تاریخ دریافت مقاله: ۹۵/۱۲/۱۶
تاریخ پذیرش مقاله: ۹۶/۰۱/۲۷

تعمیر خودکار برنامه‌ها
بررسی ادبیات و مرور نظام‌یافته فنون شاخص

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

احمد براآنی دستجردی*
دانشیار، گروه مهندسی نرم افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان
پست الکترونیکی: ahmadb@eng.ui.ac.ir

بهمن زمانی
استادیار، گروه مهندسی نرم افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان
پست الکترونیکی: zamani@eng.ui.ac.ir

چکیده

تعمیر خودکار برنامه‌ها حوزه جدیدی است و مطالعه‌های گسترده‌ای لازم دارد. از طرفی مقاله‌های بسیار، جنبه‌هایی از آن را مورد بررسی قرار داده‌اند و فنون متعددی نیز طراحی شده است. شروع پژوهش در این حوزه نیاز به مرور مقاله‌های بسیار و آشنایی با مفاهیم متعدد دارد. گام بعدی، ضرورت بررسی کارهای شاخص است تا محقق از دانش فعلی در این حوزه آگاهی یابد. مقاله حاضر نیازهای مطرح شده را برآورده می‌سازد. برای این منظور، ادبیات تعمیر خودکار برنامه‌ها با ارائه تعریف‌ها، مفاهیم و دسته‌بندی‌های موجود معرفی می‌شود. سپس ۳۱ فن شاخص از کارهای موجود انتخاب شده و بررسی می‌شوند تا سیر تحقیقی و یافته‌ها روشن شود. برای کسب شناخت بهتر، روشی جهت ارزیابی فنون نیز ارائه شده است. با تکیه بر مرور ادبیات، درس‌هایی از روش‌های پیشین بیان می‌شوند که مبنایی برای برشمردن مشکلات موجود و نیاز تحقیقی آینده شوند. ارزش مقاله در اینست که شروع تحقیق در تعمیر خودکار را تسهیل و تسریع می‌کند و کمک می‌کند محقق مسیر درست و مورد نیاز در کارهای آینده را پیدا کند.

واژه‌های کلیدی: اشکال‌زدایی نرم‌افزار، آزمون نرم‌افزار، خطا، تعمیر خودکار برنامه.

۱-مقدمه

براساس اطلاعات نیویورک تایمز، واژه نرم‌افزار در سال ۱۹۵۸ توسط آماردانی به‌نام جان ویلدر توکی در قالب مقاله‌ای در ماهنامهٔ ریاضی آمریکا ابداع گردید. از آن به بعد استفاده از نرم‌افزار به‌طور پیوسته در جامعه رواج پیدا کرد [۱] و در کاربردهای مختلف مورد استفاده قرار گرفت. روند استفاده از نرم‌افزار به‌شدت افزایش پیدا

* نویسندهٔ مسئول

کـــرد؛ طوری‌که امروزه تمام دنیا بـــرای انجام فعالیت‌های روزمره، وابســـتگی شـــدید به نرم‌افزار دارد. مســـئله این اســـت که نرم‌افزار، همیشه رفتار پیش‌بینی شده‌ای از خود نشـــان نمی‌دهد و رفتـــار مطمئنی ندارد. عـــدم اطمینان به نرم‌افزارها و عدم کارکرد مطمئن آن‌ها منجر به هزینه‌های مالی و انسانی ســـنگینی می‌گردد [۱]. یکی از دلایل اصلی عدم اطمینان به نرم‌افزارها وجود خطا در کد است. خطاها به دلیل پیچیدگی تولید نرم‌افزار و دســـتی بودن آن بروز می‌کنند و بســـیار رایجند [۲]. اتـــکای فعالیت‌های روزمره به نرم‌افزار و رواج خطاها مســـئله خطاهای نرم‌افزاری را شدیداً جدی می‌ســـازد [۱]. آمارهایی از نمونه‌های واقعی اهمیت خطاهای نرم‌افزاری را بهتر نشان می‌دهد:

نرم‌افزار حراجی شـــرکت صنایع داو جونز[1] و شرکت معروف پروکتِر و گَمبِل[2] ده‌ها ســـال‌ها بدون مشـــکل کار کرده‌اند. اما یک خطای تایپی باعث شـــده که در مِی ۲۰۱۰ ضرری معادل به‌ترتیب ۱۰٪ و ۳۷٪ در ســـهام دو شـــرکت پدید آید. این شـــواهد ضـــرورت آزمون دائمـــی نرم‌افزار را آشـــکار می‌ســـازند [۳]. خطاها پرهزینه‌اند: فعالیت‌های آزمون، اشکال‌زدایی و درســـتی‌آزمایی حدود ۵۰٪ تا ۷۵٪ کل هزینه‌های تولید و توسعه را به‌خود اختصاص می‌دهد [۴]. مطالعه‌ای در ســـال ۲۰۱۳ در دانشـــگاه کمبریج هزینه کلی اشکال‌زدایی را در دنیا حدود ۳۱۲ میلیارد دلار تخمین زده اســـت. این مطالعه نشان داده اســـت که برنامه‌سازان حدود ۵۰٪ از زمان خود را صرف رفع اشکال‌ها می‌کنند و صرف این‌که کد نوشته شده کار کند و رفتار درستی داشته باشد [۵]. اهمیت این هزینه وقتی خوب آشکار می‌شود که بدانیم کمک مالی اتحادیه اروپا در خلال سال‌های ۲۰۰۸ تا ۲۰۱۳ به چهار کشور ایرلند، پرتغال، یونان و اسپانیا حدود ۵۱۹ میلیارد دلار بوده اســـت. نتایـــج مطالعه‌ای [۶] از رفع خطاهای نرم‌افزار توسط برنامه‌سازان انسانی نشان داده است که از بین ۲۰۰۰ مورد خطاهای رفع شده، حدود ۱۴٪ تا ۲۴٪ نادرست بوده‌اند؛ و ۴۳٪ آن‌ها موجب از کارافتادگی، تخریب، توقف عملکرد یا گیرگردن[3] و مشـــکلات امنیتی در نرم‌افزار شده‌اند. به‌عبارت ساده رفع دستی خطا ممکن است باعث بروز اشکال‌های بیشتر در کد گردد [۷]. شرکت‌های بسیار و معروفی وجود دارند همچون موزیلا[4] و گوگل[5] که طرحـــی را تحت عنوان «پـــاداش در ازای خطا[6]» راه‌اندازی کرده‌اند [۸]. در این طرح به افراد و برنامه‌سازانی که بتوانند خطاهای نرم افزاری، ویژه مشکلات امنیتی را کشف کنند و کد تعمیر آن را بسازند، پاداش‌های قابل توجهی در حد چند هزار دلار داده می‌شود.

متأســـفانه نرم‌افزارها هنوز با نقص‌های شناخته شده و شناخته نشـــده عرضه می‌شـــوند [۹] چون: الف) تعداد خطاهای نرم‌افزاری که از طریق آزمون کشـــف می‌شوند بسیار بیشـــتر از تعداد خطاهایی هستند که رفع می‌شوند [۱۰]؛ ب) یافتن و رفع خطاهای نرم‌افزاری به‌صورت دستی توسط برنامه‌ســـازان صورت می‌گیرد و بسیار زمان‌گیر اســـت و این موضوع باعث می‌شود اشکال‌های نرم‌افزاری و مســـایل مربوط به آن‌ها پرهزینه شـــوند [۱۱]؛ پ) زمان و منابـــع مختلف آزمون و اشـــکال‌زدایی محدودیت دارند [۱۲]؛ ت) برنامه‌ســـازی فرایندی بســـیار پیچیده و انسانی اســـت و همین موضوع آنرا خطاپرور[7] می‌سازد. به‌همین دلیل شـــرکت‌های نرم‌افزاری برای حفظ موقعیت خود در بازار، نرم‌افزار را با خطاهایی که با آزمون کشـــف شده‌اند به بازار عرضه می‌کنند و متعهّد می‌شوند در مدت کوتاهی خطاها را رفع کنند. برای نمونه سیستم عامل ویندوز ۲۰۰۰ با ۶۳،۰۰۰ خطاهای شناخته شده به بازار عرضه شد [۱۳] کـــه علت آن ناکافـــی بودن منابع آزمون بـــود. تعمیر خطا معمولاً از طریق عرضه وصله‌هایی[8] برای نرم‌افزار صورت می‌گیرد که خطاهای مزبور را رفع[9] می‌نماید.

یافتن اشـــکال‌های برنامـــه می‌تواند پیـــش از عرضه

1- DowJones
2- Procter & Gamble

3- Hang
4- http://www.mozilla.org/security/bug-bounty.html $3,000/bug
5- http://blog.chromium.org/2010/01/encouraging-more-chromium-security.html $500/bug
6- Bug bounty
7- Fault-prone
8- Patch
9- Bug-fix, repair, correction, or patching

نرم‌افزار[10] یا بعد از اســتقرار[11] آن انجام شــود. قطعاً همه تولیدکنندگان نرم‌افزار می‌خواهند همه اشــکال‌ها و خطاها پیش از عرضه کشف و رفع شوند. به‌دلایلی که مطرح شد، این کار حداقل فعلاً قابل تحقق نیست. متأسفانه رفع خطاها بعد از عرضه نرم‌افزار گران است و مشکلاتی جدی دارد؛ گاهی بازتولید[12] آن‌ها مشــکل اســت [۱۴] چون اطلاعات گزارش‌های اشــکال ناکافی اســت و جمع‌آوری اطلاعات دقیق، گران و پرهزینه اســت؛ ممکن اســت موجب شــود داده‌های حساس و خصوصی کاربران حین اشکال‌زدایی فاش شود [۱۵, ۱۴]؛ و می‌تواند باعث شود که اعتبار شرکت نرم‌افــزاری در بازار رقابتی به‌خطر بیفتــد [۱۵]. به‌خاطر هزینه بــالای عیوب نرم‌افزار پس از اســتقرار، محققان و تولیدکننــدگان نرم‌افزار تمرکز خود را بر خودکارســازی یافتن و اشکال‌زدایی در حین توسعه گذاشته‌اند.

این مقاله بر آخرین قســمت چرخه آزمون و اشــکال زدایــی تمرکز می‌کند، یعنی تعمیر خطاها؛ و مناســب نیاز تحقیقــی و صنعتــی امــروز خودکارســازی آن‌را مورد بررســی قرار می‌دهد. تعمیر خــودکار برنامه‌ها از یافتن اشکال‌ها دشوارتر است [۱۶]. بنابر گفته توماس زیمِرمان[13] از مرکــز تحقیقات مایکروســافت، باید شــرایطی را پیدا کنیم که تعمیر خودکار در آن‌ها مفید اســت [۱۷]: «یکی از چالش‌ها شناســایی موقعیت‌های زمانی و مکانی است که تعمیر خودکار را می‌توان به کار گرفت. من انتظار ندارم که تعمیر برنامه‌ها برای همــه خطاها در دنیا کار کند (در این صورت هزاران برنامه‌ســاز بیکار خواهند شد)، اما اگر از قبل حوزه‌هایی که در آن‌ها کار می‌کند را بشناسیم، توانایی زیادی به دست می‌آوریم [۱۸, slide ۶۷].[14]»

ابداع روش‌هــای اولیه برای تعمیر خــودکار از حدود سال ۲۰۰۹ شروع شده است و تاکنون فنون متعددی برای آن ابداع شــده است. با این حال فنون تعمیر خودکار هنوز در مراحل ابتدایی خود به ســر می‌برند و یافته‌ها هنوز در مقیاس پژوهشی قابل اســتفاده اند. مسایل بسیاری هنوز ناشناخته هســتند و مطالعه‌های گسترده روی نمونه‌های واقعی مورد نیازند. حتی بعد از طراحی فنون مؤثرتر، نیاز به انتقال دانش از مقیاس پژوهشی به کاربردهای صنعتی مطرح می‌شــود. این موضوع نیازمند شناخت عوامل مؤثر در موفقیت روش و شــرایطی است که هر فن در آن بهتر کار می‌کند.

