـی بر ســـودمندی فنون اولویت‌دهی دارد. به‌دســـت آوردن اطلاعات پوشش کد برای هر آزمایه در هر تغییر در سامانه‌های نرم‌افزاری بزرگ که با چندین زبان برنامه‌ســـازی نوشـــته شده‌اند و ناهمگن هستند بسیار دشوار است [۴۸].

• قدرت آشـــکار کنندگـــی خطا را بدون اجـــرای آزمایه نمی‌توان محاسبه کرد؛ بنابراین با روش‌هایی همچون فنون آزمون جهش[30] [۵۸] یا اســـتفاده از میزان پوشش کد آن‌را تقریب می‌زنند.

• فنون طراحی شـــده گاهی تک هدفه عمل می‌کنند؛ مثلاً طوری طراحی می‌شوند که حداکثر پوشش کد به‌دست آید. گاهی فنون با در نظر گرفتن چند هدف، سعی در بهینه‌سازی همزمان آن‌ها می‌کنند؛ مثلاً ضمن افزایش پوشش کد، سعی دارند آزمایه‌هایـــی را انتخاب کنند که زمان اجرای کمتری دارند [۵۷].

۳- انتخاب مقالات و اوزان ارزیابی

برای یافتن مقالات مناسب ســـه پایگاه ACM، IEEE و Springer در بین سال‌های ۲۰۱۰ تا ۲۰۱۶ با کلید واژه‌های Test Suite Reduction، Test Case Selection، Test Case Prioritization، Regression Test و Test Case Augmentation مورد جستجو قرار گرفت. پس از جستجو مقالاتی مربوط به مجلات و کنفرانس‌های مهم نرم‌افزار همچون ICSE، IS- STA، ASE، IEEE Transactions on Software Engineering و Empirical Software Engineering. بررسی و در مجموع

30- Mutation testing

جدول ۱: معیارهای ارزیابی به همراه مقادیرشان

کد معیار	نام معیار	مقادیر
C1	نوع فن	۱) کاهش؛ ۲) انتخاب؛ ۳) اولویت‌دهی؛ ۴) افزایش
C2	نوع معیار ارزیابی	۱) کاهش اندازه مجموعه آزمون؛ ۲) فقدان در کاهش قابلیت تشخیص خطا؛ ۳) FDR؛ ۴) زمان سرتاسری؛ ۵) تعداد آزمایه انتخاب شده؛ ۶) APFD؛ ۷) تعداد انشعاب‌های پوشش داده شده در نسخه جدید
C3	نوع برنامه‌های مورد استفاده برای ارزیابی	۱) محک؛ ۲) دلخواه؛ ۳) پیدا شده حاصل از جستجوی نظام‌یافته در مخازن کد
C4	بن‌انگاره (پارادایم) برنامه‌ها	۱) رویه‌ای؛ ۲) شیئ‌گرا؛ ۳) تابعی
C5	نوع زبان برنامه‌سازی	۱) C؛ ۲) C++؛ ۳) جاوا
C6	رویکرد روش	۱) پوشش محور؛ ۲) مدل محور؛ ۳) گراف دوقسمتی؛ ۴) برنامه ریزی خطی؛ ۵) حریصانه؛ ۶) نیازمندی؛ ۷) بازیابی اطلاعات؛ ۸) برش بندی؛ ۹) تحلیل متن؛ ۱۰) فاصله رشته‌ای
C7	نوع پوشش مورد استفاده در ارزیابی	۱) انشعاب؛ ۲) مسیر؛ ۳) دستور؛ ۴) تابع؛ ۵) بلوک اولیه؛ ۷) رده
C8	نوع خطاهای موجود	۱) کاشته شده؛ ۲) واقعی
C9	تعداد خطاهای به کار رفته	۱) تک خطایی؛ ۲) چند خطایی
C10	استفاده از آزمون فرض آماری	۱) بلی؛ ۲) خیر
C11	داشتن ابزار	۱) بلی؛ ۲) خیر
C12	نیازمند فراکدگذاری	۱) بلی؛ ۲) خیر

۲۹ مقاله انتخاب گشت. برای انتخاب اوزان ارزیابی به‌دنبال معیارهایی هســـتیم که حتی‌الامکان الف) اندازه‌گیری آن‌ها دقیق و علمی قابل انجام باشـــد؛ ب) قابل اندازه‌گیری برای همه فنون باشـــد (عمومـــی)؛ پ) قابـــل اندازه‌گیری بدون پیاده‌سازی باشند؛ ت) ســـبک وزن باشند. این معیارها به هیچ وجه جامع نیســـتند و به تنهایی قادر به ارزیابی فنون نیستند. با این حال فرض می‌شود که ترکیب این معیارهای ســـاده و ســـبک وزن با هم شهود بیشـــتری از ماهیت و عملکرد هر فن ایجاد می‌نماید. از آنجا که نام برخی معیارها طولانی است و برای این که از تکرار جلوگیری شود، به هر

معیار و مقدارش، یک کد اختصاص می‌دهیم که در جدول ۱ خلاصه شده‌اند.

۴- خلاصه روش‌ها و ارزیابی آن‌ها

در این قسمت خلاصه هر یک از مقاله‌های در نظر گرفته شده بیان می‌شود. سپس هر کدام از معیارهای ارزیابی که در مقالهٔ اصلی اشاره شده باشد به همراه مقدار آن معیار ذکر می‌گردد. برای مثال طبق جدول ۱، C3:1 به این معناست که در این مقاله، نوع برنامه‌های مورد استفاده برای ارزیابی، محک هستند.

٤-۱- فنون کاهش مجموعه آزمون

روش کاهش مجموعه آزمون برحسب تقاضا[31] را ارائه کرده‌اند تا مهندسان بتوانند حد بالای خاصی بر روی فقدان در قابلیت تشخیص خطای فنون کاهش بگذارند. این روش با تنظیم سطح اطمینان[32] داده شده c٪ و یک حد بالای ۱٪ بر روی فقدان قابل پذیرش در قابلیت تشخیص خطا، کمک می‌کند تا یک زیرمجموعهٔ نماینده را که نیازمندی‌های آزمون یکسانی با مجموعه آزمون داده شده ارضا می‌کند داشته باشیم و همچنین حداکثر ۱٪ فقدان در قابلیت تشخیص خطا را در حداقل c٪ از نمونه‌هایی که این فن اعمال می‌شود، داشته باشیم. این روش ابتدا یک جدول فقدان تشخیص خطا[33] را با جمع‌آوری آمارهایی در مورد فقدان در قابلیت تشخیص خطا در سطح دستورالعمل، با در نظر گرفتن سه سطح اطمینان (۹۰٪، ۹۵٪ و ۹۹٪)، ایجاد می‌کند. بعد از جمع‌آوری این اطلاعات، مسئله را با برنامه‌ریزی خطی عدد صحیح[34] مدل می‌کند. سپس با حل این مسئله، زیرمجموعهٔ نماینده تولید می‌گردد.

معیارها:	C1:1،	C2:1,2،	C3:1،	C4:2،	C5:1,3،	C6:1،	C7:3،	C8:1،	C9:2،	C10:2،	C11:2،	C12:2.

کومار[35] و همکاران [۹]، بیان کرده‌اند که بیشتر آزمایه‌هایی که برای آزمون نرم‌افزار طراحی می‌شوند، استفاده نمی‌شوند یا تکراری‌اند. پس این‌گونه آزمایه‌ها، هزینه مورد نیاز برای آزمون نرم‌افزار را افزایش می‌دهند. مقاله مذکور از فن خوشه‌بندی فازی[36] برای کاهش تعداد آزمایه‌ها استفاده کرده تا آزمون نرم‌افزار نتایج دقیق‌تر و به‌صرفه‌تری داشته باشد. استفاده از روش فازی به هنگام خوشه‌بندی نتایج دقیق‌تری را به ارمغان می‌آورد. در خوشه‌بندی سخت[37]، داده به خوشه‌های از هم مجزا تقسیم می‌شود، به‌طوری که هر عنصر داده‌ای دقیقاً متعلق به یک خوشه است. در خوشه‌بندی فازی که به آن خوشه‌بندی نرم[38] نیز گفته می‌شود، عناصر داده‌ای می‌توانند به بیش از یک خوشه تعلق داشته باشند. همچنین هر عنصر داده‌ای یک مجموعه از سطوح عضویت دارد. این بیانگر قدرت ارتباط میان آن عنصر داده‌ای و یک خوشهٔ خاص است. این مقاله از پیچیدگی سیکلوماتیک به عنوان حدس اولیه برای تعداد خوشه‌ها استفاده می‌کند. دلیل استفاده از این معیار این است که به نقطهٔ شروع مطمئنی احتیاج است تا بتوان سازوکار خوشه‌بندی فازی را آغاز نمود.

معیارها:	C1:1،	C2:1,2،	C3:2،	C4:1،	C5:1،	C6:1،	C7:1,2،	C8:2،	C9:1،	C10:2،	C11:2،	C12:2.

