تعمیر خطاهای نرم‌افزار

[۱]علیرضا خلیلیان، دانشجوی دکتری نرم‌افزار

[۲]احمد براآنی دستجردی، دانشیار گروه مهندسی نرم‌افزار

[۳]بهمن زمانی، استادیار گروه مهندسی نرم‌افزار

[۱، ۲، ۳] گروه مهندسی نرم‌افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران

{[1]khalilian, [2]ahmadb, [3]zamani}@eng.ui.ac.ir

۱-۱- مقدمه

براساس اطلاعات نیویورک تایمز واژه نرم‌افزار در سال ۱۹۵۸ به‌وسیله یک متخصص فن آمار به‌نام جان ویلدر توکی در قالب مقاله‌ای در ماهنامه‌ی ریاضی آمریکا ابداع گردید. از آن به بعد استفاده از کامپیوتر و نرم‌افزار به‌طور پیوسته در جامعه رواج یافت. به‌نظر می‌رسد به‌ویژه طی چند دهه گذشته تعداد نرم‌افزارهایی که در دنیا مورد استفاده قرار می‌گیرد به‌طور چشمگیری افزایش داشته است، به‌طوری که در سال ۱۹۵۸ قابل پیش بینی نبود. از آموزش تا اقتصاد، از امور نظامی تا پزشکی و از هنر تا انواع سرگرمی‌ها، نرم‌افزارها تبدیل به ابزاری شده‌اند که جامعه برای انجام امور به آن متکی است. می‌توان برای نرم‌افزار مزایای بیشماری را نام برد. با وجود این فواید، توسعه‌ی نرم‌افزار با چالش‌های مهمّی نیز همراه است که می‌بایست مورد توجه اساسی قرار گیرد. یکی از مهمترین این چالش‌ها این حقیقت است که نرم‌افزار همیشه رفتار پیش‌بینی شده از خود نشان نمی‌دهد؛ به‌عبارت دیگر نرم‌افزار همیشه به شیوه‌ی مطمئن رفتار نمی‌کند. با توجه به کاربرد گسترده‌ی نرم‌افزار کامپیوتر در دنیای امروزی عواقب عدم اطمینان از نرم‌افزارها می‌تواند بسیارخطرناک باشد. این عواقب ناخوشایند می‌تواند هم در قالب هزینه‌های مالی و هم در قالب هزینه‌های انسانی بروز نموده و اندازه‌گیری شود.

یکی از دلایل اصلی عدم اطمینان نرم‌افزار وجود خطاها در کد منبع برنامه است. متأسفانه این خطاها به‌طور مکرر در نرم‌افزار رخ می‌دهند و باید رفع گردند. تجربه نشان می‌دهد که ایجاد نرم-افزار بدون اشکال بسیار دشوار است و اشکال‌زدایی برنامه دو برابر دشوارتر از وقتی است که آنرا از ابتدا می‌نویسیم. به‌همین دلیل هنوز نرم‌افزارها همراه با نقص‌های[۱] شناخته شده و شناخته نشده عرضه می‌شوند.

به‌همین دلیل شرکت‌های نرم‌افزاری برای حفظ موقعیت خود در بازار، نرم‌افزار را با خطاهایی که با آزمون کشف شده‌اند به بازار عرضه می‌کنند و متعهد می‌شوند در مدت کوتاهی خطاها را رفع کنند. برای نمونه سیستم عامل ویندوز ۲۰۰۰ با ۶۳،۰۰۰ خطاهای شناخته شده به بازار عرض شد

[۱] نقص ترجمه واژه defect است که گاهی اصطلاحاً به آن bug گفته می‌شود. البته واژه bug در کل مبهم است و به هر سه مفهوم عیب یا اشکال (fault)، خطا (error) و خرابی (failure) اشاره می‌کند.