برای تحقیق در حوزه تعمیر خودکار باید ســاختار فنون موجود و عملکرد آن‌ها را بشناسیم و نقاط قوت و ضعف آن‌ها را شناســایی نماییم. نقاط قوت درس‌هایی هســتند که برای طراحی روش‌هــای بعدی باید در نظر گرفته شــوند و نقاط ضعف مشکلاتی هستند که باید از آن‌ها اجتناب شــوند. همچنین با بررسی فنون موجود معلوم می‌شود تاکنون چه دستاوردهایی حاصل شده و نیاز آینده چیست و روند تحقیقی باید به چه سمتی ادامه پیدا کند. مونپِروس[15] با ارائه چند معیار [۴۴] سعی کرده اســت روشــی برای ارزیابی فنون تعمیر خودکار ارائه نماید. متأســفانه او اصلاً فنون موجود را مرور نکرده است و معیارهای ارائه شــده نیز محدودند. لِگوس[16] و همکاران هم در ۲۰۱۳ بدون مرور ادبیات فقط چالش‌های تعمیر خودکار را بررسی کرده اند [۱۹]. بنابراین مطالعه مروری از فنــون تعمیر خودکار همــراه با معیارهایی برای ارزیابی و مقایســه آن‌ها و شناسایی چالش‌های موجود ضروری به نظر می‌رســد. در راســتای تأمین این نیاز، مقاله حاضر فنون شــاخص تعمیر خودکار را مرور می‌کند و با استفاده از معیارهایی آن‌ها را ارزیابی می‌نماید. تا جایی که نویســندگان اطلاع دارند، این مقاله اولین گزارشی است که همزمان فنون تعمیر خودکار را به طور نظام یافته مرور و با معیارهایی ارزیابی می‌کند و چالش‌های تحقیقاتی را شناسایی می‌نماید. مرور نظام یافته ادبیات از روش‌شناســی خاصــی پیروی می‌کند که حاصل بررســی‌ها حتی الامکان بی‌طرفانه، جامع و

10- Before-release
11- Post-deployment
12- Reproduce
13- Thomas Zimmermann
14- http://www.cs.virginia.edu/_weimer/p/weimer-ssbse2013.pdf
15- Monperrus
16- Le Goues

فراگیر، تعمیم پذیر و تکرارپذیر باشد [۲۰]. نتایج حاصل از تحقیق حاضر از دو جنبه سودمند است:

• از جنبه نظری و پژوهشی برای افرادی که قصد شروع پژوهش در حوزه تعمیر خودکار دارند؛ این مقاله سیر پیشرفت در حوزه را تا زمان فعلی برای آن‌ها روشن می‌سازد و راهنمایی برای تحقیقات آتی فراهم می‌نماید.

• از جنبه تجاری و صنعتی برای افرادی که قصد دارند تجربه ای از به کارگیری فنون موجود در مقیاس‌های از کاربردهای عملی خود داشته باشند؛ به آن‌ها کمک می‌کند فن مناسب را برای کاربرد خویش شناسایی نمایند.

به‌طور خلاصه نوآوری‌های این مقاله عبارتند از:

• جمع‌آوری و ارائه تعاریف و دسته‌بندی‌های حوزه تعمیر خودکار برنامه‌ها

• مرور نظام‌یافته‌ای روی ۳۱ فن شاخص تعمیر خودکار

• ارزیابی فنون مرور شده با شش معیار از چارچوب قبلی نویسندگان

• ارائه درس‌هایی از مطالعه‌های گذشته، چالش‌های موجود و سیر تحقیقی آینده

ساختار ادامه مقاله به‌شرح زیر است: بخش دوم مقاله واژه‌ها و مفاهیم لازم را تعریف می‌کند که برای درک عملکرد فنون موجود لازم هستند. بخش سوم مقاله دسته‌بندی‌هایی ارائه می‌دهد که برای ارزیابی فنون موجود ضروری هستند. بخش چهارم ضمن تشریح روش انتخاب مقاله‌های مورد بررسی، اوزان ارزیابی را نیز معرفی می‌نماید. خلاصه‌ای از فنون انتخابی در بخش پنجم مقاله ارائه می‌شوند. بخش ششم درس‌هایی از فنون گذشته، مشکلات فنون موجود و راهنمای تحقیقات آینده را خلاصه می‌سازد. بخش هفتم نیز مقاله را جمع بندی می‌کند.

۲- تعاریف و مفاهیم ضروری

برای درک فنون تعمیر خودکار دانستن مفاهیمی ضروری است. این بخش تعریف هر یک از این مفاهیم را ارائه می‌کند تا ضمن آشنایی، هر گونه ابهام نیز برطرف گردد.

تعریف ۱: یک نقص ایستا داخل نرم افزار اشکال[17] نامیده می‌شود [۲۱]. این نقص در اثر اشتباه طراحی توسط انسان رخ می‌دهد و منجر به رفتار ناخواسته می‌گردد.

تعریف ۲: حالت داخلی نادرست در نرم افزار که اشکال (ها) را آشکار می‌سازد خطا[18] نامیده می‌شود [۲۱].

تعریف ۳: رفتار خارجی نادرست نسبت به نیازمندی‌ها یا سایر توصیف‌های رفتار مورد انتظار، خرابی[19] نامیده می‌شود [۲۱].

تعریف ۴: فرایندی که در آن نرم‌افزار تحلیل می‌شود تا اطلاعاتی راجع به درستی آن به‌دست آید را آزمون نرم‌افزار[20] می‌نامند. آزمون جایگشتی از دو نوع فعالیت آزمون است: تحریک و مشاهده که متناظر با دادن ورودی و دریافت خروجی هستند [۲۲].

تعریف ۵: فرایندی را که در آن تعیین می‌شود محصول یک مرحله از فرایند توسعه نرم افزار نیازمندی‌هایی که در مرحله قبل تأمین شده اند را برآورده می‌سازد درستی آزمایی[21] می‌نامند [۲۱]. همچنین اثبات عدم وجود نقایص در برنامه نسبت به توصیف برنامه را درستی آزمایی می‌نامند [۲]. برای درستی آزمایی معمولاً از روش‌های صوری استفاده می‌شود.

تعریف ۶: فنونی را که برای تحلیل نرم افزار یا سخت افزار از روش‌های ریاضی سخت گیر استفاده می‌کنند روش‌های صوری[22] می‌نامند. وارسی مدل، اثبات قضیه، منطق هوآر، مدل‌های مبتنی بر اصل موضوعی، معنای نشانه گذاری و تفسیر انتزاعی از جمله فنون به‌کار رفته در روش‌های صوری هستند.

تعریف ۷: فرایندی که یک برنامه و مجموعه ای از خصوصیات را دریافت می‌کند و بررسی می‌کند که آیا

17- Fault
18- Error
19- Failure
20- Software testing
21- Verification
22- Formal methods

خصوصیت‌ها در برنامه برقرار هستند یا این‌که برنامه خصوصیت‌ها را برآورده می‌سازد یا خیر وارسی مدل[۲۳] می‌نامند.

تعریف ۸: به ترکیب مقادیر آزمایه، نتایج مورد انتظار و مقادیر پیشوندی و پسوندی مورد نیاز برای اجرای کامل و ارزیابی برنامه تحت آزمون آزمایه[۲۴] (مورد آزمون) گفته می‌شود.

تعریف ۹: به مجموعه ای از آزمایه‌ها مجموعه آزمون[۲۵] اطلاق می‌شود.

تعریف ۱۰: گزاره ای را که تعیین می‌کند رشته ای از فعالیت‌های آزمون رفتار قابل پذیرشی از برنامه است یا خیر پیش گوی آزمون یا اوراکل[۲۶] می‌نامند [۲۲]. به بیان ساده، اوراکل خروجی مورد انتظار برنامه است. سازوکاری که خروجی مورد انتظار را برای آزمایه ای می‌سازد همراه با سازوکاری که خروجی واقعی را با مورد انتظار مقایسه می‌نماید، مقایسه گر اوراکل نامید می‌شود.

تعریف ۱۱: نیازمندی آزمون[۲۷] به عنصر ویژه ای از محصول یا دست ساخته نرم افزاری اطلاق می‌شود که آزمایه باید آن را برآورده سازد[۲۸] یا پوشش دهد[۲۹].

تعریف ۱۲: قانون یا مجموعه ای از قوانین را که نیازمندی‌های آزمون را بر هر آزمایه تحمیل می‌کند معیار پوشش[۳۰] می‌نامند. یک مجموعه آزمون را کافی در یک معیار آزمون می‌گویند اگر آزمایه‌های درون آن همه نیازمندی‌های آزمون از نظر آن معیار را پوشش دهند. به معیار مذکور، معیار کفایت آزمون[۳۱] اطلاق می‌شود.

تعریف ۱۳: فرایندی که در آن تعدادی گونه جهش‌یافته[۳۲] از یک برنامه ورودی ساخته می‌شود و نسبت جهش‌یافته‌هایی را که روی حداقل یک آزمایه اجرای ناموفق دارند محاسبه می‌کند، آزمون جهش[۳۳] نامیده می‌شود. این نسبت به امتیاز کفایت جهش معروف است و بالاتر بودن مقدار آن دلالت بر کیفیت مجموعه آزمون دارد. مقدار ایده‌آل این امتیاز یک است. کیفیت بالای مجموعه آزمون هم اطمینان‌پذیری بیشتر از برنامه تحت آزمون می‌دهد. گونه جهش‌یافته‌ای را که با یک جهش یا تغییر حاصل شده باشد مرتبه اول[۳۴] و بیشتر از یک را مرتبه‌های بالاتر[۳۵] می‌نامند.

تعریف ۱۴: هر فنی که بتوان اثبات کرد در حوزه همان فن مثبت کاذب یا منفی کاذب ندارد (جواب نادرست نمی‌دهد) استوار نامیده می‌شود. به‌عبارت دیگر الگوریتم یا فنی را استوار می‌گویند که همواره پاسخ صحیح یا خروجی درست برگرداند. سایر فنون با نهایت تلاش[۳۶] را نااستوار[۳۷] می‌نامند. در حوزه تعمیر خودکار، فنون با نهایت تلاش آن‌هایی هستند که ضمانت نمی‌کنند تعمیرهای نامزد واقعاً خطا را برطرف سازند (یا حتی برنامه‌های قابل اجرا باشند).

تعریف ۱۵: فن یا الگوریتمی را کامل[۳۸] می‌نامند اگر به ازای همه ورودی‌های ممکن جواب درست را در صورت وجود بیابد.

تعریف ۱۶: به کدی که خطای مورد نظر را ندارد، یعنی آن را رفع می‌نماید و اغلب جایگزین کد معیوب قبلی می‌گردد وصله[۳۹] اطلاق می‌شود. گاهی داخل برنامه دستوری گذاشته می‌شود که به‌جای اجرای کد معیوب، به کد جدید پرش کند. بعضی مقالات وصله را مجموعه‌ای از تغییرات می‌دانند که باید پشت سر هم روی برنامه صورت گیرد تا خطایش برطرف گردد. این دو تعریف در حقیقت به نوع خروجی فن تعمیر برمی‌گردد.

تعریف ۱۷: به فرایندی که در آن کد اضافه‌ای درون کد برنامه درج می‌شود که حین اجرای برنامه اطلاعاتی

23- Model checking
24- Test case
25- Test set (suite)
26- Test oracle
27- Test requirement
28- Satisfy
29- Cover
30- Coverage criteria
31- Test adequacy criteria
32- Mutant
33- Mutation testing
34- First-order
35- Higher-order
36- Best-effort
37- Unsound
38- Complete
39- Patch

را جمع‌آوری و ثبت و گـزارش نماید، فراکدگذاری[40] گفته می‌شـود. انـواع فنون مختلـف تحلیل کد بـرای آزمون، مکان‌یابی و تعمیر خطا به فراکدگذاری در پیش‌پردازش‌های خود نیاز دارند.