گوتلیب[39] و همکاران [۱۰]، از روش شبکهٔ جریان[40] برای کاهش اندازهٔ مجموعه آزمون استفاده کرده‌اند. در این مقاله سه مشکل حیطهٔ کاهش مجموعه آزمون شناسایی شده است و روش جدیدی پیشنهاد شده است که برای غلبه بر این سه مشکل، از جریانات بیشینهٔ شبکه[41] استفاده می‌کند. این سه مشکل عبارت‌اند از: الف) زیرمجموعهٔ کمینه، تضمین‌شده نیست؛ یعنی روش‌های موجود راه‌حل‌های تقریباً بهینه تولید می‌کنند؛ ب) راه‌حل‌های موجود، بین زمان کاهش و تعداد آزمایه‌ها مصالحه برقرار می‌کنند و نمی‌توانند به هر دو برسند. مثلاً روش‌های دقیق بر اساس برنامه‌ریزی خطی عدد صحیح، قادر هستند یک راه‌حل بهینه بیابند، اما با هزینهٔ زمان اجرای طولانی. در طرف مقابل، روش‌های حریصانه و اکتشافی از لحاظ زمان اجرا بهتر هستند، اما مجموعه آزمون

31- On-demand test suite reduction
32- Confidence level
33- Fault-detection-loss
34- Integer linear programming
35-Kumar

36-Fuzzy clustering technique
37-Hard clustering
38-Soft clustering
39- Gotlieb
40- Flow network
41- Network maximum flows

بزرگتری تولید می‌کننـد؛ در نتیجه نیاز به مصالحه وجود دارد؛ پ) قابلیت تشخیص خطا یا پوشش کد حفظ نمی‌شود. روش پیشنهادی که FLOWER نام دارد، روش دقیقی برای یافتن تعداد کمینهٔ آزمایه‌ها است که نیازمندی‌های یکسانی را نسـبت به مجموعه آزمون اصلی، پوشـش می‌دهد. این روش، مسئلهٔ کاهش را به شکل یک گراف دوقسمتی[42] مدل کرده است که ارتباط بین نیازمندی‌های آزمون و آزمایه‌ها را بیان می‌کند. این شبکه، محدودیت‌های ظرفیتی[43] خاصی دارد. روش FLOWER، از روش فورد-فولکرسـون[44] برای محاسـبهٔ جریانات بیشـینه و از فنون برنامه‌نویسی محدودیت‌دار[45] برای جستجو میان جریانات بهینه استفاده می‌کنـد. نویسـندگان روش FLOWER را بـا دو روش حریصانه و برنامه‌ریزی خطی عدد صحیح مقایسه کرده‌اند. از لحاظ زمان اجرایی، FLOWER به‌طور میانگین ۳۰٪ بیشتر از روش حریصانه به زمان احتیاج دارد؛ ولی بهتر از روش برنامه‌ریزی خطی عدد صحیح عمل می‌کند. از لحاظ توانایی کاهش اندازهٔ مجموعه‌های آزمون، FLOWER دقیقاً همانند روش برنامه‌ریـزی خطی عـدد صحیح عمـل می‌کند؛ در صورتی که روش حریصانه، مجموعه آزمون‌هایی با اندازهٔ ۱۰٪ الی ۱۵٪ بزرگ‌تـر از روش FLOWER و برنامه‌ریزی خطی عدد صحیح تولید می‌کند.

معیارها:	C1:1،	C2:1،	C3:2،	C6:3،	C10:2،	C11:1.

گوتلیب و همکاران [۱۱]، روشی مبتنی بر برنامه‌نویسی محدودیت‌دار برای کاهش پیشـنهاد کرده‌اند. در این مقاله سـه مدل برنامه‌نویسـی محدودیت‌دار مختلف ارائه شده اسـت. این مقالـه بیان مـی‌دارد که هر مسـئلهٔ کاهش را می‌توان توسـط یک گراف دوقسـمتی نشـان داد که یک قسمت نشان‌دهندهٔ نیازمندی‌ها و قسمت دیگر نشان‌دهندهٔ آزمایه‌ها اسـت. متغیـر دامنه[46]، متغیری اسـت که دامنهٔ محـدودی دارد. محققان سـه مدل ذکر شـده را بر روی سیستم‌های ویدئو کنفرانس سیسکو اعمال کرده‌اند. نتایج ارزیابی‌هایشـان نشـان می‌دهد که مدل Mixt هم از لحاظ کارایـی و هم بهره‌وری از دو مدل دیگر بهتر بوده اسـت. همچنین این مدل‌هـا با ابزار MINTS، ابـزار رایج کاهش مجموعه آزمون، مقایسه شده‌اند. نتایج ارزیابی بیانگر این اسـت که نه تنها Mixt از MINTS از لحاظ بهره‌وری بهتر است، بلکه مجموعه آزمون را نیز بیشتر کاهش می‌دهد.

معیارها:	C1:1،	C2:1،	C3:2،	C6:3،	C10:2،	C11:12.

٤-۲ فنون انتخاب آزمایه

همتـی و همکاران [۱۲]، روش انتخاب مبتنی بر مدل را ارائه کرده‌اند. در سـال‌های اخیـر، صنعت نرم‌افزار تمایل زیادی به خودکارسازی فرآیند تولید آزمایه‌ها با استفاده از مدل نرم‌افزارهای تحت آزمون[47] از خود نشان داده است. اما اجرای آزمون مبتنی بر مدل[48] نیاز به منابع زیادی دارد که در عمل، مشکلاتی را از لحاظ زمانی به وجود می‌آورد. بدین منظـور باید از آزمایه‌های موجـود، تعدادی آزمایه انتخاب کرد تـا نیازی به اجرای تمامی آزمایه‌ها نباشـد. فنون انتخاب در دو دستهٔ کلی انتخاب مبتنی بر پوشش[49] و انتخاب مبتنی بر تشابه[50] تقسیم‌بندی می‌شوند. فرض فنون انتخاب مبتنی بر پوشـش این است که هر چه آزمایه‌های انتخاب شده پوشش بیشـتری از کد یا مدل داشته باشند، خطاهای بیشتری می‌توانند کشف کنند. فرض فنون مبتنی بر تشابه این است که هر چه آزمایه‌ها تنوع بیشتری داشته باشند، خطاهای بیشـتری می‌توانند کشف کنند. این مقاله در دسـتهٔ دوم جای دارد. برای این‌که بتوان از دستهٔ دوم استفاده کرد، نیاز به یک معیار تشابه است تا بتوان تنوع یک مجموعه را با میانگین گرفتن از تشـابه مقادیر به صورت دودویی محاسبه کرد. بعد از تعریف معیار تشابه، نیاز به یک الگوریتم انتخاب اسـت تا زیرمجموعه‌ای از آزمایه‌ها کـه بین اعضایش، کمترین تشـابه دودویی وجود دارد را انتخاب نماید. این روش معیارهای شباهت را با استفاده از

42- Bipartite graph
43- Capacity
44- Ford-Fulkerson
45- Constraint programming
46-Domain variable

47-Software Under Test (SUT)
48-Model Based Testing (MBT)
49-Coverage-based Selection
50-Similarity-based Selection

محرک‌ها[۵۱] و محافظ‌ها[۵۲] بـر روی انتقالات[۵۳] مدل حالت[۵۴]، محاسـبه می‌کنـد و از یک الگوریتم ژنتیـک برای انتخاب آزمایه‌ها استفاده می‌کند. آزمایه‌ها از یک مدل حالت تولید می‌شـوند و به جای این‌که آزمایه‌های واقعی و قابل اجرا باشند، آزمایه‌های انتزاعی هستند. از آنجا که مقایسه‌های تشابه در سطح انتزاعی انجام می‌گیرند (اطلاعاتی که لازم نیسـتند را در نظر نمی‌گیرند) و آزمایه‌های قابل اجرا بعد از انتخـاب آزمایه‌های انتزاعی برای اجـرای مجدد تولید می‌گردند، هزینهٔ تولید آزمایه می‌تواند کاهش یابد. از چندین تابع تشابه مانند شمارش[۵۵]، همینگ[۵۶]، جاکارد[۵۷]، لونشتاین[۵۸]، سراسری و محلی برای ارزیابی روش پیشنهادی استفاده شده است. نتایج ارزیابی‌ها نشان داده است که استفاده از معیار تشابه ژاکارد بهترین نتیجه را داشته است و ۷۷٪ در هزینهٔ اجرای آزمون صرفه‌جویی شده است.

معیارها:	C1:2،	C2:3،	C3:2،	C4:2،	C5:2،	C6:2،	C8:2،	C9:2،	C10:2،	C11:1،	C12:2.

میرعرب و همکاران[۱۳]، روشی را معرفی کرده‌اند که با استفاده از برنامه‌ریزی خطی عدد صحیح و بهینه‌سازی چند معیاره، زیرمجموعه‌ای از آزمایه‌ها که اندازه‌اش از قبل مشخص می‌باشد را انتخاب می‌نماید. برخلاف روش‌های چندهدفـهٔ موجود، این تحقیـق بیان کرده اسـت که تابع بهینه‌سـازی چندمعیاره تلاش می‌کند تا به نرخ تشخیص خطای بالایی با اسـتفاده از آزمایه‌های انتخاب شده دست یابد. این روش ایمن نیسـت، به این علت که آزمایه‌هایی که می‌توانند خطاها را آشـکار سـازند را نادیده می‌گیرد. اما این روش وقتی که شـرکت تولید نرم‌افـزار با محدودیت زمان روبه‌رو اسـت، می‌تواند خیلی عملی باشد. در عمل، فرآیندهای تولید نرم‌افزار اغلب محدودیت‌های زمانی را بر روی آزمون پس‌نمایی اعمال می‌کنند؛ در نتیجه این روش می‌تواند به اسـتفادهٔ کارآمد از منابـع محدود تحت چنین شـرایطی کمک کند. این روش با چندین فن انتخاب (مبتنی بر پوشش و مبتنی بر شبکه باور بیزی) ارزیابی شده است. نتایج ارزیابی‌ها نشان داده است که روش پیشنهادی نسبت به روش‌های مقایسه شده، بهتر عمل می‌کند.