که علت آن ناکافی بودن منابع آزمون بود. تعمیر خطا معمولاً از طریق عرضه وصله‌هایی[1] برای نرم-افزار صورت می‌گیرد که خطاهای مزبور را رفع[2] می‌نماید. تولید وصله از فعالیت‌های ضروری در مرحله نگهداری نرم‌افزار محسوب می‌شود و بیشتر عمر نرم‌افزار هم در مرحله نگهداری آن سپری می‌شود. مطابق با گزارش‌های موجود تا حدود ۹۰ درصد از کل هزینه‌های پروژه‌های نرم‌افزاری برای فعالیت‌های مختلف نگهداری نرم‌افزار مصرف می‌شود.

با توجه به مطالب یاد شده، یافتن اشکال‌های برنامه می‌تواند **پیش از عرضه** نرم‌افزار[3] صورت گیرد یا **بعد از استقرار** آن[4] انجام شود. قطعاً همه تولیدکنندگان نرم‌افزار می‌خواهند **همه** اشکال-ها و خطاها پیش از عرضه کشف و رفع شوند. به دلایلی که مطرح شد، این‌کار حداقل فعلاً قابل تحقق نیست. متأسفانه رفع خطاها بعد از عرضه نرم‌افزار گران است و مشکلاتی جدی دارد؛ مثلاً می‌تواند باعث شود که اعتبار شرکت نرم‌افزاری در بازار رقابتی به‌خطر بیافتد و نیز ممکن است موجب شود داده‌های حساس و خصوصی کاربران حین اشکال‌زدایی فاش شود. به‌خاطر هزینه بالای عیوب نرم‌افزار پس از استقرار، محققان و تولیدکنندگان نرم‌افزار تمرکز خود را بر یافتن و رفع خطاها در حین توسعه گذاشته‌اند. در عمل هنوز همه تولیدکنندگان نرم‌افزار خطاها را در دو مرحله مذکور کشف و رفع می‌کنند، هر چند که تمایل داریم این فرایند بطور کامل در مرحله پیش از عرضه صورت گیرد. انگیزه خودکارسازی آزمون و اشکال‌زدایی هم تسریع و کم‌هزینه کردن آن‌هاست تا تعداد بیشتری از خطاها پیش از عرضه کشف و رفع شوند.

برای پاسخ به مشکلات مسایل مطرح شده، فنون اشکال‌زدایی و تعمیر خودکار مطرح شده‌اند. تعمیر خودکار موضوعی است که هنوز در حد آکادمیک نیز به بلوغ نرسیده و مشکلات اساسی دارد و به‌عنوان مسئله باز تحقیقاتی در حال بررسی است. از حدود سال ۲۰۰۹ توسعه فنون خودکار برای تعمیر برنامه‌ها آغاز شده است. این فنون ابتدا روی زبان‌های C/C++ توسعه پیدا کرده‌اند زیرا حجم بسیاری از نرم‌افزارهای موجود در صنعت و کاربردهای دفاعی و نظامی با این زبان‌ها توسعه پیدا کرده‌اند. سپس این فنون به زبان‌های اسمبلی و کدهای جاوا در حال گسترش هستند. تعمیر خودکار برنامه‌ها از یافتن اشکال‌ها دشوارتر است.

---

[1] Patch

[2] Bug-fix, repair, correction, or patching

[3] Before-release

[4] Post-deployment

بنابر گفته توماس زیمِرمان[1] از مرکز تحقیقات مایکروسافت، باید شرایطی را پیدا کنیم که تعمیر خودکار در آنها مفید است:

«یکی از چالش‌ها شناسایی موقعیت‌های زمانی و مکانی است که تعمیر خودکار را می‌توان به‌کار گرفت. من انتظار ندارم که تعمیر برنامه‌ها برای همه خطاها در دنیا کار کند (در این‌صورت هزاران برنامه‌نویس بیکار خواهند شد)، اما اگر از قبل حوزه‌هایی که در آن‌ها کار می‌کند را بشناسیم، توانایی زیادی به‌دست می‌آوریم.[2]»