تعریـف ۱۸: نمایش درختی سـاختار دسـتورزبانی کد منبع را درخت خلاصه نحـوی[41] می‌نامند. هر گره این درخت متناظر با یک سـاختار زبان برنامه‌سـازی در کد اسـت. درخت خلاصه اسـت زیرا همه جزئیات واقعی در کد را نمایش نمی‌دهد؛ برای نمونه پرانتزها حذف می‌شوند زیر سـاختار دستوری زبان به‌طور ضمنی آن‌ها را نشان می‌دهنـد. درخت‌های خلاصه نحوی را می‌توان به‌شـکل کارامد ساخت و همه ساختارهای برنامه را بدون از دست دادن هیچ‌گونه اطلاعاتی نمایش می‌دهند.

تعریـف ۱۹: در حوزه برنامه سـازی ژنتیـک، تمایل الگوریتم به تولید کدهای میانی غیر ضروری را که سهمی در عملکرد راه حل نهایـی ندارند تورم کد[42] می‌گویند؛ راه حل‌ها بیش از حد لازم برای بیشینه سازی مقدار برازندگی کد دارند [۲۳].

تعریـف ۲۰: فنی که بـه طور خودکار سـعی می‌کند مجموعه‌هـای کوچکتـری از خطوط کد منبـع را بیابد که سـهم بیشتری در بروز خطایی داشـته باشند، مکان یابی اشکال[43] [۲۴] نامیده می‌شود. هدف از این کار کاهش تعداد دسـتوراتی اسـت که برای شناسـایی خود خطا و تعمیر مناسب باید بررسی شوند.

تعریف ۲۱: مهندسی نرم افزار اغلب با مسائلی روبرو می‌شود که در آن فضای بزرگی از نیازمندی‌ها، طراحی‌ها، آزمایه‌هـا و کدها وجود دارد. برای بهینه سـازی در این فضاها از فنون مهندسی نرم افزار مبتنی بر جست وجو[44] استفاده می‌شود که با تابع برازندگی که مطلوبیت راه حل را نشـان می‌دهد هدایت می‌گردد. این فنون قادرند اهداف بهینه‌سـازی چندگانه، متناقض و پارازیتـی را مدیریت و مهـار کنند که اغلب در سـناریوهای مهندسـی نرم افزار رایـج اسـت [۱۶]. موفقیت هر فن در مهندسـی نرم افزار مبتنی بر جسـت وجو به پیکربندی درست و به کارگیری الگوریتم‌هـای مناسـب بسـتگی دارد. حل مسـئله تعمیر خودکار با برنامه‌سـازی ژنتیک نمونه‌ای از مهندسی نرم افزار مبتنی بر جست وجو است.

تعریف ۲۲: مجموعه ای از داده‌ها شـامل حوزه‌های از پیش تعریف شـده، متن با قالب آزاد و ضمیمه‌های آن را گزارش اشـکال[45] می‌نامند که مورد اسـتفاده در مدیریت، رده بندی و تعیین اولویت[46] و تعمیر اشکال بخصوص در پایگاه داده ردیابی اشکال ها[47] هستند.

تعریف ۲۳: آزمونی که بررسی می‌کند رفتار نرم افزار در نسـخه قبلی، هم اکنون در نسخه جدید نیز مثل گذشته کار می‌کنـد را آزمون پس نمایـی[48] می‌نامند. آزمایه‌های مورد استفاده در این آزمون به پنج دسته تقسیم می‌شوند و آن‌هـا را آزمایه‌هـای پس نمایی می‌نامنـد. آزمون پس نمایی پر هزینه اسـت و یکی از روش‌های کاهش هزینه‌ها و افزایش سودمندی آن، اولویت دهی[49] آزمایه هاست. در این روش آزمایه‌ها بر اسـاس معیاری مرتب می‌شوند که آزمایه‌های خاصی زودتر از بقیه اجرا شوند.

تعریف ۲۴: نوعی از توصیف‌های رسمی به شکل پیش شرط، پس شرط و ثابت‌های رده‌ای[50] را قرارداد[51] می‌نامند.

۳- دسته‌بندی‌های مطرح

بسـیاری از مفاهیم مطرح شـده در بخش دوم انواعی دارنـد. فنون مختلفی را هم که در بخش چهارم مورد بحث قرار می‌گیرند از جنبه‌های مختلف می‌توان دسته بندی کرد. این بخش گونه‌های مختلف مفاهیم و دسته بندی‌های متنوع فنـون را ارائه می‌کند. این گونه‌ها و دسـته بندی‌ها هنگام بیان خصوصیت‌های فنون و ارزیابی آن‌ها لازم می‌شوند.

40- Instrumentation
41- Abstract syntax tree (AST)
42- Code bloat
43- Fault localization, Locating a bug
44- Search-based software engineering (SBSE)
45- Bug report
46- Triage
47- Bug-tracking database
48- Regression testing
49- Prioritization
50- Precondition, postcondition, and class invariant
51- Contract

ابتدا گونه‌های مفاهیم مطرح شـــده در بخش دوم به همان ترتیب ارائه می‌شود و سپس دســـته بندی‌های مربوط به فنون تعمیر خودکار بیان می‌گردد.

اشکال: اشکال ممکن است فعال یا غیر فعال[52] باشد؛ کد معیوبی که هرگز اجرا نمی شود، هرگز منجر به رخداد خطا هم نخواهد شد [۱۵]. اشکال‌ها می‌توانند قطعی یا غیرقطعی[53] و ترتیبـــی[54] یا هم روندی[55] باشـــند. رقابت داده‌ای[56]، نقض تجزیه ناپذیری[57] و نقض ترتیب[58] از انواع اشـــکال‌های هم روندی هستند. دو خاصیت مهم اشکال پیچیدگی و قابلیت کشف هستند [۲۵].

آزمون: آزمون ممکن است ایستا یا پویا باشد. آزمون برنامه بـــدون اجرای آن را آزمون ایســـتا[59] می‌نامند، مثل انواع مختلف تحلیل کد و بازرســـی کـــد. فعالیت‌های انجام شـــده برای درســـتی آزمایی برنامه‌ها در ادبیات معمولاً آزمون ایســـتا نامیده می‌شـــود. آزمون برنامه به وســـیله اجـــرای آن با ورودی‌های واقعـــی آزمون پویا یا مبتنی بر اجرا[60] نامیده می‌شود. اغلب در ادبیات، آزمون پویا را فقط آزمون می‌نامند.

آزمایه: آزمایـــه ای را که اجرایش روی برنامه معیوب خروجی درست داشته باشد و نشانگر رفتار درست برنامه اســـت و این رفتار بعد از تعمیر نیز باید حفظ شود آزمایه مثبت یا با اجـــرای موفق[61] می‌نامند. هـــر آزمایه ای را که اجرایش روی برنامه معیوب خروجی نادرست داشته باشد و نشـــانگر رفتار معیوبی باشد که باید تعمیر گردد آزمایه منفی یا با اجرای ناموفق[62] می‌نامند.

نیازمنـــدی آزمـــون: نیازمندی‌های نرم افـــزار معمولاً به دو دســـته عملکردی و غیر عملکردی تقسیم می‌شوند. دســـته اول مربوط به تولید خروجی درست و دسته دوم مربـــوط به کیفیت اجرای برنامه می‌شـــوند. دســـتورهای شرطی داخل برنامه، تابع و دستورهای تعریف و استفاده متغیرها نیازمندی‌های عملکردی هستند. زمان اجرا و زنده بودن برنامه از خصوصیت‌های غیر عملکردی هستند. در حوزه تعمیر خـــودکار نیازمندی‌های عملکردی مورد نظر قرار می‌گیرنـــد و اغلب زمان اجرایی تنها خصوصیت غیر عملکردی است که لحاظ می‌شود [۱۵].

وصله: اگـــر وصله تولیدی بـــه ازای همه ورودی‌های مجموعه آزمون ارزیابی خروجی درســـت تولید کند، آن‌را باورکردنی یا معقول[63] می‌گویند. اگر وصله تولیدی خطا را کاملاً برطرف سازد، آن را درست[64] می‌گویند [۲۶, ۲۷].

مدیریت و مهار اشکال‌های نرم افزاری: دو رویکرد کلی وجـــود دارد [۱]: یکی فنون جلوگیری[65] از رخداد خطاها یا فنون پیش از توســـعه[66] که هدف ساخت نرم افزاری است که حین ساخت کیفیت آن تضمین گردد. دسته دوم، فنون پاســـخ به اشکال‌ها هستند؛ یعنی پس از ایجاد اشکال آن را به نحوی مدیریت نماییم؛ آن‌ها را شناســـایی و رفع[67] کنیم؛ یا پیامدها و اثرات منفـــی آن‌ها را تحمل کرده[68] یا از آن‌ها اجتنـــاب[69] نماییـــم. این فنون معمولاً به خـــود نرم افزار و عناصر جانبی آن مثل توصیف‌ها اعمال می‌شـــوند و فنون پس از توسعه[70] نام دارند.

فنون جلوگیری: به دو دسته تقسیم می‌شوند؛ فنونی که به روشـــگان تولید نرم افزار اعمال می‌شوند که لازم است فرایند تولید نرم‌افزار تغییر پیدا کند مثل مهندسی نرم افزار سخت گیر[71] [۲۸]. فنونی که با روش‌های رسمی نرم افزار بدون خطا می‌سازند؛ مثل ساخت نرم افزار هدایت شده با اثبات نظری[72] یا طراحی با قراردادها [۱۵].

فنون پاســـخ: هر چند فنون مورد استفاده حین توسعه

52- Dormant
53- Deterministic or nondeterministic
54- Sequential
55- Concurrency
56- Data race
57- Atomicity violation
58- Order violation
59- Static testing
60- Dynamic (execution-based) testing
61- Passing
62- Failing
63- Plausible
64- Correct
65- Prevention
66- A priori
67- Fix
68- Tolerate
69- Avoid
70- A posteriori
71- Rigorous software engineering
72- Proof-guided construction

از بـــروز خطاها تا حدی زیاد جلوگیـــری کرده و اطمینان پذیـــری نرم افزار را افزایش می‌دهنـــد، اما حداقل در آینده نزدیـــک نمی توانند از بروز خطاها به طور کامل جلوگیری کنند؛ زیرا تولید نرم افزار هنوز توســـط انســـان و دستی صورت می‌گیرد که انسان هم اشتباه می‌کند. اشکال زدایی و درستی آزمایی راه کارهایی هستند که اعتمادپذیری نرم افزارهای موجود را در حد قابل ملاحظه ای افزایش می‌دهد.

تعمیـــر کد: از نظر کدی که تغییر می‌دهد به دو دســـته تقســـیم می‌گردد [۲۹]: ســـاده یا تکی؛ مرکـــب یا چندتایی. از نظر شـــیوه انجام به دو نوع دســـتی و خودکار تقسیم می‌شود.