گلیگوریچ[۵۹] و همکاران [۱۴]، فن جدیدی به نام اکستازی[۶۰] ارائه کرده‌اند که انتخاب را با اسـتفاده از وابستگی پویای آزمایه‌ها بـه فایل‌ها، انجـام می‌دهد. نویسـندگان معیار جدیدی برای ارزیابی فن انتخاب به نام زمان سرتاسری[۶۱] معرفی کرده‌اند. نویسندگان بیان داشته‌اند در صورتی یک فن انتخاب مناسب اسـت و نسبت به فن باز آزمودن همهٔ آزمایه‌ها بهتر است که به‌طور میانگین زمان سرتاسری آن کوتاه‌تر از زمان مورد نیـاز برای اجرای تمامی آزمایه‌ها باشـد. یک فن انتخاب شامل سه مرحله است: الف) مرحله تحلیـل که آزمایه‌هـا را برای اجرا کـردن انتخاب می‌کند؛ ب) مرحله اجرا که آزمایه‌های انتخاب شده را اجرا می‌کند؛ پ) مرحله جمع‌آوری که اطلاعات اجرایی از نسـخهٔ فعلی را برای تحلیل نسـخهٔ بعدی جمع‌آوری می‌نماید. بیشـتر تحقیقات، فنون انتخاب را بر اسـاس زمان اجرایی مرحله دوم ارزیابی کرده‌انـد. برای این‌که بتوان به‌طور منصفانه فنـون انتخـاب را ارزیابـی کـرد، در نظر گرفتـن زمان سرتاسـری – از آغاز اجرای مجموعه آزمون برای نسخهٔ جدید تا زمانی که تمامی آزمایه‌ها در دسترس باشند – که تولیدکننده مشـاهده می‌کند الزامی است. اکستازی با ابزار معروف FaultTracer نیز مقایسه شده است. نتایج ارزیابی نشـان می‌دهند که اکستازی به نسـبت FaultTracer زمان سرتاسـری خیلی کمتـری دارد. همچنیـن از لحاظ تعداد آزمایه‌های انتخابی، اکستازی بهتر عمل می‌کند.

معیارها:	C1:2،	C2:4,5،	C3:2،	C4:2،	C5:3،	C6:1,2،	C7:4,6،	C10:2،	C11:1،	C12:1.

لگونسـن[۶۲] و همکاران [۱۵]، بیان کرده‌اند که کارهای انجام شـده در حـوزهٔ انتخاب آزمایـه را می‌توان در دو دسـتهٔ کلی پویا و ایستا تقسـیم‌بندی کرد. یک فن انتخاب پویا به دو نوع اطلاعات نیاز دارد: الف) اطلاعات تغییر بین

51-Trigger
52- Guard
53- Transition
54- State model
55- Counting
56- Hamming
57-Jaccard
58-Levenshtein

59-Gligoric
60-EKSTAZI
61-End-to-end time
62-Legunsen

دو نسخهٔ نرم‌افزار؛ ب) وابستگی‌های آزمون که به‌طور پویا حین اجرای آزمون‌ها بر روی نسخهٔ قدیمی نرم‌افزار، محاسبه می‌گردند. با استفاده از این اطلاعات، این فن تحلیل می‌کند که چگونه تغییرات کد با وابستگی‌ها تعامل می‌کند تا زیرمجموعه‌ای از آزمون‌ها که با تغییرات کد در ارتباط هستند را تشخیص دهد. فنون پویا از سال ۱۹۹۳ توجه محققان زیادی را به خود جلب کرده‌اند که در بین آن‌ها، FaultTracer و اکستازی را می‌توان از جدیدترین فنون انتخاب نام برد. فنونی که وابستگی‌های آزمون با ریزدانگی سطح پایین را جمع‌آوری می‌کنند، ممکن است که دقیق‌تر بوده و تعداد آزمایه‌های کمتری را برای اجرا انتخاب نمایند؛ اما به سربار تحلیل بیشتری نیاز دارند. در طرف مقابل، فنونی که وابستگی‌های آزمون با ریزدانگی سطح بالا را جمع‌آوری می‌کنند، ممکن است کمتر دقیق باشند ولی سربار تحلیل کمی دارند. همانطور که قبلاً در این گزارش بیان شد، اکستازی فن پویایی برای انتخاب می‌باشد. به دلیل نیاز داشتن این فن به وابستگی‌هایی که به‌طور پویا جمع‌آوری می‌شوند، کاربرد این فن در عمل با محدودیت مواجه است. عواملی که باعث محدودیت می‌شوند، عبارتند از: الف) وقتی از فنون پویا برای انتخاب استفاده می‌کنیم، وابستگی‌های پویای آزمون برای نسخهٔ قدیمی ممکن است همیشه در دسترس نباشند، مثلاً وقتی که این فنون را برای اولین بار، به پروژه‌ای که چندین نسخهٔ قبلی دارد، اعمال می‌کنیم؛ ب) جمع‌آوری وابستگی‌های پویای آزمون برای پروژه‌های بزرگ ممکن است امری زمان‌بر باشد؛ پ) در سامانه‌های بیدرنگ به دلیل این‌که فراکدگذاری برای تشخیص وابستگی‌ها ممکن است باعث ایجاد وقفه در روند عادی اجرا شود، انتخاب پویا ممکن است قابل اعمال نباشد؛ ت) برای برنامه‌های غیرقطعی (به دلیل ماهیت تصادفی یا همروندی)، وابستگی‌هایی که به‌طور پویا جمع‌آوری می‌شوند ممکن است تمامی مسیرهای ممکن را پوشش ندهند و منجر شود تا این فنون ایمن نباشند. نقطهٔ مقابل فنون پویا، فنون ایستا هستند که از تحلیل ایستای برنامه برای انتخاب ایمن آزمایه استفاده می‌کنند. این فنون تاکنون برای پروژه‌های واقعی مورد مطالعه قرار نگرفته‌اند. همچنین تاکنون مشخص نشده است که کدام سطح از ریزدانگی آزمون برای فنون ایستا مناسب‌تر است. اگرچه کار اخیر [۱۴] بر روی فنون پویا نشان داده است که ریزدانگی سطح رده نتایج بهتری را نسبت به سطح روش فراهم می‌کند.

در این مطالعه دو فن ایستا به نام‌های ClassSRTS و MethSRTS با فن اکستازی مقایسه شده‌اند. فن ClassSRTS، فن ایستای انتخاب در سطح ردهٔ و MethSRTS، فن ایستای انتخاب در سطح روش می‌باشد. نتایج آزمایش‌ها نشان می‌دهد که اکستازی، ClassSRTS و MethSRTS به ترتیب ۲۰/۶٪، ۴/۲۹٪ و ۴۳/۸٪ از آزمایه‌ها را برای اجرا انتخاب می‌کنند. جهت یادآوری، فنون ایمن انتخاب، فنونی هستند که تمامی آزمایه‌هایی که مرتبط با تغییرات هستند را انتخاب نمایند و فنون انتخاب دقیق، فنونی هستند که تنها آزمایه‌هایی را انتخاب کنند که با تغییرات مرتبط هستند. در مقایسه MethSRTS و ClassSTRS از لحاظ دقیق بودن و ایمن بودن، ClassSRTS به ترتیب در ۰/۲٪ و ۳۳٪ از برنامه‌ها ناامن و نادقیق بوده و MethSRTS به ترتیب در ۱۰/۶٪ و ۵۵/۷٪ از برنامه‌ها ناامن و نادقیق بوده است اما میزان ناامنی اکستازی خیلی کمتر از هر دو بوده است. نتیجهٔ مهم این تحقیق این است که فنون ایستای انتخاب در سطح رده، قابل مقایسه با اکستازی هستند و نتایج امیدوارکننده‌ای دارند و محققان باید بهبودهایشان را در این حوزه ادامه دهند.

معیارها:	C1:2،	C2:4,5،	C3:2،	C4:2،	C5:3،	C6:1,2،	C7:4,6،	C10:2،	C11:1،	C12:1.

۳-٤ فنون اولویت‌دهی آزمایه

عرفین و همکاران [۱۶] از اطلاعات نیازمندی‌ها برای بهبود اولویت‌دهی آزمایه استفاده کرده‌اند. سامانه‌های نرم‌افزاری بر اساس نیازمندی‌ها توسعه داده شده‌اند و نیازمندی‌های خاصی، نسبت به بقیه مهم‌تر (شامل ویژگی‌هایی هستند که توسط بیشتر توسط کاربران

مورد اســتفاده قرار می‌گیرد) و خطاخیزتر[۶۳] هستند. اغلب آزمونگران دانش محدودی برای درک مشکلات محصولات نرم‌افزاری دارند کــه در چنین حالتی، اطلاعات نیامندی‌ها به‌طور بالقوه می‌تواند در تشخیص منشأ خطاها کمک کند. به علاوه، با ایجاد ارتباط میان نیازمندی‌ها، کد و خطاهای کشف‌شــده حین آزمــون پس‌نمایی، مهندســان نرم‌افزار می‌توانند محصولات نرم‌افزاری را به‌طور جامعی نگهداری نمایند. در نتیجه می‌توان محصولات ایمن‌تری را توســعه داد. نویسندگان بیان کرده‌اند با وجود این‌که اهمیت دخیل کردن اطلاعات نیازمندی‌ها در حین مرحله آزمون نرم‌افزار، به‌خوبی توسط جامعهٔ مهندسان نرم‌افزار درک شده است، تنها تعداد کمی از محققان، استفاده از نیازمندی‌ها در آزمون نرم‌افزار را مورد مطالعه قرار داده‌اند. برای مثال مقالهٔ [۱۷] گزارش کرده است که آزمایه‌های اولویت‌دهی‌شده بر اساس اهمیت و خطاخیزبــودن نیازمندی‌ها، قادر بوده اســت تا خطاهای دشوار را زودتر کشف کند. معمولاً نیازمندی‌های مرتبط یا مشــابه در ردهٔ یکسان یا در رده‌هایی که در یک ســامانه فرعی داخلی هستند، پیاده‌سازی می‌شوند که این کار باعث منسجم‌ترشــدن محصول نرم‌افزاری می‌شود. این بدان معنا است که آزمایه‌های مربوط به نیازمندی‌های مرتبط یا یکســان، مجموعــه رده‌های مشــابهی را مورد آزمــون قــرار می‌دهنــد. در کار قبلی همین نویســندگان [۱۸]، مشــاهده شده اســت که آزمایه‌های با خصوصیات مشترک، متمایل به داشتن قابلیت تشخیص خطای یکسان هستند. اولویت‌دهی آزمایه‌ها با در نظر گرفتن ارتباط میان آزمایه‌ها و نیازمندی‌هــا، می‌تواند به بهبود فرآیند آزمون پس‌نمایی کمک کند. از این رو، هدف این تحقیق این اســت که آیا خوشه‌بندی آزمایه‌ها بر اساس شباهت نیازمندی‌ها می‌تواند کارایی فنون پس‌نمایی، مخصوصاً اولویت‌دهی را بهبود دهد یا خیر. نتایج ارزیابی این تحقیق نشــان می‌دهد که روش خوشــه‌بندی صرف نظر از روش انتخابی که به کار برده است، کارا می‌باشد.