مسئله اینجاست که فنون توسعه یافته هنوز در صنعت وارد نشده‌اند و پژوهش وسیعی در این حوزه در حال انجام است. حتی اگر فن خودکار تعمیر همواره وصله‌های خوب و با کیفیت برای تعمیر تولید نکند، تعمیرهای پیشنهادی آن می‌تواند به‌عنوان راهنمایی برای برنامه‌نویسان در اشکال‌زدایی مورد استفاده قرار گیرد؛ ضمن اینکه چنین تعمیرهایی را (که از کیفیت بالا برخوردار نیستند) اغلب می‌توان به‌عنوان نوعی کمک اولیه در سیستم معیوب مستقر کرد تا حداقل جلوی از کارافتادگی سیستم گرفته شود و آنرا قادر به ادامه اجرا سازد. کمی افزایش توانایی ما برای رفع خطاها و خودکارسازی فنون اشکال‌زدایی تأثیر چشمگیر در کیفیت و بهره‌وری نرم‌افزار می‌گذارد. هدف نهایی این است که فنون تعمیر خودکار آن‌چنان مقیاس‌پذیر شوند که با هزینه کم و کارایی بالا در صنعت قابل استفاده گردند که مزایای زیر برای آن‌ها متصور است:

- جایگزین شدن فنون خودکار به‌جای برنامه‌نویسان انسانی
- کاهش قابل ملاحظه هزینه‌های سازمان
- افزایش سرعت تعمیر برنامه‌ها
- افزایش اعتمادپذیری نرم‌افزارها و متعاقباً سیستم‌ها

۱-۲- تعمیر نرم‌افزار

مطالعه‌ها نشان می‌دهند که برنامه‌نویسان تصادفی برنامه نمی‌نویسند؛ به‌همین دلیل برنامه‌ی خطادار از نظر ساختاری بسیار شبیه و نزدیک به برنامه‌ی درست است. ضمن اینکه نشان داده شده

[1] Thomas Zimmermann

[2] http://www.cs.virginia.edu/_weimer/p/weimer-ssbse2013.pdf

است که اکثریت تعمیرهای نرم‌افزار کوتاه و ساده هستند. لذا با بهره‌گیری از این خاصیت برنامه-نویسان قادرند با تغییرات کمی برنامه خطادار را تصحیح نمایند. ضمن اینکه تولید همه تعمیرهای کوچک ممکن نیز از نظر محاسباتی امکان‌پذیر می‌گردد. با این حال علی‌رغم اینکه تغییرات لازم برای اصلاح برنامه‌ها معمولاً کم و کوچک هستند، اما فضای حالت‌های ممکن برای تغییر برنامه در کل نامتناهی است. به این دلیل فنون خودکار تعمیر برنامه‌ها به‌دنبال راه‌کارهایی برای کاهش و مهار این فضای حالت هستند. مهندسی نرم‌افزار مبتنی بر جست‌وجو یکی از راه‌های مؤثری است که در چند سال اخیر برای این منظور شناسایی شده است؛ زیرا فرایند تعمیر دستی خطاها بسیاری از خصوصیت‌های مسئله جست‌وجو را دارد که هدفش اعمال چندین تغییر در کد منبع برنامه است تا رفتار مسئله‌ساز به‌طرز مطلوبی مرتفع گردد.