تعمیر خودکار: دو رویکرد دارد: دستیار اشکال زدایی که با تولید پیشنهادها و توضیحات، برنامه‌ساز را همچون یک دســـتیار اشـــکال‌زدایی راهنمایی می‌کنـــد [۳۰, ۳۱]؛ و تغییردهنده کـــد [۱۰] که خودش به دو دســـته صحیح در حین ســـاخت (یا مبتنی بر ترکیب و هـــم گذاری[۷۳]) و تولید و اعتبارسنجی[۷۴] تقسیم می‌شود [۱۷]. این دو دسته اهداف ســـطح بالای مشـــترکی با هم دارند [۳۲]. فنون صحیح در حین ســـاخت با اســـتفاده از فرمول بندی رسمی مسئله و محدودیت ها، وصله‌های اســـتواری می‌ســـازند که بعد از ساخت درســـت هســـتند [۳۲]. این محدودیت ها، درستی آزمایی رسمی، قراردادها یا توصیف‌های داده شده توسط کاربر یا اســـتنتاجی هســـتند [۳۳]. درستی این وصله‌ها از نظر ریاضی قابل اثبات اســـت [۱۷]. برای هر اشـــکال داده شـــده اغلب یک وصله [۳۲] یا چند وصله محدود ســـاخته می‌شـــود. کارایی این گونـــه روش‌ها ارتباط بســـیاری با قدرت سیســـتم اثبات دارد [۳۲]. چنین شـــیوه تعمیر برای سیســـتم‌های جدید و به‌ویژه ایمنی-حیاتـــی اهمیت دارد. فنون تولید و اعتبارســـنجی با روش‌هـــای ابتکاری [۱۷] و اکتشـــاف در فضایی از تعمیرهای نامزد جست وجو کرده [۳۳] و معمولاً تعداد بسیاری از وصله‌های نامزد برای یک اشـــکال خاص می‌سازند و آن‌ها را ارزیابی می‌نمایند [۳۲]

73- Synthesis-based
74- Generate-and-validate

که آزمون روش غالب برای ارزیابی است. این طرز ساختن وصله‌ها از ایده‌های مهندســـی نرم افزار مبتنی بر جست وجو [۳۴] و آزمون مبتنی بر جهش [۳۵] یا قالب‌های تعمیر از پیش تعریف شـــده [۳۶, ۳۷, ۳۸] استفاده می‌کند. برخی از فنون تعمیر [۱۰, ۳۹] که با رویکرد تولید و اعتبارسنجی کار می‌کنند بر اساس فرضیه برنامه ساز لایق[۷۵] [۴۰] بنیان گذاری شـــده‌اند [۳۹]. راهبرد برشـــمردن تعمیرهای نامزد (ترتیب بررسی ترمیم‌های نامزد) و راهبرد آزمون (ترتیب اجرای آزمایه ها) دو عامل تأثیر گذار بر هزینه فنون تولید و اعتبارســـنجی هستند [۳۹]. فلسفه اصلی رویکرد تولید و اعتبارســـنجی این اســـت که فقط وصله‌های باورکردنی و معقولی را بپذیریم که بـــه ازای همه ورودی‌های مجموعه آزمـــون ارزیابی، خروجی درســـت می‌ســـازند [۲۶]. این رویکرد برای سیستم‌های نرم افزاری فعلی موروثی بسیار کاراست زیرا نیاز به حاشیه نویسی یا توصیف ندارد که در مورد این گونه نرم افزارها معمولاً موجود نیست [۳۲]. حتی گاهی کدهای موروثی از توصیف زبان برنامه سازیشـــان نیـــز چندان تبعیت نمـــی کنند [۴۱] و نوشـــتن قراردادهای رسمی برای آن‌ها ممکن است میسر نباشد. نشان داده شده که فنون تولید و اعتبار سنجی همزاد آزمون جهش هستند و از این رو پیشـــرفت‌های حوزه آزمون جهش را می‌توان برای تعمیر خودکار نیز به‌کار برد [۳۹].

توصیف رفتار درست برنامه: دو شکل کلی دارد [۴۲]: ضمنی: مثل الگوهای آسیب پذیری اعداد صحیح؛ صریح که خودش به سه دسته تقسیم می‌شود: آزمایه ها، توصیف‌های رســـمی و ترکیبی. توصیف‌های رسمی در سطح مختلف رسمیت وجود دارند مثل حاشیه نویسی[۷۶]، مدل رسمی در قالب منطق مرتبه اول، نمودارهای مدلســـازی یا معماری؛ حاشیه نویسی‌های ســـطح پایین در برنامه‌ها رواج بسیار دارند ولی توصیف‌های رسمی جامع هنوز نادرند [۴۳].

ســـناریوهای تعمیر: به دو دسته تقسیم می‌شوند [۴۴] شـــامل تعمیر موقتی زمـــان اجرا[۷۷] و تعمیـــر دائمی زمان

75- Competent Programmer Hypothesis (CPH)
76- Annotation
77- Workaround, Online

نگهداری[۷۸]. اولی ممکن اســت روی کــد دودویی[۷۹] یا روی حالت برنامه مثل داده ها، ثبات ها، ساختمان داده‌ها باشد. دومی نیز ممکن اســت روی کد منبــع یا روی کد دودویی قابل اجرا باشد.

انواع اشکال‌ها برای تعمیر خودکار: فنون موجود اغلب روی اشــکال‌هایی کار می‌کنند که نزدیک شدنی[۸۰] هستند؛ یعنی می‌توان آن‌ها را با روشــی دوبــاره ظاهر کرد. این گونه اشــکال‌ها را در حال حاضر تعمیرشدنی[۸۱] می‌نامند. دستهٔ دیگری از اشکال‌ها آن‌هایی هستند که ظهور و وقوع مجددشان ساده نیست؛ مثلاً غیر قطعی هستند یا در شرایط خاصی بروز می‌کنند. این اشــکال‌ها را سخت تجدیدپذیر[۸۲] می‌نامند. تقســیم بندی اشــکال‌ها به دو دستهٔ [۴۵] نزدیک شدنی و سخت تجدیدپذیر بسیار درشت دانه است ولی از منظری خاص مشکل فنون موجود را نشان می‌دهد. دسته بندی مشــابه دیگر خطاها را به دو دسته سطحی و عمیق تقسیم می‌کند [۴۶]: خطاهای عمیق شامل مواردی همچون مشکلات طراحی، خطاهای ناشی از توصیف‌های نادرست یــا ناکامل و خطاهای غیرعملکردی می‌شــود. در حالی‌که خطاهای عملکردی در دستهٔ خطاهای سطحی که در سطح متن کد دیده می‌شــوند قرار می‌گیرند. فنون موجود اغلب روی تعمیر خطاهای ســطحی کار می‌کنند. تعمیر خطاهای نوع دوم بسیار دشوارتر است.

۴- انتخاب مقالات و اوزان ارزیابی

برای آشنا کردن پژوهشــگران داخلی با حوزه تعمیر خــودکار و ارائه شــواهد نمونــه ای از مطالعه‌های انجام شــده، از بین مقالات بســیار متعدد موجود، ۳۱ مقاله که بین ســال‌های ۱۹۹۷ تــا ۲۰۱۵ در معتبرترین کنفرانس‌ها و مجلات چاپ شــده اند، مورد بررسی قرار گرفته است. مقاله‌هــای انتخابی به هیــچ وجه جامع نیســتند و صرفاً گلچینی از بهترین کارهای شــناخته شــده تاکنون هستند. مقاله‌ها طوری انتخاب شده اند که موضوع تعمیر خودکار را با رویکردهــای مختلف، روی زبان‌های مختلف و روی ســاختارهای مختلف زبان پوشــش دهند. این مقاله‌ها را می‌توان در هفت دســته قرار داد: دسته اول کارهای اولیه در حوزه تعمیر خودکار را بررســی می‌کند. دســته دوم مقالاتی هســتند که اولین پیشنهاد برای استفاده از برنامه سازی ژنتیک را جهت تعمیر خودکار مطرح کرده اند. دسته ســوم مقالاتی هســتند که کاربرد برنامه سازی ژنتیک را بــرای تعمیر خودکار در مطالعه‌های گســترده ای نشــان داده اند و همگی مســتخرج از رســاله دکتری خانم لِگُوس هســتند. دسته چهارم مطالعه‌های دیگری هستند که تعمیر خودکار را روی کدهای ماشین و روی دستگاه‌های خاص بررســی کرده اند. دســته پنجم مقالاتی هستند که تعمیر خودکار برنامه‌های زبان C را با پیشــنهاد تبدیلات خاص منظوره روی دستورهای آن حل کرده اند. دسته ششم به روش‌های تعمیر خودکار برنامه‌های زبان ایفل می‌پردازد. دسته هفتم نیز سایر مطالعه‌هایی است که تعمیر خودکار را از جنبه‌های گوناگون مورد مطالعه قرار داده اند.

وقتی فنونی در یک حوزه تحقیقاتی مرور می‌شوند، یک گام دیگر در شــناخت بهتر آن‌ها ارزیابی فنون و مقایسه آن هاســت. اگر محقق بخواهد ارزیابی‌ها و مقایســه نظام یافته و غیرســلیقه ای باشــد، بایــد ضابطه‌هایی را تعیین کنــد و بر اســاس آن‌ها روش‌هــا را مــورد ارزیابی قرار دهد. در مقاله حاضر نیز از این شــیوه اســتفاده می‌شود. پیشتر همین نویســندگان چارچوبی [۶۸] برای ارزیابی و مقایســه فنون تعمیر خودکار ارائه کرده اند که بر اساس تعداد بسیاری از معیارها پیشنهاد شده است. این معیارها با تجزیه فنون تعمیر خودکار به اجزای ساختاری مشترک، آن‌ها را مقایســه پذیر می‌نمایند. در اینجا شــش مورد از معیارهای پیشــنهادی برای ارزیابی و مقایسه فنون مورد بررســی اســتفاده می‌گردد. معیارهای مورد استفاده در اینجا و حتی در مقاله اصلی [۶۸] به هیچ وجه جامع نیستند. این معیارها بدون پیاده ســازی فنون و صرفاً با بررسی

78- Offline
79- Binary hot patching
80- Approachable
81- Fixable
82- Hard to reproduce

جدول ۱: معیارهای ارزیابی به همراه مقادیرشان

کد معیار	نام معیار	مقادیر
C1	رویکرد تعمیر	۱) صحیح در حین ساخت؛ ۲) تولید و اعتبارسنجی
C2	سیستم هدف یا حوزه تعمیر	۱) سیستم‌های موروثی؛ ۲) سیستم‌های نهفته؛ ۳) نرم افزار شیئ گرا؛ ۴) زبان‌های سخت افزار؛ ۵) همروندی؛ ۶) هیچ کدام
C3	ابزار ارزیابی و ساخت وصله‌ها	۱) آزمایه ها؛ ۲) توصیف رسمی؛ ۳) هر دو
C4	نوع تعمیر	۱) دائمی و ایستا؛ ۲) موقتی در زمان اجرا
C5	نوع خطاها	۱) خطاهای عمومی؛ ۲) خطاهای خاص
C6	زبان هدف	۱) سی؛ ۲) جاوا؛ ۳) آیفل؛ ۴) ماشین و سطح پایین؛ ۵) سخت افزار؛ ۶) ASPECTJ؛ ۷) هیچ کدام

مقاله پیشـــنهاد دهنده روش قابل اندازه گیری اند. درست اســـت که فقط شش مورد از انبوه معیارهای پیشنهادی در اینجا مورد استفاده قرار گرفته است ولی هدف نشان دادن کاربرد و ســـودمندی این معیارها در موقعیت‌های واقعی اســـت و نتایج ارزیابی‌ها برای اولیـــن بار در مقاله حاضر گزارش می‌شوند. جدول ۱ معیارها را با تخصیص کد، نام و برشـــمردن برخی از مقادیر ممکن خلاصه کرده اســـت. در پایان توضیح هر فن، معیارها و مقادیرشـــان در رشته ارزیابی برای آن فن گزارش می‌شوند. در این رشته، شش بخش به‌صورت «مقدار : معیار» که با نقطه ویرگول از هم جدا شده‌اند وجود دارد. برای نمونه C2:3 نشان می‌دهد که این فن برای تعمیر نرم‌افزارهای شـــیئ‌گرایی طراحی شده است.

۵- خلاصه فنون و ارزیابی آن‌ها

در بخش چهارم اشـــاره شد که فنون مورد بررسی در هفت دســـته ارائه می‌شوند. هفت فن در دسته اول، سه فن در دسته دوم، شـــش فن در دسته سوم، پنج فن در دسته چهارم، دو فن در دســـته پنجم، ســـه فن در دسته ششم و پنج فن در دســـته هفتم بررســـی می‌شـــوند. پایان بخش توضیح هر فن رشـــته ارزیابی است که آن فن را ارزیابی می‌کند. ساختار رشـــته ارزیابی در بخش چهارم توصیف شـــده است. فنون دسته اول کارهای اولیه در حوزه تعمیر خودکار به شـــمار می‌روند. ویژگی این فنون این است که اغلب روی تعمیر داده‌های خـــراب حین اجرای برنامه کار می‌کنند و از روش‌های مبتنی بر مدل و صوری و مشـــابه آن استفاده کرده اند.