معیارها:	C1:3،	C2:6،	C3:2،	C4:2،	C5:3،	C6:6،	C8:1،	C9:2،	C10:2،	C11:2،	C12:2.

63- Fault-prone

توماس[۶۴] و همــکاران [۱۹]، اولویت‌دهــی آزمایه‌ها را به صورت ایســتا و با اســتفاده از مدل‌های موضوعی[۶۵] انجام داده‌اند. نویســندگان بیان داشته‌اند که بیشتر فنون اولویت‌دهی که در تاکنون ارائه شده‌اند، از رفتار اجرایی یا مدل خصوصیات هر آزمایه برای هدایت فرآیند اولویت‌دهی استفاده می‌کنند. در این مقاله حالتی مفروض است که این منابع اطلاعاتی در دســترس نیســتند و همچنین کد منبع ســامانه تحت آزمون نیز در دسترس نیست. در نتیجه فن ارائه شده در دستهٔ فنون ایستای جعبه سیاه می‌باشد. این فن یــک الگوریتم تحلیل متن به نام مــدل موضوعی را بر رویدادهای زبانی[۶۶] آزمایه‌ها که شامل توضیحات، ثابت‌ها و رشته‌ها هستند، اعمال می‌کند تا عملکرد هر آزمایه راتقریب بزند که به فن پیشنهادی این امکان را می‌دهد تا اولویت بالا را به آزمایه‌هایی بدهد که عملکردهای مختلفی از نرم‌افزار تحــت آزمون را بیازمایند. در واقع این فن از داده‌های مهم که درآزمایه‌ها تعبیه شــده‌اند، اســتفاده می‌کنــد. ایدهٔ این روش از آن‌جا آمده است که پژوهش‌های اخیر یافته‌اند که موضوع‌هایی که از داده‌های زبانی کشف می‌شوند، تقریب خوبی از عملکرد در کد منبع را نشــان می‌دهند. این مقاله فرض کرده اســت وقتی دوآزمایه، موضوع‌های یکســان داشته باشــند، در عملکرد مشــابه‌اند و خطاهای یکسانی را تشــخیص خواهند داد. با وجــود داده‌های زبانی، طیف وســیعی از الگوریتم‌های تحلیل متــن را می‌توان بر روی این داده‌ها اعمال کرد. در یک ســوی این طیف، می‌توان از الگوریتم‌های ســادهٔ مقایسه متن مانند تساوی رشته‌ها استفاده کرد.

با وجودی که این روش سریع و ساده است، اما پتانسیل نادیده گرفتن جنبه‌های مهم آزمایه‌ها وجود دارد. در طرف دیگر، پردازش زبان طبیعی با اســتفاده از درخت تجزیه و دســتور زبان می‌باشد که نیازمند داده‌های آموزشی است و کارایی لازم را ندارد و خودکارســازی اولویت‌دهی را با مشــکل مواجه می‌سازد. بین این دو ســر طیف، مدل‌های

64- Thomas
65- Topic modeling
66-Linguistic data

موضوعی قرار دارند که از مدل کیسهٔ کلمات[67] و رخداد کلمات برای تقریب عملکرد هر آزمایه استفاده می‌کنند. مزیت این روش این است که سریع است و به صورت بدون نظارت[68] انجام می‌گیرد و به هیچ دادهٔ آموزشی یا طبقه‌بندیِ از قبل تعریف شده نیاز ندارد. این روش با سه روش اولویت‌دهی تصادفی، نسخهٔ جعبه سیاه گراف فراخوانی[69] و مبتنی بر رشته مقایسه شده است. مشاهده شده است که این روش ۳۱٪ از روش‌های ذکر شده کاراتر می‌باشد و بهتر عمل کرده است.

معیارها:	C1:3،	C2:6،	C3:1،	C4:2،	C5:3،	C6:9،	C8:1،	C9:2،	C10:1،	C11:2،	C12:2.

رودرمل و همکاران [۲۰]، روشی ارائه کرده‌اند که از فراوانی واژه[70] و معکوس فراوانی سند[71] که از مفاهیم بازیابی اطلاعات[72] هستند، برای غلبه بر محدودیت‌های فنون مبتنی بر پوشش استفاده می‌کند. امتیاز TF در اولویت‌دهی، فراوانی پوشش یک عنصر کد توسط یک آزمایه است؛ هرچه یک آزمایه، عنصر خاصی از کد را چندین بار اجرا کند، آن آزمایه امتیاز TF بیشتری به آن عنصر می‌دهد. مفهوم TF ممکن است برای یافتن خطاهایی که توسط اجراکردن چندین بارِ کد خاصی ناشی می‌شوند، مناسب باشد. مثلاً به کارگرفتن چندین بارهٔ یک شرط ممکن است شانس ملاقات دیگر حالت‌های برنامه را افزایش دهد. در این حالت، آزمایه‌هایی که عناصری از کد با امتیاز TF بالا را پوشش می‌دهند، ممکن است شانس بیشتری نسبت به سایر آزمایه‌ها برای تشخیص خطاها در برنامه را داشته باشند. امتیاز DF یک عنصر کد در اولویت‌دهی، متناسب است با تعداد آزمایه‌هایی که آن عنصر را می‌آزمایند. هر چه تعداد آزمایه‌هایی که یک عنصر را پوشش می‌دهند افزایش یابد، امتیاز DF آن عنصر افزایش می‌یابد. مقدار IDF معکوس DF است؛ در نتیجه از امتیاز IDF به منظور یافتن عناصر نادر کد استفاده می‌شود. آزمایه‌هایی که تعداد زیادی از عناصر با IDF بالا را پوشش می‌دهند، ممکن است ظرفیت بالایی برای یافتن خطاها در برنامه را داشته باشند؛ زیرا عناصر منحصربه فردِ زیادی را پوشش می‌دهند. این مقاله پیشنهاد کرده است زمانی که وابستگی بین پوشش و قابلیت تشخیص خطا کم باشد، روش ارائه شده نسبت به روش‌های مبتنی بر پوشش قابل اعتمادتر است. در غیر این صورت بهتر است از روش‌های مبتنی بر پوشش برای اولویت‌دهی استفاده کرد.

معیارها:	C1:3،	C2:6،	C3:1،	C4:2،	C5:3،	C6:1,7،	C7:1,3,4،	C8:1،	C9:2،	C10:2،	C11:2،	C12:2.

ساها[73] و همکاران [۲۱]، روشی مبتنی بر بازیابی اطلاعات را برای اولویت‌دهی ارائه کرده‌اند. بیشتر سندهایی که توسط مهندسان نرم‌افزار تولید می‌شود، متنی هستند و تولیدکنندگان نرم‌افزار از اسامی معناداری برای توضیحات درون منبع کد و نام متغیرها استفاده می‌کنند. همچنین حین نوشتن آزمایه‌ها از اسامی مشابهی با اسامی که در کد به کار برده‌اند، استفاده می‌کنند. این تحقیق مسئلهٔ اولویت‌دهی را به یک مسئلهٔ بازیابی اطلاعات تبدیل کرده است. تفاوت بین دو نسخه، پرس‌وجو[74] است و آزمایه‌ها، مجموعه اسناد[75] هستند. سپس آزمایه‌ها بر اساس میزان تشابه بین تفاوت‌های برنامه و آزمایه‌ها امتیازدهی می‌شوند. در این روش از اطلاعات پوشش استفاده نشده است.

معیارها:	C1:3،	C2:6،	C3:1,2،	C4:2،	C5:3،	C6:7،	C8:1,2،	C9:2،	C10:2،	C11:1،	C12:2.

پاندا[76] و همکاران [۲۲]، روشی طراحی کرده‌اند که تنها آن قسمت‌هایی از برنامه که توسط تغییرات، تحت تأثیر قرار گرفته‌اند، مشخص شده و دوباره آزمون می‌گردند. این مقاله روشی ایستا برای اولویت‌دهی آزمایه‌های برنامه‌های شیئ‌گرا پیشنهادکرده است. گزارش شده که واحدی که مقدار اتصال[77] بالایی دارد، امکان وجود خطای بیشتری نسبت به واحدهای دیگر دارد؛ در نتیجه آزمایه‌ای که واحدی با مقدار اتصال بیشتر را اجرا می‌کند، خطاهای بیشتری را نسبت به دیگر آزمایه‌ها آشکار می‌نماید؛ از این رو این مقاله از معیار اتصال بخش‌های تحت تأثیر

67-Bag-of-word Model
68-unsupervised
69-Call graph
70-Term Frequency (TF)
71-Inverted Document Frequency (IDF)
72-Information Retrieval
73- Saha
74- Query
75- Document collection
76- Panda
77- Coupling

برنامه بـــرای اولویت‌دهی آزمایه‌ها اســـتفاده می‌کند. این روش ابتـــدا، یک گراف میانی از برنامه را با در نظر گرفتن وابســـتگی‌های ممکن بین عناصر مختلف برنامه می‌سازد. سپس با استفاده از برش‌بندی[۷۸] و با توجه به تغییری که به برنامه داده شده است، گرافی حاصل می‌شود به‌نام گراف برش تأثیریافته[۷۹] نشان‌دهنده گره‌های تحت تأثیرِ تغییرات و وابستگی‌های‌شـــان. ســـپس برای هر گره گراف، اتصال مولفهٔ تأثیریافته[۸۰] محاســـبه می‌گردد. هر گرهی که مقدار ACC اش بیشـــتر باشد، نشـــان‌دهندهٔ این است که احتمال خطاخیـــز بـــودن آن گره بالاتر اســـت. در واقـــع گره‌های دیگر وابســـتگی بیشتری نســـبت به این گره دارند. سپس مقادیر ACC برای تمامی گره‌ها، خوشه‌بندی[۸۱] می‌شـــوند. خوشـــه‌بندی به این علت اســـت که تمامـــی گره‌های ASG به‌طـــور مســـاوی خطاخیز نیســـتند و برخـــی، از برخی دیگرخطاخیزترند. ســـپس تمامی آزمایه‌ها اجرا می‌شوند تا اطلاعات پوشـــش آن‌ها استخراج گردد. وزن هر آزمایه برابر می‌شـــود با مجموع وزن گره‌هایی که پوشـــش داده است. در نهایت آزمایه‌ها بر اساس مجموع وزن گره‌هایی که پوشش داده‌اند، اولویت‌دهی می‌شوند.

معیارها: C1:3، C2:6، C3:1،2، C4:2، C5:3،3، C6:1،8، C7:4، C8:1، C9:2، C10:2، C11:2، C12:1.

اقبالی و همکاران [۲۳]، بیان کرده‌اند که روش‌های رایج بـــرای اولویت‌دهی، از اطلاعات آزمایه‌های قبلاً اجرا شـــده، مانند اطلاعات پوشش استفاده می‌کنند. کارایی این روش‌ها در صورتـــی که تعداد گره‌هـــا[۸۲] در گام‌هـــای اولویت‌دهی افزایش یابد، کاهش می‌یابد. این مقاله با اســـتفاده از ترتیب الفبایـــی[۸۳]، روش جدیـــدی ارائـــه کرده کـــه در فنون مبتنی بر پوشـــش، گره‌ها را می‌شـــکند و به‌طور قابل توجهی نرخ تشخیص خطا را افزایش می‌دهد. فنون حریصانه و تکراری اولویت‌دهی را بر اســـاس پوشش انجام می‌دهند که شامل دو فن کامل[۸۴] و افزایشی[۸۵] هستند. این فنون در هر قدم، یک آزمایـــه از آزمایه‌هایی را که قبلاً انتخاب نشـــده‌اند انتخاب می‌نمایند. هدف فنون کامل این است که تعداد عناصری که پوشـــش داده می‌شوند، بیشینه گردد. در صورتی که فنون افزایشـــی می‌خواهند آزمایه‌هایی زودتر انتخاب شـــوند که بیشترین تعداد از عناصری که در قدم‌های قبلی پوشش داده نشده‌اند، پوشش داده شوند. بنابراین در هر مرحله از فنون کامل، انتخاب آزمایه مســـتقل از انتخاب‌های قبلی است. اما درفنون افزایشی، انتخاب در هر مرحله، تأثیر آزمایه‌هایی که قبلاً انتخاب شده بودند را نیز درنظرمی‌گیرد. در هر قدم از فنون افزایشی، احتمال این که تساوی اتفاق بیفتد وجود دارد؛ یعنی بیش از یک آزمایه وجود دارد که بیشترین پوشش از موجودیت‌های هنوز پوشش داده نشده را دارد. اگر تساوی اتفاق بیفتد، فنون افزایشی به‌طور تصادفی یکی از آزمایه‌ها را انتخاب می‌کند. هنگامی که تساوی اتفاق می‌افتد، انتخاب تصادفـــی، به دو علـــت از لحاظ کارایی مشـــکل دارد: الف) تعداد دفعات رخداد تســـاوی زیاد اســـت.؛ ب) وقتی تساوی اتفاق می‌افتد، تعداد آزمایه‌هایی که می‌توانند انتخاب شوند، زیاد هســـتند؛ در نتیجه انتخاب تصادفـــی، به نوعی انتخاب حریصانه اســـت. از این رو اگر آزمایه‌ها به‌طور نظام‌یافته انتخاب گردند، مزیت محســـوب می‌شود. این روش به جای این‌که موجودیت‌ها را به دو گروه پوشش داده شده و هنوز پوشش داده نشـــده طبقه‌بندی کند، آن‌ها را بر اساس تعداد دفعاتی که تا قدم جاری پوشش داده شده‌اند، مرتب می‌کند. بـــا این کار، تمام موجودیت‌ها در انتخـــاب آزمایهٔ بعدی در نظر گرفته می‌شـــوند. پس ایدهٔ کلیدی این مقاله این اســـت که موجودیت‌هایی که کمتر پوشـــش داده شده‌اند، اولویت بالاتری برای انتخاب دارند. به عبارتی وقتی تســـاوی اتفاق بیفتد، آن آزمایه‌ای که پوشش حداکثری از موجودیت‌هایی که یک بار تاکنون پوشش داده شده‌اند دارد، انتخاب می‌گردد و به همین ترتیب الی آخر. پیاده سازی این ایده با استفاده از ترتیب الفبایی به خوبی انجام می‌گیرد.

معیارها: C1:3، C2:6، C3:1، C4:2، C5:3، C6:1،5، C7:4،5 ، C8:1، C9:2، C10:2، C11:1، C12:1.

هائو و همکاران [۶]، تفاوت ترتیبی که فنون افزایشی تولید

78- Slicing
79- Affected Slice Graph (ASG)
80- Affected Component Coupling (ACC)
81-Clustering
82- Tie
83- Lexicographical ordering
84- Total technique
85-Additional technique

می‌کند را با ترتیب بهینه، از لحاظ پوشش بررسی کرده‌اند. همان‌طور که می‌دانیم، فنون اولویت‌دهی به دنبال تشخیص قابلیت تشخیص خطای آزمایه‌ها هستند تا بتوانند آن‌ها را به درستی اولویت‌دهی کنند؛ اما از آنجا که پیش از اجرای آزمایه‌ها، قابلیت تشخیص خطایشان مشخص نیست، باید به دنبال هدف میانی بود. تاکنون، فنی که رودرمل به نام فن مبتنی بر پوشش افزایشی، ارائه کرده است، کاراترین روش برای هدف میانی بوده است. با توجه به نتایج تجربی فوق‌العاده خوب فن مبتنی بر پوشش افزایشی، نویسندگان در مورد این‌که چقدر اختلاف و تفاوت بین ترتیب آزمایه‌های حاصل از فن مبتنی بر پوشش افزایشی و ترتیب آزمایه‌ها با مقدار بهینهٔ هدف میانی وجود دارد، کنجکاو شدند. برای پی بردن به تفاوت هزینه و کارایی بین فن مبتنی بر پوشش افزایشی و ترتیب بهینه، مطالعهٔ تجربی بر روی ۱۰ پروژه انجام گرفته و خصوصیات تجربی این دو فن در مقایسه با یکدیگر وارسی شده است. به منظور این که بتوان این مطالعه را انجام داد، نویسندگان فن اولویت‌دهی مبتنی بر پوشش بهینه را به صورت مسئله برنامه‌ریزی خطی عدد صحیح مدل کرده‌اند و در نتیجه قادر به رسیدن به ترتیب بهینه برحسب پوشش، در زمان قابل قبول با استفاده از حل‌کننده‌های موجود برنامه‌ریزی خطی عدد صحیح، برای برنامه‌های بزرگ بوده‌اند. علاوه بر آن، برای دانستن حد بالای اولویت‌دهی، فن اولویت‌دهی بهینهٔ ایده‌آل را نیز پیاده سازی کرده‌اند که ترتیب اجرای آزمایه‌ها را بر اساس تعداد خطاهای تشخیص داده شده مرتب می‌کند. اگر چه فن ایده‌آل، عملی نیست، اما به عنوان یک فن کنترلی عمل می‌کند.

بنابراین همانطور که اشاره شد، این مقاله سه فن را با هم مقایسه می‌کند: الف) فن مبتنی بر پوشش افزایشی که رودرمل پیشنهاد کرده است؛ ب) فن مبتنی بر پوشش بهینه که اولویت‌دهی را به گونه‌ای با استفاده از برنامه‌ریزی خطی انجام می‌دهد تا پوشش بهینه را به دست می‌آورد؛ پ) فن بهینهٔ ایده‌آل که به عنوان فن کنترلی است و بهینه‌ترین ترتیب آزمایه‌ها را نشان می‌دهد که قابلیت تشخیص خطای بیشینه دارند. حال برای ارزیابی و مقایسهٔ این سه فن، دو مطالعه تجربی انجام داده‌اند.

مطالعهٔ اولی که انجام داده‌اند، دو فن اول باهم مقایسه شده است. از این مطالعه سه نتیجه به دست آمده است: الف) فن مبتنی بر پوشش بهینه از لحاظ پوشش، اندکی از فن افزایشی بهتر است؛ ب) فن مبتنی بر پوشش بهینه از لحاظ قابلیت تشخیص خطا خیلی ضعیف‌تر از فن افزایشی عمل می‌کند؛ پ) فن مبتنی بر پوشش افزایشی از لحاظ بهره‌وری خیلی ضعیف‌تر از فن افزایشی عمل می‌کند.

مطالعه دومی که صورت گرفته، هر سه فن با هم مقایسه شده‌اند. نتایج به دست آمده بدین صورت هستند: فن مبتنی بر پوشش بهینه از لحاظ پوشش خیلی بهتر از فن بهینهٔ ایده‌آل عمل می‌کند، اما از لحاظ قابلیت تشخیص خطا، خیلی ضعیف‌تر عمل می‌کند. همچنین فن مبتنی بر پوشش افزایشی از لحاظ پوشش خیلی بهتر از فن بهینهٔ ایده‌آل عمل می‌کند، اما از لحاظ قابلیت تشخیص خطا، خیلی ضعیف‌تر عمل می‌کند. نتیجهٔ ارزشمندی که می‌توان از این دو مطالعهٔ تجربی به دست آورد، این است که نباید با در نظر گرفتن پوشش به عنوان هدف میانی، به دنبال بهینگی در اولویت‌دهی باشیم و باید هدف میانی به‌طور محتاطانه‌ای انتخاب گردد.