تعمیر، کد را تغییر می‌دهد و از این نظر به دو دسته قابل تقسیم است:

۱. تعمیر ساده یا تکی
۲. تعمیر مرکب[1] یا چندتایی (درک تعمیر مرکب معمولاً دشوار است و نیاز به تلاش مضاعف دارد)
   أ. تغییرات روی یک فایل صورت گیرد یا چند فایل
   ب. تغییرات روی یک ویژگی صورت گیرد یا چند ویژگی
   ت. تغییرات برای حل یک مشکل باشد یا چند مشکل (اشکال)

۱-۲-۱- تعمیر دستی

برای اینکه برنامه‌نویس بتواند به‌طور دستی تعمیر مناسبی برای اشکال داده شده پیدا کند باید سه اطلاع به‌دست آورد:

۱. توصیف رفتار مطلوب را بفهمد (نرم‌افزار درست چه رفتاری باید از خودش نشان دهد)؛
۲. آن بخش از پیاده‌سازی که حاوی اشکال است کجاست؛
۳. رفتار مطلوب و پیاده‌سازی اشکال‌دار چه تفاوتی با هم دارند و تفاوت کجاست.

معمولاً برنامه‌نویسان برای بازرسی برنامه و بررسی قدم به قدم آن از ابزارهایی استفاده می-

[1] Composite

کنند که اجرای منجر به اشکال را ردیابی کنند یا اطلاعاتی در مورد ماهیت خرابی به‌دست آورند. ابزارهای GDB و Valgrind نمونه‌هایی هستند که برای این منظور استفاده می‌شوند. ابزار Valgrindبرای اشکال‌زدایی خطاهای حافظه بسیار مفید است. چنین ابزارهایی نیاز به بررسی‌ها و استدلال‌های پیچیده در ساختار برنامه و گراف جریان کنترلی آن دارند تا منشأ مشکل کاملاً معلوم شود.

تعمیر خوب در حالت ایده‌آل باید پنج خصوصیات داشته باشد:

۱. رفتار معیوب را درست کند؛
۲. اشکال جدیدی در نرم‌افزار ایجاد نکند؛
۳. عملکرد و رفتار درست فعلی نرم‌افزار را حفظ کند؛
۴. با اهداف طراحی کلی سیستم سازگاری داشته باشد؛
۵. قابلیت نگهداری نرم‌افزار در آتی همچنان امکان‌پذیر باشد[۱].

مسئله اینجاست که اندازه‌گیری این خصوصیات بسیار دشوار است چه برسد به اینکه بخواهیم تأمین شدنشان را ضمانت کنیم. برای نمونه اندازه‌گیری قابلیت نگهداری نرم‌افزار در طول زمان مسئله تحقیقاتی فعالی است و هنوز حل نشده است.

۱-۲-۲- تعمیر خودکار

تعمیر خودکار روال مورد استفاده در تعمیر دستی را اقتباس می‌کند و نوعی سیستم پیشنهادگر برای مهندسی نرم‌افزار محسوب می‌شود. برای تعمیر خودکار برنامه‌ها تاکنون دو **رویکرد** مورد استفاده قرار گرفته است:

۱. رویکرد اول با تولید پیشنهادها و توضیحات برنامه‌نویس را همچون یک دستیار اشکال-زدایی راهنمایی می‌کند.
۲. رویکرد دوم با اعمال تغییراتی روی برنامه آن‌را تعمیر می‌نماید. این رویکرد خودش به دو

[۱] سه چیز برای نگهداری موفق نرم‌افزار لازم است: رده‌بندی و اولویت‌بندی اشکال‌ها، کشف موارد تکراری و رفع خطاها.

دسته از فنون تقسیم می‌شود که **صحیح در حین ساخت**[1] (مبتنی بر سنتز و هم-گذاری[2]) و **تولید و اعتبارسنجی**[3] نام دارند. این دو دسته اهداف سطح بالای مشترکی با هم دارند. دقت شود که آزمایه‌ها در هر یک از دو دسته فنون رویکرد دوم ممکن است به-کار روند.