اِســـتامپنِر و وُتاوا[۸۳] با اســـتفاده از مدل اشکال روشی برای تعمیر خطاهای کوچک در زبان VHDL (مورد استفاده در توصیف و برنامه‌سازی سخت افزار) پیشنهاد کرده اند [۴۷]. فرض اصلی آن‌ها این است که تعمیر برنامه به نسخه خطادار خیلی نزدیک است و با تغییرات کمی از روی نسخه خطادار می‌توان به نسخه درست رسید. این روش به عنوان دستیار اشکال زدایی، تعمیرهای محدودی را پیشنهاد می‌دهد و مبتنی بر مدل رسمی کار می‌نماید. روش پیشنهادی برای مهار فضای نامتناهی جست وجو فرض‌ها و محدودیت‌های بسیاری در نظر گرفته اســـت و از نظر زبان، نوع خطاها و شرایط دیگر کاملاً محدود و خاص منظوره است. به گفته این مقاله، فرض نزدیکی نسخه خطادار به برنامه صحیح یکی از روش‌های ابتکاری معقول برای محدود کردن فضای جست وجوی تعمیرهای مناســـب است. همچنین ادعا شده که سه چیز فضای انواع شیوه‌های مختلف تعمیر برنامه را محدود می‌کند: الف) دستور زبان و معنای زبان برنامه سازی؛ ب) برنامه خطادار و معیاری مربوط به پیچیدگی عملیات تعمیر؛ پ) آزمایه‌های داده شده.

معیارها: C1:1; C2:4; C3:2; C4:1; C5:2; C6: 5.

ســـیدیروگلو[۸۴] و همکاران روشـــی پویا در قالب ابزار

83- Stumptner and Wotawa
84- Sidiroglou

DYBOC ارائه کرده اند [۵۰] که با پایش زمان اجرا سعی می‌کند پس از بروز خطای امنیتی سرریز میانگیر، سیستم را احیاء نماید. این روش قسمت‌های از کد منبع زبان C را که ممکن است نسبت به حمله‌های سرریز میانگیر آسیب پذیر باشند، فراکدگذاری می‌نماید. سربار محاسباتی و افزایش حجم کد از معایب این روش هستند. همچنین وصله‌های تولید شده از نظر کیفیت ارزیابی نشده‌اند و بررسی نشده است که برنامه‌های تعمیر شده رفتار مطلوب را خراب نکرده باشند.

معیارها: C1:2; C2:1; C3:1; C4:2; C5:2; C6:1.

اِستابر[۸۵] و همکاران با استفاده از وارسی مدل سیستمی برای مکان یابی و تصحیح خودکار طراحی کرده اند [۴۸]. فن پیشنهادی مبتنی بر توصیف‌های رسمی LTL طراحی شده و در حوزه سیستم‌های حالت متناهی مثل مدارهای دیجیتال قابل استفاده است. تعمیرهای حاصل از این روش برای سیستم‌های متناهی استوار است و در شرایط خاصی کامل هم هست. چون هم اکنون توصیف‌های رسمی در کاربردهای واقعی نادرند و پیچیدگی‌های کاربردی هم دارند، این روش در حالت عمومی و برای همه قابل استفاده نیست. همچنین روش پیشنهادی بسیار پر هزینه است و برای برنامه‌های بزرگ اصلاً مقیاس پذیر نیست. ارزیابی روش پیشنهادی هم بسیار مختصر و فقط روی برنامه‌های بسیار کوچک انجام شده است.

معیارها: C1:1; C2:4; C3:3; C4:1; C5:2; C6:5.

وِیمِر[۸۶] با استفاده از وارسی مدل و سیاست‌های داده شده مبتنی بر مدل‌های رسمی، روشی برای تعمیر در حوزه سیاست‌های نقض ایمنی[۸۷] ارائه کرده است [۵۱]. روش پیشنهادی استوار است ولی فقط به یک دسته از خطاها محدود می‌شود و به خاطر اتکاء بر توصیف‌های رسمی مشکل مقیاس پذیری و پیچیدگی دارد. این روش روی برنامه‌های کوچک زبان جاوا ارزیابی شده است. از نظر نویسنده، معمولاً برنامه نویس اولیه نرم افزار در دسترس نیست و برنامه نویس فعلی گزارش اشکال را متوجه نمی شود. سودمندی روش پیشنهادی در این است که سعی می‌کند تعمیر نامزدی برای هر یک از گزارش‌های خطا بسازد تا راهنما و نقطه شروع خوبی برای تعمیر مناسب خطای مربوطه باشد.

معیارها: C1:1; C2:3; C3:2; C4:1; C5:2; C6:2.

دِمسکی[۸۸] و همکاران روشی طراحی کرده‌اند [۴۹] که با استفاده از توصیف‌های رسمی به تعمیر زمان اجرای ساختمان داده‌ها در برنامه‌های زبان C می‌پردازد. این روش فقط بر خطاهای خراب کننده ساختمان داده‌ها تمرکز دارد و سایر خطاهای منطقی را پوشش نمی‌دهد. ضمن این‌که وصله‌های تولید شده با درج کد پایش زمان اجرا ساخته می‌شوند و این‌کار موجب سربار محاسباتی می‌گردد. روش پیشنهادی برای کارش نیاز به توصیف سازگاری ساختمان داده و ساختمان دادهٔ نقض کننده این خصوصیات دارد و روی کد منبع برنامه فراکدگذاری انجام می‌دهد. علاوه بر این‌ها، چگونگی ارزیابی وصله‌های تولید شده نامعلوم است و بررسی هم نشده که تعمیرهای تولید شده رفتار از پیش درست برنامه را خراب نکنند.

معیارها: C1:2; C2:1; C3:1; C4:2; C5:2; C6:1.

مشابه روش قبلی، لوکاستو[۸۹] و همکاران نیز روشی برای تعمیر زمان اجرا طراحی کرده اند [۵۲]. روش پیشنهادی کدهایی در کد دودویی زمان اجرا درج می‌کند که رفتار زمان اجرا را پایش نماید و به همین دلیل سربار اجرایی دارد. این روش از رفتار اجرایی برنامه الگوبرداری می‌نماید, آن‌ها را با اطلاعاتی حاصل از تحلیل ایستا روی کد ترکیب می‌کند و از اطلاعات به دست آمده جهت کشف اشکال و تعمیر مناسب زمان اجرا بهره برداری می‌نماید. روش پیشنهادی برای استفاده در سیستم‌های خود درمان[۹۰]، نرم‌افزارهای موروثی[۹۱] و سیستم‌های آماده موجود در بازار[۹۲] کاربرد دارد.

85- Staber
86- Weimer
87- Safety violation policy

88- Demsky
89- Locasto
90- Self-healing
91- Legacy
92- Commercial-Off-The-Shelf (COTS)

معیارها: C1:2; C2:1; C3:1; C4:2; C5:2; C6:1.

مشابه روش دمسکی، الکارابلیه و خورشید[93] روشی تحت عنوان Juzi طراحی کرده اند [۵۳] که برای تعمیر زمان اجرای ساختمان داده‌های پیچیده استفاده می‌شود. این روش به عنوان ورودی ردهٔ جاوایی حاوی تعریف ساختمان داده و روشی حاوی محدودیت‌های رسمی صحت دریافت می‌کند و از فراکدگذاری و جست وجوی نظام یافته مبتنی بر اجرای نمادین برای به دست آوردن تعمیر مناسب استفاده می‌نماید. روش کلی کار این است که ساختمان داده را آن قدر جهش و تغییر می‌دهد تا محدودیت‌های داده شده برآورده شوند. این روش نیاز به پایش زمان اجرا دارد که موجب بروز سربار محاسباتی می‌گردد.

معیارها: C1:2; C2:3; C3:3; C4:2; C5:2; C6:2.

دسته دوم سه مقاله‌های هستند که در آن‌ها ایده استفاده از برنامه سازی ژنتیک جهت تعمیر خودکار مطرح شده و توسعه داده شده است. آرکوری[94] برای اولین بار [۵۴] ایده تعمیر خودکار خطاهای نرم افزاری را با کمک برنامه سازی ژنتیک پیشنهاد داد. برنامه خطادار یکی از اعضای جمعیت در نظر گرفته می‌شود و آن قدر تکامل پیدا می‌کند که توصیف‌های رسمی یا آزمایه‌های داده شده را برآورده سازد. او فرض کرده است که برنامه سازان برنامه‌ها را به طور تصادفی نمی نویسند و می‌توان فرض کرد که نرم افزار خطادار از نظر ساختاری شباهت زیادی با برنامه بدون خطا دارد.

معیارها: C1:2; C2:6; C3:3; C4:1; C5:1; C6:7.

آرکوری سپس ایده خود را توسعه داده [۵۵] و با ارائه یک مثال انگیزشی، جزئیات بیشتری از روش پیشنهادی تشریح کرده و آن را با فرمول‌ها و تعاریف دقیق، رسمی سازی کرده است. او توضیح داده که تعمیر خودکار برنامه‌ها شباهت بسیاری با مسئله برنامه‌سازی خودکار دارد و به مراتب از آن ساده تر است. منتها تفاوت این دو مسئله این است که اولی از برنامه خطادار شروع می‌کند که احتمالاً نزدیکی ساختاری زیادی با برنامه معادل بدون خطایش دارد؛ در حالی‌که برنامه‌سازی خودکار از چند برنامه اولیه که کاملاً تصادفی ساخته شده اند شروع می‌کند. آرکوری در دو مقاله اولیه خود صرفاً ایده پیشنهادی را روی شبه کدی برای مرتب سازی حبابی بررسی و ارزیابی کرده است.

معیارها: C1:2; C2:6; C3:3; C4:1; C5:1; C6:7.

بعد از این دو مقاله، آرکوری ایده خود را تغییر و توسعه اساسی داده است [۵۶] و ارزیابی مفصل تری با ارائه ابزار نمونه برای برنامه‌های زبان جاوا صورت داده است. او ادعا کرده است که روش مورد مطالعه اش بدون هرگونه محدودیتی قابل استفاده برای تعمیر انواع خطاهاست. برای طراحی روش توسعه یافته، آرکوری عملگرهای جدیدی برای جست وجو بر مبنای فنون مکان یابی ارائه کرده که فضای جست وجو را کاهش دهد. او روش خودش را با الگوریتم تصادفی و تپه نوردی مقایسه کرده است. برای این‌که حین تکامل برنامه‌ها جهت تعمیر، از تولید برنامه‌هایی که مشکل گرامری دارند اجتناب شود، دستور زبان در عملگرها و عملکرد داخلی برنامه‌سازی ژنتیک دخالت داده شده است.

معیارها: C1:2; C2:3; C3:1; C4:1; C5:1; C6:2.

دسته سوم مقاله‌های گسترده ای هستند که همگی دستاوردهای پژوهشی خانم لگوس در رساله دکتری وی هستند. اولین پیاده‌سازی واقعی از برنامه سازی ژنتیک برای تعمیر برنامه‌های زبان C استفاده شد [۵۷] که تابع برازندگی بر اساس آزمایه‌های مثبت و منفی طراحی شده است. روش پیشنهادی با الهام از پیشنهاد آرکوری طراحی شده و روی ۱۱ برنامه واقعی C ارزیابی شده است. هر چند نویسندگان ادعا کرده اند روش پیشنهادی شان روی همه خطاها کار می‌کند اما عملاً آن‌ها فقط به تعمیر تعدادی از خطاهای امنیتی پرداخته اند و البته اذعان کرده اند که روش آن‌ها برای تعمیر خطاهای هم روندی قابل استفاده

93- Elkarablieh and Khurshid
94- Arcuri

نیســت. درخت خلاصه نحوی ســاختمان مورد استفاده در پیاده سازی روش پیشــنهادی نویسندگان بوده است. همچنین آن‌ها برای خنثی کردن مشــکل تــورم کد که در روش‌های مبتنی بر برنامه‌سازی ژنتیک رایج است، از فنون اشــکال زدایی خودکار به نام اشــکال زدایی دلتا استفاده کرده‌اند.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

همان گروه نویســندگان با انجام توســعه‌ها در روش پیشنهادی خود، همان کار را با توضیحات و ارزیابی‌های بیشــتر نیز ارائه کرده اند [۹]. روش پیشــنهادی با فرض برنامه ســاز لایق طراحی شده که فرض می‌کند کد تعمیر خطا در محل دیگری در همان کد وجود دارد. نویســندگان روششــان را روی کدهایــی با مجموع ۶۳ هــزار خط کد واقعی ارزیابی کرده اند. مسئله اینجاست که کدهای تعمیر شــده صرفاً روی آزمایه‌های موجود اجرای موفق دارند و بنابرایــن کیفیت آن‌ها به خوبی مجموعه آزمون موجود است.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

ســپس برای بهبود نتایج، نویسندگان به طراحی توابع برازندگــی بهتــر [۵۸] و طراحی عملگرهــا و نمایش‌های بهتری [۵۹] برای عملیات برنامه سازی ژنتیک حین تعمیر خــودکار پرداختند. از آنجائی‌که حین فرایند تکامل، کیفیت برنامه میانی در حال تغییر بــا کمک تابع برازندگی اندازه گیری می‌شود، طراحی توابع برازندگی که نزدیکی برنامه در حال تکامل را به برنامه صحیح، بهتر اندازه گیری کند، ســرعت همگرایی را بیشتر می‌کند و کیفیت برنامه حاصل را افزایش می‌دهد. طراحی تابع برازندگی مناسب برای هر حوزه کاربردی برنامه سازی ژنتیک به‌ویژه تعمیر خودکار ضرورت دارد و باعث بهبود چشــمگیر نتایج می‌شود. این موضوع بــرای عملگرهای برش و جهش مورد اســتفاده و نمایــش میانــی برنامه‌ها حین تکامل مصــداق دارد. در الگوریتم‌هــای فرا ابتکاری همچون برنامه ســازی ژنتیک، روش کلی کار در همه کاربردها تقریباً یکســان است ولی برای هر مســئله ای باید اجــزای مخصوص آن کاربرد را طراحی کرد.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

را سپس نویســندگان الگوریتم کامل روش پیشنهادی خود را همــراه با ارزیابی‌های مفصلــی روی برنامه‌های زبان C با حدود یک میلیون خط کد و هشت کلاس خطا در پژوهش دیگری ارائه کردند [۱۰]. در این مقاله، نویسندگان ضمن ارائه ابــزار GenProg، مطالب مفصلی جهت ارزیابی برنامه‌های تعمیر شده از جنبه‌های مختلف ارائه کرده اند. در مقاله مذکور نویســندگان سیستم کشف نفوذ و تعمیر همزمان خودکاری نیز بر اساس ایده تعمیر خودکار ارائه و تشریح کرده اند. عدم نیاز به توصیف‌های رسمی، نداشتن ســربار محاســباتی و کدهای اضافی پایش زمان اجرا از مزیت‌های عمده روش پیشنهادی است.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

در پژوهــش بعدی [۸] نویســندگان به طور نظام یافته تعدادی برنامه به زبان C همراه با آزمایه‌های لازم و سایر داده‌های مورد نیاز را تهیه کردند و در دسترس عموم قرار دادنــد. این مجموعه آماده شــده می‌تواند به عنوان محکی جهت ارزیابی هر روش جدید پیشــنهادی و مقایسه آن با GenProg مورد اســتفاده قرار گیرد. نویسندگان GenProg را روی مجموعه تهیه شده ارزیابی کرده اند و نشان داده شده که هر یک از ۵۵ خطای تعمیر شده فقط به هشت دلار منابع محاسباتی نیاز داشته است.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

دســته چهارم فنونی که در اینجا مورد بررســی قرار می‌گیرد، سایر پژوهش‌هایی هستند که بر پایه ایده برنامه ســازی ژنتیک و توســط یک یا چند مورد از نویسندگان کارهای دسته سوم مطالعه شــده اند. شالت[95] و همکاران ایده تعمیر خودکار کدهای اســمبلی را بــا تکیه بر برنامه سازی ژنتیک مورد مطالعه قرار داده اند [۶۰]. آن‌ها روش خــود را روی کدهای اســمبلی و بایت کدهای جاوا مورد ارزیابی قرار داده اند. اکثر تصمیمات طراحی مشابه کارهای

95- Shulte

دسته سوم است با این تفاوت که برای نمایش برنامه‌ها از آرایه خطی استفاده شده است. روش پیشنهادی روی تعدادی از برنامه‌های سیستم عامل یونیکس، یک کارساز وب و معادل اسمبلی برنامه‌های C کارهای دسته سوم ارزیابی شده است. نتایج آزمایش‌ها نشان می‌دهد که خصوصیت‌های عملکردی برنامه‌های تعمیر شده نسبت به تغییرات و دستکاری‌های کد در سطح اسمبلی، بسیار مقاوم است که از آن به نام استحکام جهشی[۹۶] در ادبیات یاد می‌شود.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

سپس همین نویسندگان کارشان را در مقاله بعدی [۶۱] توسعه دادند و روشی برای تعمیر خطاهای دلخواه نرم افزاری روی سیستم‌های نهفته که محدودیت‌های اساسی در فضای حافظه و پردازنده دارند، طراحی کردند. روش پیشنهادی آن‌ها با چند نوآوری الگوریتمی، بدون نیاز به فراکدگذاری و ماشین مجازی اجرا می‌شود. این روش روی کدهای اسمبلی و دودویی‌های ELF[۹۷] کار می‌کند و روی گوشی هوشمند نوکیا N900 پیاده سازی شده و نشان داده شده که هر گوشی برای تعمیر خودکار به طور متوسط هفت پیام کوتاه ارسال می‌کند.

معیارها: C1:2; C2:2; C3:1; C4:1; C5:1; C6:4.

وِیمر و همکارانش بر پایه GenProg روش دیگری توسعه داده اند [۳۹] که سعی کرده مشکل اجرای زمانگیر آزمایه‌ها را از سه جهت حل کند: ترتیب دهی در ارزیابی وصله‌های نامزد؛ اولویت دهی آزمایه‌ها و شناسایی وصله‌های که از نظر معنایی معادلند. با تعمیر برنامه‌هایی با حدود پنج میلیون خط کد زبان C حاوی ۱۰۵ خطا، روش پیشنهادی مورد ارزیابی قرار گرفته است. این روش هم مشابه GenProg به چند آزمایه مثبت و منفی نیاز دارد و بر پایه فرضیه برنامه ساز لایق طراحی شده است.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

اسمیت[۹۸]، لگوس و همکاران در مطالعه‌ای [۳۳] روش GenProg و روش مشابه دیگری را روی نسخه‌های مختلف شش برنامه به زبان C حاوی ۹۹۸ خطا مورد ارزیابی دقیقتر قرار دادند. حاصل مطالعه‌های آن‌ها نشان داد که وصله‌های ساخته شده آن چنان ساخته می‌شوند که به شدت برازنده مجموعه آزمایه‌های مورد استفاده هستند. پس کیفیت وصله به کیفیت آزمایه‌های مورد استفاده وابسته می‌شود و گاهی وصله ساخته شده خطای مورد نظر را برطرف می‌کند ولی اشکال‌های دیگری در عملکرد برنامه به‌وجود می‌آورد. چون آزمایه‌هایی در مجموعه آزمون برای بررسی این اشکال‌های جدید وجود ندارد، متوجه بروز آن‌ها نمی شویم. حاصل کار اینست که وصله‌های تولیدی ضمن تعمیر رفتار معیوب برنامه، برخی از رفتارهای درست برنامه را نیز خراب می‌کنند. این مشکلات در اثر استفاده از آزمایه‌ها برای ارزیابی وصله‌ها پدید می‌آید.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

یالین که[۹۹]، لگوس و همکاران در پژوهش دیگری با استفاده از جست وجوی معنایی در کد اقدام به طراحی روش دیگری برای تعمیر خودکار برنامه‌ها کرده اند [۴۲]. این روش در مخزنی از کدهای واقعی به دنبال تکه کدی می‌گردد که شباهت معنایی با تکه کد معیوب داشته باشد. سپس تکه کد پیدا شده را جایگزین کد معیوب می‌کند و آن را با آزمایه‌های موجود مورد ارزیابی قرار می‌دهد. بنابراین، نیاز به مخزنی از کدهای آماده و نیز ابزار خودکار حل کننده محدودیت ها[۱۰۰] دو نیازمندی اساسی این روش هستند. ابزار حل محدودیت‌ها معمولاً بار محاسباتی سنگینی دارد و می‌تواند گلوگاه روش پیشنهادی در کاربردهای واقعی باشد.

معیارها: C1:2; C2:1; C3:3; C4:1; C5:1; C6:1.

در دسته پنجم دو روش خاص مورد بررسی قرار می‌گیرد که اولی برای تعمیر خطاهای هم روندی و دومی

96- Mutational robustness
97- Executable and Linkable Format
98- Smith
99- Yalin Ke
100- Constraint solver

برای تعمیر خطاهـــای خاصی در زبـــان C کاربرد دارند. برادبوری و جالبرت[101] با استفاده از برنامه سازی ژنتیک و تابع برازندگی مناسب و با معرفی سه عمگر جهش خاص، روشی را برای تعمیر خطاهای هم روندی پیشنهاد کرده اند [۶۲]. این روش روی کد منبع کار می‌کند و قادر اســت دو خطای رقابت داده ای و بن بست را تعمیر نماید.

معیارها: C1:2; C2:5; C3:1; C4:1; C5:2; C6:2.

کوکر و حفیظ[102] با پیشـــنهاد ســـه تبدیـــل در کد منبع برنامه‌های C توانستند چهار نوع از خطاهای ناشی از داده صحیح را در برنامه‌های زبان C تعمیر نمایند [۳۶]. تبدیلات پیشـــنهادی ســـعی می‌کنند با حفظ رفتار موجود برنامه، برخی از مشـــکلات امنیتی در سطح کد را برطرف سازند. روش پیشـــنهادی روی ۷۱۴۷ برنامـــه و حدود ۱۵ میلیون خط کد برنامه‌های واقعی مورد ارزیابی قرار گرفته اســـت. نتایج آزمایش‌ها نشـــان داده که ســـربار ناشی از تبدیلات اعمال شـــده بسیار کم است و به‌ندرت رفتار اصلی برنامه تغییر کرده است.

معیارها: C1:1; C2:1; C3:1; C4:1; C5:2; C6:1.

دسته ششـــم از فنون مورد بحث مطالعه‌های شاخص روی زبان‌های شـــیءگرای ویژه همچون ایفل هســـتند که تحت سرپرستی برتراند مِیر[103] صورت گرفته است. دالمِیر، آندریـــاس زِلِر و مِیـــر[104] روش PACHIKA را برای تعمیر برنامه‌هـــای زبان ASPECTJ پیشـــنهاد و طراحی کرده‌اند [۶۳]. ایـــن روش با اســـتخراج مـــدل رفتاری اشـــیاء بین اجراهای موفق و ناموفق و پیدا کردن اختلاف آن ها، تعمیر کد را می‌سازد و با مجموعه آزمایه‌های پس نمایی آن‌ها را ارزیابـــی می‌نماید. این روش نیاز به فراکدگذاری دارد و از تحلیل‌های ایستا و پویای کد استفاده می‌نماید. خود روش با ۳۰ هزار خط کد جاوا پیاده ســـازی شده است. ارزیابی این روش آن قدر محدود است که نتایج تعمیم پذیر نیستند.

معیارها: C1:2; C2:3; C3:1; C4:1; C5:2; C6:6.

گروه دیگری از محققان به سرپرستی زِلِر و مِیر روش دیگری پیشنهاد داده اند [۶۴] که با استفاده از قراردادها به تعمیر خودکار برنامه‌های زبان ایفل می‌پردازد. این روش AutoFix-E نامیده می‌شـــود و تعمیرهای ساخته شده آن روش استوار هستند. وجود قراردادهای رسمی پیش نیاز اســـتفاده از این روش است و به خاطر اســـتفاده از ابزار اثبات قضیه سربار محاسباتی بالایی دارد.

معیارها: C1:1; C2:3; C3:2; C4:1; C5:2; C6:3.

سپس نویسندگان این کار تحقیقی را بسیار گسترش دادند [۴۵] و ارزیابی‌های مفصلی روی برنامه‌های بزرگتر انجام دادند. ایـــن برنامه‌ها حاوی حدود ۲۰۰ اشـــکال با کیفیـــت و بلوغ مختلف هســـتند. این مقاله بـــرای ارزیابی وصله‌ها دو گونه وصله‌های معتبر و محض[105] را معرفی کرده است.

معیارها: C1:1; C2:3; C3:2; C4:1; C5:2; C6:3.

نهایتاً در دســـته هفتم فنون متنوع دیگری قرار دارند که هر کدام جنبه‌های ویژه ای از بحث تعمیر خودکار را مورد مطالعه قرار داده‌انـــد. کیم[106] و همکاران با ترکیب GenProg و مجموعـــه ای از الگوهای تعمیر که توســـط برنامه سازان انسانی پیشنهاد شده، روش PAR [۳۸] را بـــرای تولید خودکار وصله‌های تعمیر نرم افزار طراحی کـــرده اند. ادعا شـــده که چون قالب‌های تعمیر توســـط برنامه سازان انسانی ساخته شده اند، بر مشکل تغییرات تصادفی و بی معنی شـــدن وصله‌های حاصل از روش GenProg غلبه خواهد شـــد. همچنیـــن قالب‌های خودکار اگرچه دستی ساخته می‌شـــوند ولی این فرایند فقط یک بار انجام می‌شـــود و بارها مورد استفاده قرار می‌گیرد. ایـــن مقاله معیار قابلیت پذیرش را برای وصله‌ها معرفی کرده اند کـــه معادل وصله‌های محـــض تعمیر خودکار برنامه‌های زبان ایفل [۴۵] اســـت. روش پیشنهادی روی شـــش برنامه جاوا با حـــدود ۵۰۰ هزار خط کد آزمایش شـــده اســـت. در این آزمایش‌ها ۲۷ اشـــکال از بین ۱۱۹

101- Bradbury and Jalbert
102- Coker and Hafiz
103- Bertrand Meyer
104- Dallmeier, Andreas Zeller, and Meyer
105- Valid and proper
106- Kim

اشکال تعمیر شده ولی جزئیات آزمایش‌ها و ارزیابی‌ها نامعلوم است.

معیارها: C1:2; C2:3; C3:1; C4:1; C5:1; C6:2.

کی[۱۰۷] و همکاران ضمن اندازه گیری مسئله بیش برازش و بررسی آن در سه روش تعمیر خودکار، روش جدیدی به نام kali را طراحی و پیشنهاد کرده اند [۲۶]. نتایج آزمایش‌ها روی سه روش تعمیر نشان داده که اکثریت قریب به اتفاق وصله‌های تولیدی درست نیستند و صرفاً تغییر ساده ای هستند که دو پیامد بد دارند: ایجاد آسیب پذیری‌های امنیتی و از بین بردن رفتار مطلوب برنامه. روش جدید با فضای جست وجوی ساده تر و کاراتر سعی می‌کند بر دو مشکل مذکور غلبه نماید. این روش تحلیل دقیقتری برای پذیرش وصله‌های تولیدی انجام می‌دهد و وصله ای پذیرفته می‌شود که روی همه آزمایه‌های مجموعه آزمون اعتبارسنجی خروجی درستی تولید کند.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

تَن و روی چودهاری[۱۰۸] با کمک آزمایه‌ها برای تعمیر اشکال‌های پس نمایی روش relifix [۶۵] را پیشنهاد داده‌اند. در این روش با بررسی دستی ۷۳ اشکال پس نمایی، مجموعه ای از تبدیلات کد استخراج شده است. برای تعمیر کد، روی عملگرهای تبدیل کد جستجو می‌کند که این عملگرها در این مقاله معرفی شده اند. بر خلاف GenProg که به عملیات کمینه سازی روی وصله تولید شده نیاز دارد، روش جدید از این گام بی نیاز است زیرا تغییرهای کمی روی کد صورت می‌دهد. در ارزیابی relifix نشان داده شده که ۲۳ اشکال از بین ۳۵ اشکال پس نمایی تعمیر شدند در حالی‌که GenProg فقط پنج مورد از این اشکال‌ها را توانست تعمیر نماید.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

روش RSRepair که توسط کی و همکاران پیشنهاد شده [۶۶] برای تعمیر خودکار خطاهای موجود در برنامه‌های زبان C طراحی شده است. این روش با همان عملیات جهش شبیه GenProg کار می‌کند ولی به جای برنامه سازی ژنتیک از جست وجوی تصادفی استفاده می‌نماید. ادعا شده که جست وجوی تصادفی محدودیت‌های Gen-Prog را ندارد. روش پیشنهادی طوری طراحی شده که تا بفهمد وصله نامزد روی آزمایه ای اجرای ناموفق دارد، آن را کنار می‌گذارد. همچنین با بهره گیری از اولویت دهی، فرایند شناخت وصله نامعتبر سریعتر هم می‌گردد. بنابراین در مقایسه با GenProg تعداد وصله‌های نامزد کمتری تولید می‌شود و با اجرای آزمایه‌های کمتر، کارایی کلی روش بالاتر رفته است.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:1; C6:1.

آخرین فن تعمیر خودکار مورد بررسی در این مقاله NOPOL [۶۷] نام دارد و توسط دی مارکو[۱۰۹] و همکاران برای تعمیر دو اشکال بخصوص طراحی و پیاده سازی شده است. این روش برنامه ای معیوب و مجموعه ای از آزمایه‌های آن را دریافت می‌کند (تعدادی مثبت و دست کم یک منفی) و با فراکدگذاری کدهای منبع داده‌های را جمع آوری می‌نماید. سپس این داده‌ها به نمونه ای از مسئله SMT[۱۱۰] تبدیل می‌شود و در صورت حل مسئله، آن را به کد وصله تعمیری ترجمه می‌نماید. روش پیشنهادی روی برنامه‌های شیئ‌گرای جاوا ارزیابی شده است ولی ارزیابی‌ها بسیار ضعیفند و نتایج قابل تعمیم نیستند.

معیارها: C1:2; C2:1; C3:1; C4:1; C5:2; C6:2.

۶- درس‌هایی از فنون پیشین، مشکلات موجود و سیر تحقیقی آینده

از هر مطالعه‌ای باید درس‌های آموخت. نقاط مثبت هر مطالعه در طراحی روش‌های بعدی به کار می‌رود و از نقاط منفی باید در فنون بعدی اجتناب کرد. با تحلیل و تعمق روی درس‌ها و شناخت مشکلات موجود، می‌توان مسیر درستی در تحقیقات آتی انتخاب کرد. این بخش برای

107- Qi
108- Tan and Roychoudhury
109- DeMarco
110- Satisfiability Modulo Theory

دسترسی آسان تر به مسیر درست تحقیقی در آینده تنظیم شده است.

درس‌هـــای حاصـــل از روش‌های پیشـــین و نتایج آزمایش‌هـــای آن ها: ارزیابی وصله‌ها با آزمایه‌ها غالب تریـــن روش مورد اســـتفاده در اکثر فنون اســـت. علت این اســـت که آزمایه‌ها ســـاده ترین دســـت ساخته‌های نرم افزاری موجود هســـتند یا به ســـادگی قابل تولیدند. توصیف‌های رســـمی منجر به تولید وصله‌های استوار می‌شوند و اگر مقیاس نرم افزار کوچک باشد یا عملکرد درســـت آن اهمیت حیاتی داشته باشد، استفاده از آن‌ها معقـــول و توجیه پذیر اســـت. روش‌های ملهم از طبیعت همچـــون روش تکاملی برنامه ســـازی ژنتیک در زمره بهترین روش‌هایی است که در عمل برای تعمیر خودکار برنامه‌هـــا مورد اســـتفاده قرار گرفته انـــد. طراحی تابع برازندگی مناســـب و خاص حوزه تعمیر خودکار، نقش اساســـی در موفقیـــت، کارایی و ســـودمندی روش‌های تکاملی تعمیر خودکار داشته اســـت. استفاده از کدهای موجود کـــه در مخزن‌های نرم افزاری موجودند انتخاب مناسبی برای جست وجو به دنبال کدهای تعمیر مناسب است. اغلب روش‌های که ادعا می‌کنند انواع خطا را تعمیر می‌کننـــد، صرفاً قادر به تعمیـــر خطاهای غیرهم روندی هســـتند. روش‌های تعمیر برای هر زبان برنامه ســـازی جداگانه باید طراحی شـــود زیرا هر زبـــان ویژگی‌های دســـتوری و معنایی مخصـــوص به خـــود دارد. فراهم کردن قالب‌های از پیش ساخته شـــده برای تعمیر انواع خاصی از خطاها، تأثیر اساســـی در کیفیت و خوانایی و قابلیت نگهداری وصله‌های تولید شده دارد. هر چه تعداد رده‌های خطاهایی که روش تعمیر روی آن‌ها کار می‌کند کمتر باشد، سودمندی روش بیشتر می‌گردد.

مشـــکلات و چالش‌های موجود: تعمیر مطلقاً خودکار تقریباً برای فنون موجود مصداق ندارد. چون تنظیم پارامتر، دادن ورودی مناســـب، مقداردهی‌هـــای اولیه و پیش‌گوی آزمون، نوشتن توصیف‌های رسمی و امثال این‌ها نیازهای اولیه هر فن تعمیر هستند. غیر از آزمایه‌ها از توصیف‌های رســـمی هم می‌توان برای ارزیابـــی وصله‌های حاصل از تعمیر استفاده کرد. مســـئله اینجاست که تولید دستی این توصیف‌ها بسیار دشوار است و نرخ مثبت کاذب در اغلب فنون خـــودکار موجود که آن‌ها را از متن برنامه‌ها کاوش و استنتاج می‌کنند بســـیار بالاست. تعمیر خودکار معمولاً فرایندی اســـت مبتنی بر آزمون و خطا و بســـیاری تعمیر باید ارزیابی شـــوند تا بالاخره تعمیر معتبر و مناسبی پیدا شود. لذا صرف نظر از اطمینان پذیری و کیفیت، تعداد زیاد آزمایه‌ها یـــا وجود آزمایه‌های که اجـــرای طولانی دارند، فرایند ارزیابی وصله‌های تعمیری را بســـیار زمان گیر و غیر عملی می‌سازد.

مســـیر تحقیقاتی آینده: چند زمینه برای تحقیقات آتی ضروری هســـتند. اول طراحی فنونی اســـت که بر تعمیر کارای انواع اشکال‌های هم‌روندی تمرکز داشته باشد. دوم اندازه گیری و بهبود کیفیت وصله‌های تولیدی است. مسئله سوم تولید وصله‌هایی است نه تنها اشکال هدف را برطرف ســـازند، بلکه وصله‌های تولید کنند که خوانا و قابل درک برای برنامه ساز انسانی باشد و قابلیت نگهداری نرم‌افزار را کاهش ندهد. همچنین نیاز به فنونی داریم که سه مرحله آزمون نرم‌افزار، مکان‌یابی اشکال و تعمیر را به‌طور یک‌جا انجام دهد.

۷- نتیجه‌گیری

ایـــن مقاله بـــرای اولین بار، تعاریف، مفاهیم و دســـته بندی لازم در حـــوزه تعمیر خـــودکار برنامه‌ها را تجمیع کرده اســـت. سپس ۳۱ فن شاخص را خلاصه‌سازی کرده و روشی برای ارزیابی آن‌ها معرفی کرده است و فنون را ارزیابی نموده است. در نهایت با برجسته کردن درس‌های حاصل از مطالعه‌ها، مشکلات موجود و مسیر تحقیقی آینده را روشن کرده است. تولید وصله‌های تعمیری باکیفیت با فنون تولید و اعتبارسنجی یکی از نیازهای اساسی تحقیقی در آینده اســـت. برای پژوهشگرانی که قصد شروع تحقیق

در حوزه تعمیر خــودکار برنامه‌ها را دارند، مطالعه مقاله حاضر توصیه می‌شود.

مراجع

1- Jeffrey, D. B. (2009). Dynamic state alteration techniques for automatically locating software errors (Doctoral dissertation, University of California Riverside).

2- Zeller, A. (2009). Why programs fail: a guide to systematic debugging. Elsevier.

3- Kung, D. (2013). Object-oriented Software Engineering: An Agile Unified Methodology. McGraw-Hill Higher Education.

4- Hailpern, B., & Santhanam, P. (2002). Software debugging, testing, and verification. IBM Systems Journal, 41(1), 4-12.

5- Britton, T., Jeng, L., Carver, G., Cheak, P., & Katzenellenbogen, T. (2013). Reversible debugging software. Technical Report, University of Cambridge, Judge Business School.

6- Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., & Bairavasundaram, L. (2011). How do fixes become bugs?. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (pp. 26-36). ACM.

7-Malik, M. Z., Ghori, K., Elkarablieh, B., & Khurshid, S. (2009). A case for automated debugging using data structure repair. In Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on (pp. 620-624). IEEE.

8-Le Goues, C., Dewey-Vogt, M., Forrest, S., & Weimer, W. (2012). A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In Software Engineering (ICSE), 2012 34th International Conference on(pp. 3-13). IEEE.

9-Weimer, W., Nguyen, T., Le Goues, C., & Forrest, S. (2009). Automatically finding patches using genetic programming. In Proceedings of the 31st International Conference on Software Engineering (pp. 364-374). IEEE Computer Society.

10-Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2012). GenProg: A generic method for automatic software repair. Software Engineering, IEEE Transactions on, 38(1), 54-72.

11-Weiss, Cathrin, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. (2007). "How long will it take to fix this bug?." In Proceedings of the Fourth International Workshop on Mining Software Repositories, p. 1. IEEE Computer Society.

12-Naone, E. (2010). So Many Bugs, So Little Time. MIT Technology Review.

13-CNN, "Will bugs scare off users of new Windows 2000," Feb 2000.

14-Jin, W., & Orso, A. (2012). BugRedux: reproducing field failures for in-house debugging. In Proceedings of the 34th international conference on software engineering (pp. 474-484). IEEE Press.

15-Le Goues, C. (2013). Automatic program repair using genetic programming (Doctoral dissertation, University of Virginia).

16-Harman, M. (2010). Technical Perspective Automated Patching Techniques: The Fix Is In. Communications of the ACM, 53(5).

17-Le Goues, C., Holtschulte, N., Smith, E. K., Brun, Y., Devanbu, P., Forrest, S., & Weimer, W. (2015). The ManyBugs and IntroClass benchmarks for automated program repair. IEEE Transactions on Software Engineering. 41(12), 1236-1256.

18-Weimer, W. (2013). Advances in automated program repair and a call to arms. In Search Based Software Engineering (pp. 1-3). Springer Berlin Heidelberg.

19-Le Goues, C., Forrest, S., & Weimer, W. (2013). Current challenges in automatic software repair. Software Quality Journal, 21(3), 421-443.

20-Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. Information and software technology, 51(1), 7-15.

21-Ammann, P., & Offutt, J. (2016). Introduction to software testing. Cambridge University Press.

22-Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2013). A comprehensive survey of trends in oracles for software testing. University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01.

23-Gustafson, S., Ekárt, A., Burke, E., & Kendall, G. (2004). Problem difficulty and code growth in genetic programming. Genetic Programming and Evolvable Machines, 5(3), 271-290.

24-Ali, S., Andrews, J. H., Dhandapani, T., & Wang, W. (2009). Evaluating the accuracy of fault localization techniques. In Proceedings of the 2009 IEEE/ACM international conference on automated software engineering (pp. 76-87). IEEE Computer Society.

25-Böhme, M., & Roychoudhury, A. (2014). Corebench: Studying complexity of regression errors. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 105-115). ACM.

26-Qi, Z., Long, F., Achour, S., & Rinard, M. (2015). An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In Proceedings of the 2015 International Symposium on Software Testing and Analysis (pp. 24-36). ACM.

27-Qi, Z., Long, F., Achour, S., & Rinard, M. (2015). An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. Technical Report MIT-CSAIL-TR-2015-021.

28-Wirsing, M., & Hoelzl, M. (2011). Rigorous Software Engineering for Service-oriented Systems: Results of the SENSORIA Project on Software Engineering for Service-oriented Computing (Vol. 6582). Springer Science & Business Media.

29-Tao, Y., Dang, Y., Xie, T., Zhang, D., & Kim, S. (2012). How do software engineers understand code changes?: an exploratory study in industry. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of

generation learned from human-written patches: essay on the problem statement and the evaluation of automatic software repair. In Proceedings of the 36th International Conference on Software Engineering (pp. 234-242). ACM.
45-Pei, Y., Furia, C., Nordio, M., Wei, Y., Meyer, B., & Zeller, A. (2014). Automated fixing of programs with contracts. Software Engineering, IEEE Transactions on, 40(5), 427-449.
46-Wei, Y., Pei, Y., Furia, C. A., Silva, L. S., Buchholz, S., Meyer, B., & Zeller, A. (2010). Automated fixing of programs with contracts. In Proceedings of the 19th international symposium on Software testing and analysis (pp. 61-72). ACM.
47-Stumptner, M., & Wotawa, F. (1997). Model-based program debugging and repair. In Proc. Ninth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'96) (pp. 155-160).
48-Staber, S., Jobstmann, B., & Bloem, R. (2005). Finding and fixing faults. In Advanced Research Working Conference on Correct Hardware Design and Verification Methods (pp. 35-49). Springer Berlin Heidelberg.
49-Demsky, B., Ernst, M. D., Guo, P. J., McCamant, S., Perkins, J. H., & Rinard, M. (2006). Inference and enforcement of data structure consistency specifications. In Proceedings of the 2006 international symposium on Software testing and analysis (pp. 233-244). ACM.
50-Sidiroglou, S., Giovanidis, G., & Keromytis, A. D. (2005). A dynamic mechanism for recovering from buffer overflow attacks. InInternational Conference on Information Security (pp. 1-15). Springer Berlin Heidelberg.
51-Weimer, W. (2006). Patches as better bug reports. In Proceedings of the 5th international conference on Generative programming and component engineering (pp. 181-190). ACM.
52-Sidiroglou, S., Locasto, M. E., Boyd, S. W., & Keromytis, A. D. (2007). From STEM to SEAD: Speculative execution for automated defense. In USENIX Annual Technical Conference. USENIX.
53-Elkarablieh, B., & Khurshid, S. (2008). Juzi: A Tool for Repairing Complex Data Structures. In Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on (pp. 855-858). IEEE.
54-Arcuri, A. (2008). On the automation of fixing software bugs. InCompanion of the 30th international conference on Software engineering (pp. 1003-1006). ACM.
55-Arcuri, A., & Yao, X. (2008). A novel co-evolutionary approach to automatic software bug fixing. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on(pp. 162-168). IEEE.
56-Arcuri, A. (2011). Evolutionary repair of faulty software. Applied Soft Computing, 11(4), 3494-3514.
57-Forrest, S., Nguyen, T., Weimer, W., & Le Goues, C. (2009). A genetic programming approach to automated software repair. In Proceedings of the 11th Annual conference on Genetic and evolutionary computation (pp. 947-954). ACM.
58-Fast, E., Le Goues, C., Forrest, S., & Weimer, W. (2010). Designing better fitness functions for automated program repair. In Software Engineering (p. 51). ACM.
30-Jeffrey, D., Feng, M., Gupta, N., & Gupta, N. (2009). BugFix: A learning-based tool to assist developers in fixing bugs. In Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on (pp. 70-79). IEEE.
31-Kaleeswaran, S., Tulsian, V., Kanade, A., & Orso, A. (2014). Minthint: Automated synthesis of repair hints. In Proceedings of the 36th International Conference on Software Engineering (pp. 266-276). ACM
32-Weimer, W. (2013). Advances in automated program repair and a call to arms. In Search Based Software Engineering (pp. 1-3). Springer Berlin Heidelberg.
33-Smith, E. K., Barr, E. T., Le Goues, C., & Brun, Y. (2015). Is the cure worse than the disease? overfitting in automated program repair. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 532-543). ACM.
34-Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys (CSUR), 45(1), 11.
35-Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. Software Engineering, IEEE Transactions on, 37(5), 649-678.
36-Coker, Z., & Hafiz, M. (2013). Program transformations to fix C integers. In Proceedings of the 2013 International Conference on Software Engineering (pp. 792-801). IEEE Press.
37-Perkins, J. H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., ... & Rinard, M. (2009). Automatically patching errors in deployed software. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (pp. 87-102). ACM.
38-Kim, D., Nam, J., Song, J., & Kim, S. (2013). Automatic patch generation learned from human-written patches. In Proceedings of the 2013 International Conference on Software Engineering (pp. 802-811). IEEE Press.
39-Weimer, W., Fry, Z. P., & Forrest, S. (2013). Leveraging program equivalence for adaptive program repair: Models and first results. In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on (pp. 356-366). IEEE.
40-Fry, Z. P., Landau, B., & Weimer, W. (2012). A human study of patch maintainability. In Proceedings of the 2012 International Symposium on Software Testing and Analysis (pp. 177-187). ACM.
41-Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., ... & Engler, D. (2010). A few billion lines of code later: using static analysis to find bugs in the real world. Communications of the ACM, 53(2), 66-75.
42-Ke, Y., Stolee, K. T., Le Goues, C., & Brun, Y. (2015). Repairing programs with semantic code search. In Proceedings of the 30th IEEE/ACM International Conference On Automated Software Engineering (ASE), Lincoln, NE, USA.
43-Le Goues, C., & Weimer, W. (2012). Measuring code quality to improve specification mining. Software Engineering, IEEE Transactions on, 38(1), 175-190.
44-Monperrus, M. (2014). A critical review of automatic patch

pair. In Proceedings of the 12th annual conference on Genetic and evolutionary computation (pp. 965-972). ACM.
59- Le Goues, C., Weimer, W., & Forrest, S. (2012). Representations and operators for improving evolutionary software repair. In Proceedings of the 14th annual conference on Genetic and evolutionary computation (pp. 959-966). ACM.
60- Schulte, E., Forrest, S., & Weimer, W. (2010). Automated program repair through the evolution of assembly code. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 313-316). ACM.
61- Schulte, E., DiLorenzo, J., Weimer, W., & Forrest, S. (2013). Automated repair of binary and assembly programs for cooperating embedded devices. In ACM SIGPLAN Notices (Vol. 48, No. 4, pp. 317-328). ACM.
62- Bradbury, J. S., & Jalbert, K. (2010). Automatic repair of concurrency bugs. In International symposium on search based software engineering—fast abstracts (pp. 1-2).
63- Dallmeier, V., Zeller, A., & Meyer, B. (2009). Generating fixes from object behavior anomalies. In Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on (pp. 550-554). IEEE.
64- Wei, Y., Pei, Y., Furia, C. A., Silva, L. S., Buchholz, S., Meyer, B., & Zeller, A. (2010). Automated debugging of programs with contracts. In Proceedings of the 19th international symposium on Software testing and analysis (pp. 61-72). ACM.
65- Tan, S. H., & Roychoudhury, A. (2015). relifix: Automated repair of software regressions. In Proceedings of the 37th International Conference on Software Engineering-Volume 1 (pp. 471-482). IEEE Press.
66-Qi, Y., Mao, X., Lei, Y., Dai, Z., & Wang, C. (2014). The strength of random search on automated program repair. In Proceedings of the 36th International Conference on Software Engineering (pp. 254-265). ACM.
67-DeMarco, F., Xuan, J., Le Berre, D., & Monperrus, M. (2014). Automatic repair of buggy if conditions and missing preconditions with SMT. In Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis (pp. 30-39). ACM.
68-Khalilian, A., Baraani-Dastjerdi, A., & Zamani, B. (2016). On the evaluation of automatic program repair techniques and tools. In Electrical Engineering (ICEE), 2016 24th Iranian Conference on (pp. 61-66). IEEE.