معیارها:	C1:3،	C2:6،	C3:2،	C4:2،	C5:3،	C6:1,5,9,10،	C7:3،	C8:1،	C9:2،	C10:1،	C11:2،	C12:2.

لو[86] و همکاران [۲۴]، فنون اولویت‌دهی را در دنیای واقعی تولید نرم‌افزار مورد بررسی قرار داده‌اند. نویسندگان بیان داشته‌اند که فنون فعلی، تمرکزشان بر روی ارزیابی بر اساس تغییرات مصنوعی ساده بر روی کد منبع و آزمون‌ها است. برای مثال، تغییرات در منبع کد معمولاً خطاهای کاشته‌شده[87] هستند و مجموعه آزمون اصلاً افزایش[88] نمی‌یابد. در طرف مقابل، در دنیای واقعی تولید نرم‌افزار، تغییرات مختلفی در منبع کد و مجموعه آزمون سامانه‌های نرم‌افزاری ایجاد می‌شود. از این رو،

86-Lu
87-Seeded
88-Augment

واضح نیســت که نتایجی که تاکنون، تحقیقات مختلف در اولویت‌دهــی به دســت آورده‌اند، در دنیــای واقعی تولید نرم‌افزار معتبر هســتند یا خیر. در این مقاله، اولین مطالعهٔ تجربی برای بررسی تهدید اعتبار فنون اولویت‌دهی انجام شــده اســت. برای این کار، ۲۴ فن مختلف بر روی هشت برنامــهٔ واقعی آزمایش شــده‌اند. نتایــج ارزیابی‌ها بدین صورت هســتند: الف) هرچه از اطلاعات پوشش به‌روزتر برای اولویت‌دهی استفاده گردد، کارایی همهٔ فنون افزایش می‌یابــد؛ ب) همهٔ فنون زمانی که مجموعه آزمون، افزایش داشته باشد، کارایی‌شان به شدت افت می‌کند؛ پ) همهٔ فنون زمانی که تنها تغییرات در کد داشــته باشــیم، کارایی‌شان زیاد تغییر نمی‌کند و پایدار هستند.

معیارها:	C1:3,4،	C2:6،	C3:2،	C4:2،	C5:3،	C6:1,4,5،	C7:1,3,4،	C8:1،	C9:2،	C10:1،	C11:2،	C12:1.

لیو[89] و همکاران [۲۵]، فنون ایستا و پویای اولویت‌دهی را به‌طور تجربی مورد مقایســه قرار داده‌اند. فنون ایســتا به‌طور مستقیم با کد منبع و آزمایه سر و کار دارند. فنون پویا بر اســاس اطلاعات اجرایی عمل می‌کنند. نویسندگان سه فن ایســتا به نام‌های فن مبتنی بر گراف فراخوانی‌ها، فــن مبتنی بر فاصلهٔ رشــته[90] و فن مبتنی بــر عنوان را با چهار فن پویا به نام‌های فن حریصانه کامل، فن حریصانه افزایشــی، فن تصادفی تطبیقی و فن مبتنی بر جستجو از لحاظ کارایی، بهره‌وری و تشابه خطاهای کشف شده، در ریزدانگی‌های مختلف آزمایه (برای مثال ســطح روش و رده) مورد مقایســه قرار داده‌اند. نتایج آزمایش‌ها عبارتند از:

• تفاوت زیادی از لحاظ کارایی بین فنون مورد مطالعه وجــود دارد. به‌طور میانگین، فن ایســتای مبتنی بر گراف فراخوانی در سطح رده، کاراترین روش است، در حالی که فن پویای افزایشــی در سطح روش، کاراترین روش است. به‌طور کلی، فنون ایســتا در ســطح رده از فنون پویا بهتر عمل می‌کنند. اما فنون پویا در ســطح روش از فنون ایستا بهتر عمل می‌کنند.

• ریزدانگی آزمون به‌طور چشــمگیری، کارایی فنون اولویت‌دهــی را تحت تأثیر قرار می‌دهــد. تمامی فنون مطالعه شــده در ســطح روش بهتر از سطح رده عمل می‌کننــد. تغییــرات مقادیر APFD در ســطح روش در مقایســه با سطح رده کمتر اســت که دلالت بر این دارد که عملکرد فنون مطالعه شــده در سطح روش پایدارتر است. این یافته به پژوهشــگران پیشنهاد می‌دهد که از ریزدانگــی ســطح روش یا حتی ریزدانگی‌های ســطح پایین‌تری استفاده کنند.

• فنون مطالعه شده خطاهای غیرمشابهی را برای بالاترین آزمایه‌های اولویت‌دهی شده، کشف می‌کنند. به‌طور خاص، در سطح روش این فنون تنها در ۳۰٪ از خطاهای تشخیص داده شده برای ۱۰٪ آزمایه‌های اولویت‌دهی شده، مشترک هستند. این مشــاهده برای کارهای آینده چندین پیشنهاد می‌دهــد: الف) می‌توان برای تشــخیص خطاهای خاص و ســخت، به دنبال فنون اولویت‌دهی خاصی بود؛ ب) برای هدف قــرار دادن خطاهای خــاص در مکان‌های خاص از برنامه، فنون مختلف اولویت‌دهی می‌توانند مورد اســتفاده قرار گیرند؛ پ) برای دســتیابی به کارایی بالاتر، اطلاعات پویا و ایستا باید با یکدیگر ترکیب شوند؛ ت) APFD تصویر واضحی از کارایی نســبی فنون مختلف را بیان می‌کند؛ در حالی‌کــه تفاوت مجموعه‌ای بین خطاهای تشــخیص داده شده توسط فنون مختلف را نمی‌تواند به‌طور واضح نشان دهــد. در نتیجه این تفاوت مجموعــه‌ای می‌تواند به عنوان معیار جدیدی برای تحقیقات آینده در نظر گرفته شود.

• در سطح روش، فن مبتنی بر گراف فراخوانی از کارایی خیلی بیشتری برخوردار است. فنون مبتنی بر فاصلهٔ رشته و عنوان، وقتی که تعداد آزمایه‌ها افزایش می‌یابد، به زمان بیشتری احتیاج دارند.

معیارها:	C1:3،	C2:6،	C3:2،	C4:2،	C5:3،	C6:1,5,9,10،	C7:3،	C8:1،	C9:2،	C10:1،	C11:2،	C12:2.

هنارد[91] و همکاران [۲۶]، فنون اولویت‌دهی جعبه سیاه را با جعبه سفید مقایسه کرده‌اند. نویسندگان بیان کرده‌اند که با وجودی که فنون اولویت‌دهی جعبه سفید طی دو دههٔ

89-Luo
90-String-Distance-Based
91-Henard

جدول ۲: فنون اولویت‌دهی جعبه سفید و جعبه سیاه [۲۶]

فن	شماره	نام اختصار	نام کامل	هدف اولویتدهی
جعبه سفید	۱	TS	Total Statement[28][29][30]	پوشش حداکثر تعداد دستورالعمل‌ها
	۲	AS	Additional Statement[28][29][30]	پوشش حداکثر تعداد دستورالعمل‌های پوشش داده نشده
	۳	TB	Total Branch[28][29][30]	پوشش حداکثر تعداد شرط‌ها
	۴	AB	Additional Branch[28][29][30]	پوشش حداکثر تعداد شرط‌های پوشش داده نشده
	۵	TM	Total Method[28][29][30]	پوشش حداکثر تعداد روش‌ها
	۶	AM	Additional Method[28][29][30]	پوشش حداکثر تعداد روش‌های پوشش داده نشده
	۷	ASS	Additional Spanning Statements[31]	پوشش حداکثر تعداد دستورالعمل‌های غلبه کننده پوشش داده نشده
	۸	ASB	Additional Spanning Branches[31]	پوشش حداکثر تعداد شرط‌های غلبه کننده پوشش داده نشده
	۹	SD	Statement Diversity[32][33]	فاصلهٔ ژاکارد بین دستورالعمل‌ها بیشینه شود
	۱۰	BD	Branch Diversity[32][33]	فاصلهٔ ژاکارد بین شرط‌ها بیشینه شود
جعبه سیاه	۱	t-W	t-wise[34][35][36]	حداکثر تعاملات بین t تا ورودی مدل پوشش داده شود
	۲	IMD	Input Model Diversity[37][38]	فاصله ژاکارد بین ورودی‌های مدل بیشینه شود
	۳	TIMM	Total Input Model Mutation[38][39]	تعداد مدل‌های جهش یافته کشته شده بیشینه شود
	۴	AIMM	Additional Input Model Mutation[39] [38]	تعداد مدل‌های جهش یافته اخیراً کشته شده بیشینه شود
	۵	MiOD	Min. Output Diversity[40]	فاصله NCD بین خروجی‌ها کمینه شود
	۶	MaOD	Max. Output Diversity[40]	فاصله NCD بین خروجی‌ها بیشینه شود
	۷	ID-NCD	Input Diversity w/NCD[40]	فاصله NCD بین ورودی‌ها بیشینه شود
	۸	ID-Lev	Input Diversity w/Levenshtein[41][42]	فاصله لونشتاین بین ورودی‌ها بیشینه شود
	۹	I-TSD	Input Test Set Diameter[43]	فاصله NCD بین مجموعه چندتایی از ورودی‌ها بیشینه شود
	۱۰	O-TSD	Output Test Set Diameter[43]	فاصله NCD بین مجموعه چندتایی از خروجی‌ها بیشینه شود

گذشته، به‌طور زیادی مورد مطالعه قرار گرفته‌اند، فنون جعبه سیاه کمتر مورد مطالعه قرار گرفته‌اند. همچنین فنون جعبه سیاه نه تنها با فنون جعبه سفید مقایسه نشده‌اند، بلکه بین خودشان نیز مورد مقایسه قرار نگرفته‌اند. از این رو مشخص نیست که فنون جعبه سفید در مقایسه با فنون جعبه سیاه چگونه عمل می‌کنند. مزیت فنون جعبه سیاه این است که نیازی به کد منبع ندارند و در نتیجه نیازی به دسترس‌پذیری کد منبع و فراکدگذاری نیست. در طرف مقابل، ممکن است این فرضیه وجود داشته باشد که دسترسی به اطلاعات کد، به فنون جعبه سفید این اجازه را می‌دهد تا پوشش کد را افزایش دهند و در نتیجه قابلیت تشخیص خطا افزایش می‌یابد. همچنین ادعا شده است [۲۷] که فنون جعبه سفید پرهزینه هستند و استفاده از اطلاعات پوشش از نسخه‌های قبلی، باعث کاهش کارایی فنون اولویت‌دهی در چندین نسخه می‌گردد. این فرضیه‌ها باعث شده است تا برای مقایسهٔ کامل فنون اولویت‌دهی، این تحقیق انجام گیرد. برای این کار ۲۰ فن اولویت‌دهی شامل ۱۰ فن اولویت‌دهی جعبه سفید با ۱۰ فن اولویت‌دهی جعبه سیاه با یکدیگر مقایسه شده‌اند. این فنون در جدول ۲ خلاصه شده‌اند.

نتایجی که آزمایش‌ها بر روی این ۲۰ فن انجام داده‌اند بدین شرح است:

- بهترین فنون جعبه سفید، ASS، AB و ASB می‌باشند که بهترین نرخ تشخیص خطا را نشان داده‌اند.

• بهتریــن فنــون جعبــه ســیاه، t-W ، I-TSD و IMD می‌باشند.

• وقتی این شش فن با هم مقایسه شدند، مشاهده شد که تفاوت APFD بین فنون جعبه سفید و جعبه سیاه، بین ۲٪ تا ۴٪ می‌باشد و فنون جعبه سفید در ۵۶٪ تا ۶۰٪ موارد بهتر از فنون جعبه ســیاه عمل می‌کننــد. از بین این فنون، ASB بهترین نتیجه را داشته است. این نتایج بیانگر این است که فنون جعبه ســیاه به‌طور چشــمگیری با فنون جعبه سفید قابل رقابت هســتند، با وجودی کــه در این فنون اطلاعات ساختاری در دسترس نمی‌باشد.

• بعد از اجرای ۱۰٪ اول آزمایه‌ها بر روی فنون مختلف، مشــاهده شد که فنون جعبه ســفید و جعبه سیاه در ۶۰٪ خطاهای یافت شــده اشــتراک دارند. در نتیجه فنون جعبه ســفید و جعبه ســیاه مجموعه‌های یکســانی از خطاها را کشف می‌کنند.

• فنون جعبه ســیاه به‌طور کلی نســبت به فنون جعبه سفید، به زمان بیشتری برای اولویت‌دهی مجموعه آزمون نیاز دارند.

معیارها:	C1:3،	C2:6،	C3:1،	C4:1 ،	C5:1،	C6:1,2,5,10،	C7:1,4،	C8:1،	C9:2،	C10:2،	C11:2،	C12:1،

٤-٤ فنون افزایش مجموعه آزمون

در مقــالات [۴۴] و [۴۵] فنونــی ارائه شــده اســت که تمرکزشــان بــر روی فنــون تولیــد آزمایه اســت که از آزمایه‌های موجود استفاده می‌کنند. این دو مقاله به ترتیب از الگوریتم‌هــای کانکولیــک[92] و ژنتیک اســتفاده می‌کنند. مطالعات تجربی نشــان داده اســت که هر کدام از این دو می‌تواند در پوشش کدی که جدید اضافه شده است، مؤثر باشد.

معیارها:	C1:4،	C2:7،	C3:2،	C4:2،	C5:3،	C6:1،	C7:1،	C10:1،	C11:2،	C12:1،

در مقالهٔ [۴۶]، فنون افزایش با در نظر گرفتن عواملی که بر روی آن‌ها تأثیر می‌گذارد، بررســی شده‌اند. این عوامل عبارت‌اند از: الگوریتمی که برای تولید آزمایه‌ها اســتفاده شده اســت، ترتیبی که هدف‌ها در حین تولید آزمایه‌ها در نظر گرفته شــده‌اند و روشــی که آزمایه‌ها استفادهٔ مجدد می‌شــوند. نتایج این تحقیق نشــان داد که عامل اصلی که افزایش را تحــت تأثیر قرار می‌دهد، الگوریتمی اســت که برای تولید آزمایه‌ها اســتفاده می‌شود. این مقاله پیشنهاد کرده اســت که فنون افزایش ترکیبــی[93] می‌توانند به‌طور مؤثر این الگوریتم‌ها را ترکیب کنند تا روش‌هایی تولید کنند که مقرون‌به‌صرفه‌تر از هر الگوریتمی باشــد که به تنهایی استفاده می‌شود.

معیارها:	C1:4،	C2:7،	C3:1،	C4:1،	C5:1،	C6:1،	C7:1،	C10:2،	C11:2،	C12:2،

پیشنهاد اخیر در مقالهٔ دیگری [۴۷] بررسی شده است. این مقالــه یک روش افزایش ترکیبی ارائه کرده اســت که ابتــدا الگوریتم تولید آزمایه کانکولیــک را بر روی برنامه اعمال می‌کند و ســپس الگوریتم ژنتیــک را اعمال می‌کند. به این روش درهم‌گذاری ثابت[94] گفته می‌شــود. طبق نتایج تحقیق، این روش از روش غیرترکیبی کارآمدتر است، اما از لحاظ بهره‌وری ناامیدکننده بوده اســت. عیب دیگر این روش این است که فرض می‌کند مهندسان فنون افزایش را تا کامل‌شدن ادامه می‌دهند؛ یعنی تا زمانی پوشش را تقویت می‌کنند که هیچ راهی برای پوشــش هدف‌ها وجود نداشته باشــد. وقتی که زمــان، اجازهٔ چنین آزمونــی را می‌دهد، این فرض منطقی اســت. اما آزمون پس‌نمایی مکرراً تحت محدودیت‌های زمانی انجام می‌شــود. در این موارد هدف، پیداکردن فنون افزایشی نیست که در پایان حلقهٔ اجرایش، بهترین پوشش از عناصر جدیداً اضافه شده را تولید کند؛ بلکه هدف این است که فن افزایشی را پیدا کنیم که می‌تواند به پوشش، سریع‌تر از بقیه برسد.

معیارها:	C1:4،	C2:7،	C3:1،	C4:1،	C5:1،	C6:1،	C7:1،	C10:2،	C11:2،	C12:2،

مقالهٔ دیگری [۷] هر دو مســئلهٔ فوق را بررســی کرده اســت. این مقاله چارچوبی برای روش ترکیبی ارائه کرده اســت که الگوریتم‌های تولید آزمایه را بــه صورت پویا، درهم‌گــذاری می‌کند. ایــن چارچوب می‌تواند بر اســاس الگوریتم‌هــای تولیــد آزمایه و ترتیب هــدف و روش‌های

92- Concolic

93- Hybrid

94-Fixed interleaving

استفاده مجدد از آزمایه، پارامتری شود. در این چارچوب یک کنترل‌کننده وجود دارد که به‌طور پویا در مورد این‌که در هر زمانی چه الگوریتمی اســتفاده شود، تصمیم‌گیری می‌کند. در نتیجه در طی زمان از قدرت هر الگوریتم تولید آزمایه اســتفاده می‌شــود. این روش اکثر اوقات نسبت به روش درهم‌گذاری ثابت به پوشش بیشتری درزمان کمتر دست می‌یابد.

معیارها:	C1:4،	C2:7،	C3:1،	C4:1،	C5:1،	C6:1،	C7:1،	C10:2،	C11:2،	C12:2،

۵- نتیجه‌گیری

این مقالــه مهم‌ترین تحقیقــات و یافته‌های آزمون پس‌نمایی را گزارش کرده است. برای نیل به این هدف، مقالات مروری آزمون پس‌نمایی بررســی شــده‌اند و کاستی‌های آن‌ها شناسایی شده است. سپس با روشی نظام‌یافته مقالاتی از مهم‌ترین کنفرانس‌ها و مجلات در دوره شــش ســاله ۲۰۱۰ تا ۲۰۱۶ شناسایی و پالایش شــدند. در مرحله بعد ضمن تبییــن مفاهیم ضروری، معیارهایی معرفی شــده‌اند که امــکان ارزیابی علمی، غیرسلیقه‌ای و تکرارپذیر برای فنون آزمون پس‌نمایی میســر گردد. با فراهم شــدن چارچوب لازم، در بخش اعظم مقاله خلاصه مقالات یافت شده تشریح شده، نتایج مطالعه‌های هر یک گزارش شــده و سپس با معیارهای معرفی شده ارزیابی شــده‌اند. به‌طور خلاصه می‌توان گفت هم اکنــون در دوران گذار فنون آزمون پس‌نمایی از ســطح تحقیقات نظری به ســمت کاربردهای واقعی صنعتی به‌ســر می‌بریم. بنابراین نیاز تحقیقاتی امروز مطالعه‌هایی اســت که بفهمد هر روش در چه کاربردی موفق‌تر عمل می‌کند و عوامل زمینه‌ای مؤثر در موفقیت آن کدام است. همچنین نیاز به مخازن کدهایی داریم که از پروژه‌های واقعی جمع‌آوری شــده باشند که همگان روش‌هــای خود را با آن‌هــا ارزیابــی نمایند. حاصل این‌کار تعمیم‌پذیری بیشــتر نتیجــه مطالعه‌ها و امکان مقایسه معنی‌دار است.

مراجع

[1]H. Do, “Recent Advances in Regression Testing Techniques,” in Advances in Computers, Elsevier, 2016, pp. 1–25.

[2]S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey,” Softw. Testing, Verif. Reliab., vol. 22, no. 2, pp. 67–120, 2012.

[3]S. Wang, S. Ali, and A. Gotlieb, “Minimizing test suites in software product lines using weight-based genetic algorithms,” Proceeding fifteenth Annu. Conf. Genet. Evol. Comput. Conf., pp. 1493–1500, 2013.

[4]G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, “Empirical studies of test suite reduction,” Softw. Testing, Verif. Reliab., vol. 12, no. 4, pp. 219–249, 2002.

[5]G. Rothermel and M. J. Harrold, “Analyzing regression test selection techniques,” IEEE Trans. Softw. Eng., vol. 22, no. 8, pp. 529–551, 1996.

[6]D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, “To Be Optimal Or Not in Test-Case Prioritization,” IEEE Trans. Softw. Eng., vol. 6, no. 1, pp. 1–20, 2015.

[7]Y. Kim, Z. Zu, M. Kim, M. B. Cohen, and G. Rothermel, “Hybrid Directed Test Suite Augmentation: An Interleaving Framework,” in 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, 2014, pp. 263–272.

[8]D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, “On-demand test suite reduction,” in 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 738–748.

[9]G. Kumar and P. K. Bhatia, “Software testing optimization through test suite reduction using fuzzy clustering,” CSI Trans. ICT, vol. 1, no. 3, pp. 253–260, 2013.

[10]A. Gotlieb and D. Marijan, “FLOWER: optimal test suite reduction as a network maximum flow,” in Proceedings of the 2014 International Symposium on Software Testing and Analysis, 2014, pp. 171–180.

[11]A. Gotlieb, M. Carlsson, M. Liaaen, D. Marijan, and A. Pétillon, “Automated Regression Testing Using Constraint Programming,” 2016.

[12]H. Hemmati and L. Briand, “An industrial investigation of similarity measures for model-based test case selection,” in 2010 IEEE 21st International Symposium on Software Reliability Engineering, 2010, pp. 141–150.

[13]S. Mirarab, S. A. Esfahani, and L. Tahvildari, “Size-Constrained Regression Test Case Selection Using Multi-Criteria Optimization,” IEEE Trans. Softw. Eng., vol. 99, no. 1, pp. 936–956, 2012.

[14]M. Gligoric, L. Eloussi, and D. Marinov, “Practical regression test selection with dynamic file dependencies,” Proc. 2015 Int. Symp. Softw. Test. Anal. - ISSTA 2015, vol. 520, pp. 211–222, 2015.

[15]O. Legunsen, F. Hariri, A. Shi, Y. Lu, L. Zhang, and D. Marinov, “An extensive study of static regression test selection in modern software evolution,” in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 583–594.

[16]M. J. Arafeen and H. Do, “Test Case Prioritization Using

erage testing," IEEE Trans. Softw. Eng., vol. 29, no. 11, pp. 974–984, 2003.
[32]Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in 2013 13th International Conference on Quality Software, 2013, pp. 153–162.
[33]Z. Q. Zhou, A. Sinaga, and W. Susilo, "On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites," in System Science (HICSS), 2012 45th Hawaii International Conference on, 2012, pp. 5584–5593.
[34]R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," Inf. Softw. Technol., vol. 48, no. 10, pp. 960–970, 2006.
[35]R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting, 2007, pp. 1–7.
[36]J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 26–36.
[37]C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," IEEE Trans. Softw. Eng., vol. 40, no. 7, pp. 650–670, 2014.
[38]C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon, "Assessing software product line testing via model-based mutation: An application to similarity testing," in Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on, 2013, pp. 188–197.
[39]M. Papadakis, C. Henard, and Y. Le Traon, "Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing," in 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, 2014, pp. 1–10.
[40]E. Rogstad, L. Briand, and R. Torkar, "Test case selection for black-box regression testing of database applications," Inf. Softw. Technol., vol. 55, no. 10, pp. 1781–1795, 2013.
[41]H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," ACM Trans. Softw. Eng. Methodol., vol. 22, no. 1, p. 6, 2013.
[42]Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," Autom. Softw. Eng., vol. 19, no. 1, pp. 65–95, 2012.
[43]R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," arXiv Prepr. arXiv1506.03482, 2015.
[44]Z. Xu and G. Rothermel, "Directed test suite augmentation," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, pp. 406–413, 2009.
[45]Z. Xu, M. B. Cohen, and G. Rothermel, "Factors Affect-Requirements-Based Clustering," in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, pp. 312–321.
[17]H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in 2005 International Symposium on Empirical Software Engineering, 2005., 2005, p. 10–pp.
[18]R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 382–391, 2011.
[19]S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," Empir. Softw. Eng., vol. 19, no. 1, pp. 182–212, 2014.
[20]J. H. Kwon, I. Y. Ko, G. Rothermel, and M. Staats, "Test case prioritization based on information retrieval concepts," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, vol. 1, pp. 19–26, 2014.
[21]R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "REPiR: An Information Retrieval based Approach for Regression Test Prioritization," in 37th International Conference on Software Engineering, Florence Italy, Mary, 2015.
[22]S. Panda, D. Munjal, and D. P. Mohapatra, "A Slice-Based Change Impact Analysis for Regression Test Case Prioritization of Object-Oriented Programs," Adv. Softw. Eng., vol. 2016, p. 1, 2016.
[23]S. Eghbali and L. Tahvildari, "Test Case Prioritization Using Lexicographical Ordering," IEEE Trans. Softw. Eng., vol. 42, no. 12, pp. 1178–1195, 2016.
[24]Y. Lu, Y. Lou, S. Cheng, L. Zhang, D. Hao, Y. Zhou, and L. Zhang, "How does regression test prioritization perform in real-world software evolution?," Proc. 38th Int. Conf. Softw. Eng. - ICSE '16, pp. 535–546, 2016.
[25]Q. Luo, K. Moran, and D. Poshyvanyk, "A large-scale empirical comparison of static and dynamic test case prioritization techniques," in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 559–570.
[26]C. Henard, M. Papadakis, M. Harman, and Y. Le Traon, "Comparing White-box and Black-box Test Prioritization," pp. 523–534, 2016.
[27]H. Mei, D. Hao, L. L. Zhang, L. L. Zhang, J. Zhou, and G. Rothermel, "A static approach to prioritizing JUnit test cases," IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1258–1275, Nov. 2012.
[28]S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," IEEE Trans. Softw. Eng., vol. 28, no. 2, pp. 159–182, 2002.
[29]S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," Softw. Qual. J., vol. 12, no. 3, pp. 185–210, 2004.
[30]G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," IEEE Trans. Softw. Eng., vol. 27, no. 10, pp. 929–948, 2001.
[31]M. Marré and A. Bertolino, "Using spanning sets for cov-

ing the Use of Genetic Algorithms in Test Suite Augmentation," Proc. 12th Annu. Conf. Genet. Evol. Comput., no. 1, pp. 1365–1372, 2010.
[46]Z. Xu, Y. Kim, M. Kim, G. Rothermel, and M. B. Cohen, "Directed Test Suite Augmentation: Techniques and Tradeoffs," FSE '10 Proc. eighteenth ACM SIGSOFT Int. Symp. Found. Softw. Eng., pp. 257–266, 2010.
[47]Z. Xu, Y. Kim, M. Kim, and G. Rothermel, "A hybrid directed test suite augmentation technique," Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE, pp. 150–159, 2011.
[48]Busjaeger, Benjamin, and Tao Xie. "Learning for test prioritization: an industrial case study." In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 975-980. ACM, 2016.
[49]Qiu, Dong, Bixin Li, Shunhui Ji, and Hareton Leung. "Regression testing of web service: a systematic mapping study." ACM Computing Surveys (CSUR)47, no. 2 (2015): 21.
[50]Catal, Cagatay, and Deepti Mishra. "Test case prioritization: a systematic mapping study." Software Quality Journal 21, no. 3 (2013): 445-478.
[51]Engström, Emelie, Per Runeson, and Mats Skoglund. "A systematic review on regression test selection techniques." Information and Software Technology 52, no. 1 (2010): 14-30.
[52]Sipser, Michael. Introduction to the Theory of Computation. 3rd Edition. Boston: Thomson Course Technology, 2013.
[53]Anderson, Jeffrey Ryan. "Understanding contextual factors in regression testing techniques." PhD diss., NORTH DAKOTA STATE UNIVERSITY, 2016.
[54]Ammann, Paul, and Jeff Offutt. Introduction to software testing. Cambridge University Press, 2016.
[55]Jeffrey, Dennis, and Neelam Gupta. "Improving fault detection capability by selectively retaining test cases during test suite reduction." IEEE Transactions on software Engineering 33, no. 2 (2007).
[56]Khalilian, Alireza, Mohammad Abdollahi Azgomi, and Yalda Fazlalizadeh. "An improved method for test case prioritization by incorporating historical test case data." Science of Computer Programming 78, no. 1 (2012): 93-116.
[57]Zhang, Lu, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. "Time-aware test-case prioritization using integer linear programming." In Proceedings of the eighteenth international symposium on Software testing and analysis, pp. 213-224. ACM, 2009.
[58]Jia, Yue, and Mark Harman. "An analysis and survey of the development of mutation testing." IEEE transactions on software engineering 37, no. 5 (2011): 649-678.
[59]Nethercote, Nicholas, and Julian Seward. "Valgrind: a framework for heavyweight dynamic binary instrumentation." In ACM Sigplan notices, vol. 42, no. 6, pp. 89-100. ACM, 2007.
[60]Emam, Seyedeh Sepideh, and James Miller. "Test case prioritization using extended digraphs." ACM Transactions on Software Engineering and Methodology (TOSEM) 25, no. 1 (2015): 6.
[61]Kung, David. Object-oriented Software Engineering: An Agile Unified Methodology. McGraw-Hill Higher Education, 2013.
[62]Do, Hyunsook, Siavash Mirarab, Ladan Tahvildari, and Gregg Rothermel. "The effects of time constraints on test case prioritization: A series of controlled experiments." IEEE Transactions on Software Engineering 36, no. 5 (2010): 593-617.