فنون تولید و اعتبارسنجی باید چند مانع را برای یافتن وصله‌های مناسب از میان بردارند:

1. **فضای اشکال**[4]: مجموعه مکان‌هایی از برنامه که باید تغییر داده شوند همراه با احتمال تغییر هر کدام (فنون مکان‌یابی خطا یکی از راه‌های مقابله با این مانع است.)
2. **فضای تعمیر**[5]: شیوه‌های بسیار گوناگون تغییر تکه کد محتمل به اشکال برای تعمیر آن (یک راه‌حل مقابله این‌است که فرض کنیم اشکال موجود در برنامه در جای دیگر همان کد یا کد سایر پروژه‌ها، به‌درستی پیاده‌سازی شده و عمگرهای روش تعمیر، کد مناسب را از مکان دیگر همان برنامه اشکال‌دار، نسخه‌های دیگر آن برنامه یا کد سایر پروژه‌ها در مخازن کد برداشت کنند؛ فرض وجود کد مناسب تعمیر در مکان دیگر همان برنامه اشکال‌دار که بر اساس مشاهده‌ها تقویت شده است را تکرار داخلی[6] می‌گویند. اختصاصاً نشان داده شده است[7] که حجم بزرگی از نرم‌افزارها چه از نظر گرامری و ساختاری و چه از نظر معنایی افزونه هستند. همچنین نشان داده شده که دستورها یا عبارت‌های مورد نیاز برای تعمیر اغلب در اعزام[8] قبلی برنامه‌ها موجود است.)

---

[1] همچنین به آن صحیح به‌وسیله روش ساخت هم می‌گویند؛ یعنی روش ساخت، صحیح بودن آن‌را ضمانت می‌کند. صحیح به‌وسیله روش ساخت و امن به‌وسیله روش ساخت دو روش طراحی برنامه‌ها هستند. روش‌های صحیح یا امن به‌وسیله روش ساخت با فرایند توسعه نرم-افزار کاملاً مهندسی برخورد می‌کنند؛ یعنی ابتدا قبل از تولید کد، مدل ریاضی از طراحی ساخته می‌شود. از این مدل برای استدلال در مورد راه‌حل پیشنهادی استفاده می‌شود. همچنین مدل ریاضی برای تضمین نمایش رفتار درست و تأمین همه عملکردهای ضروری نیز مورد استفاده قرار می‌گیرد. برنامه‌ای که به این طریق تولید می‌شود، آزمون می‌شود ولی آزمون جهت اعتبارسنجی فرایند صحیح به‌وسیله روش ساخت مورد استفاده قرار می‌گیرد نه یافتن خطاها - Correct-by-construction

[2] Synthesis-based

[3] Generate-and-validate

[4] Fault space

[5] Fix space

[6] Internal repetition

[7] مطالعه ۶۰۰۰ پروژه نرم‌افزاری و بیش از ۴۲۰ میلیون خط کد.

[8] در سیستم‌های کنترل نسخه (نگارش)، هر اعزام آخرین تغییرات را به کد منبع موجود در مخزن اضافه می‌کند و کد موجود با این تغییرات به‌عنوان آخرین نسخه نرم‌افزار تا آن لحظه شناخته می‌شود - Commit

۳. **کمینه‌سازی کد تعمیری (تورم کد):** این مانع مخصوص روش‌هایی است که مبتنی بر برنامه‌نویسی ژنتیک[۱] کار می‌کنند همچون GenProg. یکی از راه‌حل‌ها، کمینه‌سازی کد متورم در انتهای تولید وصله تعمیری است.

۴. **اعتبارسنجی وصله کاندید:** فنون تعمیر وصله‌های کاندید را اعتبارسنجی می‌کنند و بر اساس نتایج اعتبارسنجی فضای وصله‌های کاندید پیمایش می‌شود. اعتبارسنجی وصله کاندید یا در حالت کلی‌تر ارزیابی کیفیت آن تاکنون در مطالعه‌ها با روش‌هایی همچون آزمایه‌های موجود، آزمایه‌های کنار گذاشته شده[۲] و آزمون فاز، قضاوت‌های انسانی در مورد مورد قابلیت نگهداری و قابلیت پذیرش، ارزیابی توسط Red Teams و یا ترکیبی از این موارد صورت گرفته‌اند. هیچ‌کدام از این روش‌ها ضمانت همه جانبه بر کیفیت تعمیر تولیدی نمی‌کنند.

هر فن تعمیر معمولاً سه ورودی دارد:

۱. برنامه خطادار در شکل کد منبع، دودویی یا اسمبلی[۳]
۲. توصیفی از رفتار درست و مطلوب برنامه که باید حفظ شود
۳. سرنخی از خطا.

با کمک این سه ورودی باید دو اطلاع مهم را از برنامه به‌دست آوریم:

۱. مکان خطا
۲. حالتی از برنامه که در آن خطا رخ داده (منجر به خطا شده) است.

۱-۳- نتیجه‌گیری

مقاله حاضر سعی کرد که مفهوم تعمیر خودکار نرم‌افزار را به‌طور بسیار خلاصه معرفی کند. برای این منظور، ابتدا اهمیت پرداختن به مسئله خطاهای نرم‌افزاری تبیین شد. سپس بحث تعمیر

---

[۱] تعمیر خودکار با برنامه‌نویسی ژنتیک کاریست شبیه جراحی پلاستیک که از اول شروع نمی‌کند، بلکه از ناحیه مشکوک شروع کرده و از کد سایر قسمت‌ها بهره می‌برد. در نهایت، کمینه‌سازی و پاک‌سازی تعمیر را مؤثر و تمیز می‌سازد.

[2] Held-out

[۳] می‌توان به آن برنامه آگاه از اشکال هم اطلاق کرد - Bug-aware.

خطاهای نرم‌افزار با شروع از روش‌های دستی و در ادامه تعمیر خودکار نرم‌افـزار مـورد بحـث قـرار گرفت. همچنین دسته‌بندی‌های اولیه فنون تعمیر و روش‌های آن‌ها نیز در این مقاله معرفی گشت.

مراجع

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

[2] Le Goues, C. (2013). *Automatic program repair using genetic programming* (Doctoral dissertation, University of Virginia).

[3] Arcuri, A. (2011). Evolutionary repair of faulty software. *Applied Soft Computing*,*11*(4), 3494-3514.

[4] Dallmeier, V., Zeller, A., & Meyer, B. (2009, November). Generating fixes from object behavior anomalies. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering* (pp. 550-554). IEEE Computer Society.

[5] Forrest, S., Nguyen, T., Weimer, W., & Le Goues, C. (2009, July). A genetic programming approach to automated software repair. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 947-954). ACM.

[6] Liu, P., & Zhang, C. (2012, June). Axis: Automatically fixing atomicity violations through solving control constraints. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 299-309). IEEE Press.

[7] Le Goues, C., Holtschulte, N., Smith, E. K., Brun, Y., Devanbu, P., Forrest, S., & Weimer, W. The ManyBugs and IntroClass benchmarks for automated program repair. *IEEE Transactions on Software Engineering. In Press.*

[8] Kim, D., Nam, J., Song, J., & Kim, S. (2013, May). Automatic patch generation learned from human-written patches. In *Proceedings of the 2013 International Conference on Software Engineering* (pp. 802-811). IEEE Press.

[9] CNN, "Will bugs scare off users of new Windows 2000," Feb 2000.

[10] National Vulnerability Database. http://nvd.nist.gov/statistics.cfm, April 2006.

[11] Kernighan, B. W., & Plauger, P. J. (1978). The elements of programming style. *The elements of programming style, by Kernighan, Brian W.; Plauger, PJ New York: McGraw-Hill, c1978.*, *1*.

[12] Zeller, A. (2009). *Why programs fail: a guide to systematic debugging*. Elsevier.

[13] The Guardian. (2006). Why we all sell code with bugs.

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

[15] Seacord, R. C., Plakosh, D., & Lewis, G. A. (2003). *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional.