مروری بر روش های آزمون نرم افزارهای شی گرا واولویت بندی موارد آزمون در آنها

سمیه عشوریان[1]، شهره آجودانیان*[2]

۱- دانشجوی کارشناسی ارشد ، دانشکده مهندسی کامپیوتر، واحد نجف آباد ،دانشگاه آزاد اسلامی ، نجف آباد ، ایران

۲- عضو هیات علمی ، دانشکده مهندسی کامپیوتر ،واحد نجف آباد ،دانشگاه آزاد اسلامی ، نجف آباد ، ایران

ashurians@sco.iaun.ac.ir

چکیده

آزمون نرم افزارهای شیءگرا به دلیل پیچیدگی و ویژگی‌های خاص آن یک کارفرآیند چالش‌برانگیز می‌باشد. آزمون شیءگرایی به همان اندازه مهم است که آزمون‌های سنتی اهمیت دارند. اغلب رویکردهایی که برای آزمون شیءگرایی استفاده می‌شود، تکنیک‌های آزمون سنتی می‌باشد. با توجه به تفاوت‌های شیءگرایی مانند پنهان‌سازی، چندشکلی و ارث‌بری نیازمند تعریف رویکردهای جدید برای آزمایش می‌باشد. برخلاف سیستم‌های سنتی که واحد آزمون آن‌ها زیر برنامه است، شیءگرایی بر مبنای آزمون کلاس می‌باشد. با پیشرفت روزافزون روش‌های طراحی نرم‌افزار، روش‌های آزمون نرم افزار نیز می‌بایست به فراخور این تغییر، به‌روز شوند. آزمون نرم افزار یکی از مراحل مهم چرخه حیات توسعه نرم‌افزار است. آزمون نرم افزار زمان زیادی می‌گیرد و هزینه‌ی آزمون نرم افزار به‌سرعت در حال افزایش است. هر صنعت نرم‌افزاری عمدتاً نگرانی‌هایی در مورد کاهش هزینه‌های آزمون و عیب‌یابی نرم‌افزار با در نظر گرفتن حداقل زمان دارد. در این مقاله مروری بر روش های آزمون نرم افزارهای شی گرا واولویت بندی موارد آزمون در آنها می باشد .

کلمات کلیدی: کلمات کلیدی: آزمون نرم افزار، نرم‌افزار شیءگرا، اولویت بندی مورد آزمون ، توسعه نرم‌افزار

۱–مقدمه

برای ارائه نرم‌افزار باکیفیت، شناسایی خطاهای احتمالی و تعمیر آن‌ها ضروری است. عوامل متعددی نشان‌دهنده کیفیت نرم‌افزار می‌باشد. این عوامل عبارت‌اند از کارآمدی، صحت، کامل بودن، کار آیی، قابلیت حمل، قابلیت استفاده، قابلیت اطمینان، یکپارچگی، قابلیت نگهداری و غیره. با توجه به پیشرفت تکنولوژی و انتظارات مشتری، ارائه نرم‌افزار باکیفیت در بودجه تخصیص‌یافته، کار چالش‌برانگیزی است.

کیفیت نرم‌افزار نیز به مهارت و تجربه‌ی توسعه‌دهنده وابسته است. امروزه زبان‌های مختلف توسط توسعه‌دهندگان مورداستفاده قرارمی‌گیرند. هر زبان دارای ویژگی‌های مختلفی است و کارایی‌های متفاوتی دارد. بسیاری از بسته‌ها نیز در بازار موجود هستند که به‌صورت مستقیم در نرم‌افزار وارد می‌شوند. اگر ویژگی‌های مختلف زبان و بسته‌ها به‌طور مؤثر مورداستفاده قرار نگیرند، ممکن است در نرم‌افزار خطای احتمالی ایجاد شود [۲۶].

اگر توسعه‌دهندگان از شرایط برنامه‌نویسی آگاهی نداشته باشند، قطعاً باگ ایجاد خواهد کرد. گاهی اوقات کارآمدی پیاده‌سازی شده عاری از خطا است، اما ممکن است حافظه بیشتری اشغال کرده و زمان اجرای بیشتری صرف کند. مصرف زیاد حافظه و زمان اجرا بر کیفیت نرم‌افزار تأثیر می‌گذارد. بنابراین باید تکنیکی برای شناسایی عواملی که قادر به معرفی خطا، افزایش زمان اجرای نرم‌افزار و مصرف حافظه بیشتر باشند، ارائه گردد. اولویت‌بندی موارد آزمون باید به نحوی صورت گیرد که حداکثر خطاهای قابل‌تشخیص زودتر تشخیص داده شوند [۱۸].

آزمون نرم افزار یک مرحله مهم در چرخه حیات توسعه نرم‌افزار (SDLC[۱]) است و زمان قابل‌توجهی را به خود اختصاص می‌دهد. یک استراتژی مناسب برای اثربخشی آزمون نرم افزار در انجام فعالیت‌های آزمایشی موردنیاز است [۴۶].

بنابراین آزمون نرم افزار، آزمون محصول نرم‌افزاری ازنظر کار آیی و دقت آن است و آزمون‌ها با این هدف انجام می‌شوند که بتوان خطاهای نرم‌افزار را در حداقل زمان و تلاش تعیین کرد. آزمون باید تشخیص دهد که محصول طبق مشخصات تعیین‌شده است یا خیر. باید نشان دهد که محصول دارای کیفیت خوبی است و باید قادر به آزمایش مواردی باشد که خطاهای ناشناخته را پوشش می‌دهند [۳۴]. آزمون نرم افزار با یک معضل بزرگ روبرو می‌باشد، از یک‌سو فکر کردن در مورد این‌که نرم‌افزار باید با خطاهای صفر ساخته شود، و از سوی دیگر هدف اصلی گروه آزمون فهمیدن حداکثر اشتباهات در حداقل زمان و آزمون خروجی محصول به هر ورودی احتمالی آزمایش‌شده است و این بدان معنی است که ممکن است معتبر یا نامعتبر باشد [۲۲]. عمدتاً دو نوع آزمون داریم، آزمون استاتیک و آزمون دینامیک. دو رویکرد اصلی آزمون عبارت است از رویکرد کاربردی و رویکرد ساختاری.

۲- آزمون نرم افزار

آزمون نرم افزار یکی از اصلی‌ترین تکنیک‌های اولیه برای به دست آوردن کیفیت بالای نرم‌افزار می‌باشد. آزمون نرم افزار برای تشخیص وجود خطاها که به علت خرابی نرم‌افزارها است انجام می‌شود. بااین‌حال، آزمون نرم افزار کار وقت‌گیر و گرانی است. برای پیاده‌سازی یک نرم‌افزار می‌بایست اصولی را رعایت نمود که معمولاً عبارت‌اند از: جمع‌آوری داده، تجزیه‌وتحلیل داده‌ها، طراحی، آزمون و پیاده‌سازی [۱۵].

آزمون محصول شامل عملیات بررسی برای وجود خطای عملیاتی و غیرعملیاتی است. خطای غیرعملیاتی، بر روی مدل نرم‌افزار قابل‌بررسی است و درنتیجه طراحی غیر صحیح به وجود می‌آید. عواملی که ممکن است در به وجود آمدن خطاهای عملیاتی که عموم ناشی از پیاده‌سازی غیر صحیح است مؤثر باشند عبارت‌اند از: حلقه، شرط، پرش، فراخوانی، ارسال پارامتر و... [۳۴].

عوامل ذکرشده در لیست فوق در نرم‌افزارهای تهیه‌شده به روش شیءگرا و رویه‌ای مشترک هستند. حال اینکه در نرم‌افزارهای شیءگرا عوامل دیگری نیز دخیل هستند. عوامل مؤثر درروش شیءگرا ناشی از مفاهیم جدید مطرح‌شده در این رویکرد شیءگرا بوده و بخشی از این عوامل در ذیل آمده‌اند [۴۱].

- وراثت
- ارتباط(تناظر)
- فراخوانی برون کلاس
- پارامترهای پیچیده(ارسال کلاس به‌عنوان پارامتر)
- پیاده‌سازی واسط و...

هر راهکاری که بخواهد نرم‌افزار شیءگرا را موردبررسی قرار دهد، می‌بایست موارد مذکور را مدنظر داشته باشد.

[۱] Software development life cycle

طراحی نرم‌افزار به روش شیءگرایی مزایای بسیاری مانند قابلیت استفاده مجدد، تجزیه مسائل پیچیده و نگهداری نرم‌افزار ایجاد می‌نماید. شیءگرایی به‌سرعت در حال تبدیل‌شدن یک پارادایم برای طراحی سیستم‌های مقیاس بزرگ می‌باشد [۱۹].

۳–ادبیات تحقیق

اندازه آزمون متناسب با تکامل نرم‌افزار، افزایش می‌یابد. موارد آزمون، برای آزمون برنامه موجود مورداستفاده قرارمی‌گیرند. اگر نرم‌افزار به علت اضافه شدن قابلیت‌های جدید اصلاح شود، موارد جدید آزمون می‌تواند به موارد آزمون موجود اضافه شود. محدودیت‌های زیادی در صنعت مانند منابع، زمان و هزینه وجود دارد. بنابراین مهم است که موارد آزمون اولویت‌بندی شوند تا احتمال تشخیص خطا بالاتر و زودتر صورت گیرد. در این بخش، یک مرور کلی روی کارهای محققان مختلف درزمینه‌ی آزمون نرم افزار ارائه‌شده است.

Saraph و همکاران [۳۵] روشی برای شناسایی خودکار موارد مهم آزمون ارائه کرده‌اند، این موارد شامل خصوصیات ورودی هستند که به ارزش خروجی کمک می‌کند، کاهش تعداد موارد آزمون برای شناسایی روابط ورودی و خروجی است که باعث صرفه‌جویی در منابع آزمون می‌شود. روش ارائه‌شده یک روش آزمون جعبه سیاه است که از شبکه عصبی استفاده کرده است. ایده اصلی اجرای آنالیز ورودی و خروجی بر روی قطعه کد داده‌شده است. شبکه عصبی آموزش‌دیده، هرس شده و قوانین از شبکه هرس شده استخراج می‌شوند. ورودی‌ها برای یک خروجی خاص شناسایی می‌شوند. موارد آزمون شامل خصوصیات نامربوط از یک مجموعه ترکیبی موارد آزمون هستند که بتوانند یک خروجی خاص را حذف کنند. به‌عنوان نتیجه یک مجموعه کوچک از موارد آزمون، برای اجرا از منابع آزمون کمتری استفاده می‌کند. این روش به چهار مرحله متمایز: ۱– ساختار و آموزش شبکه عصبی۲– هرس و لیست ویژگی‌ها ۳– استخراج قوانین ۴– تولید موارد آزمون تقسیم‌شده است. الگوریتم استخراج قوانین و الگوریتم هرس برای آنالیز ورودی خروجی از یک برنامه آزمون‌شده اجراشده‌اند. این روش بر روی برنامه‌های تجاری چند خروجی اجراشده که به‌صورت خودکار بیشتر ورودی‌های مؤثر بر خروجی‌های کاربردی را شناسایی می‌کند. مزیت دیگر این روش، تعیین ارزیابی رده‌ها برای خصوصیات جزئی و به دست آوردن لیست منظمی از ویژگی‌هاست.

Khan و همکاران [۴۳] تأثیر کاهش موارد آزمون و اولویت‌بندی در فرایند آزمون نرم افزار را با استفاده از مطالعات تجربیات قبلی بررسی کردند. نتایج نشان داد که این تکنیک‌ها به‌طور قابل‌ملاحظه‌ای باعث بهبود اثربخشی آزمون می‌شود و این روش‌ها پیشرفت مؤثر آزمون را نشان می‌دهند و ترکیبات مختلف مفیدی ازاین‌روها برای حالات مختلف آزمون مفید هستند.

Islam و همکاران [۳۰] رویکرد اولویت‌بندی چندهدفه موارد آزمون بر اساس اندیس گذاری معنایی نهان پیشنهاد کردند. اندیس گذاری معنایی نهان، به معنی روش بازیابی اطلاعات (IR) می‌باشد. این تکنیک پیشنهادی پوشش کد، نیازمندی‌های نرم‌افزار و هزینه اجرایی موارد آزمون را شمارش می‌کند. رویکرد بازیابی قابلیت رهگیری مبتنی بر IR[۱] اعمال می‌شود تا مصنوعات نرم‌افزار را (برای مثال مشخصات نیازمندی‌ها) به کد پیوند دهد. ترتیب موارد آزمون با استفاده از بهینه‌سازی چندمنظوره تعیین و بر اساس الگوریتم NSGA-II[۲] اجرا می‌شوند. این رویکرد با دو نرم‌افزار کوچک جاوا تحت ارزیابی قرارگرفته است.

Gupta و همکاران [۳] یک روش برای تولید و کاهش موارد آزمون با استفاده از جدول تصمیم، شامل نیازهای نرم‌افزار و خصوصیات تعریف‌شده توسط کاربرد ارائه کردند. این روش نیازمندی‌های زبان طبیعی را به‌عنوان ورودی می‌گیرند، شرایط یا عملیات لازم را از این نیازمندی‌ها استخراج می‌کنند. شرایط یا عملیات برای طراحی جدول تصمیم استفاده می‌شوند. از جدول تصمیم برای تعیین و استخراج قوانین و تولید موارد آزمون استفاده می‌شود. سپس داده‌های تعیین‌شده آزمون و موارد آزمون تولیدشده، بر روی کد برای آزمون واحد که در مرحله اجرا تولیدشده، اجرا می‌شوند و درنهایت خروجی آزمون تولید می‌شود.

---

[۱] IR-based traceability recovery approach

[۲] Non-dominated Sorting Genetic Algorithm II

این روش به آزمونگر اجازه می‌دهد یک برآورد از خطاها به دست آورد بدون اینکه آزمونگر اطلاعی از کد نویسی یا منطق برنامه‌نویسی داشته باشد. بنابراین تمام هزینه و زمان آزمون را کاهش می‌دهد.

Wang و همکاران [۴۷] رویکرد اولویت‌بندی پویا موارد آزمون بر اساس معیارهای چندگانه ارائه دادند. در این روش پیشنهادی ارزش‌های اولویت‌بندی موارد آزمون به‌صورت جداگانه و بر اساس پنج معیار پوشش، احتمال بالقوه‌ی در معرض خطا قرارگرفته، نیازمندی‌ها، اطلاعات تاریخچه و زمان محاسبه می‌شوند و مجموع وزنی نتایج بهینه اندازه‌گیری می‌شود. سپس مجموعه آزمون بر اساس ارزش محاسبه‌شده توسط مجموع وزنی مرتب می‌شود. این رویکرد با نمونه کوچک موارد آزمون بررسی‌شده است. این رویکرد پیشنهادی پیچیده بوده و در طول اولویت‌بندی موارد آزمون زمان زیادی را صرف می‌کند.

Mohapatra و همکاران [۴۲] یک الگوریتم برای کاهش موارد آزمون ارائه کردند که مجموعه نماینده مورد آزمون[۱] از مجموعه موارد آزمون را معیار آزمون در نظر گرفتند. این الگوریتم با استفاده از الگوریتم ژنتیک (GA) با تغییر طول کروموزوم[۲] و با یافتن مجموعه نماینده مورد آزمون، موارد آزمون را کاهش می‌دهد. درنتیجه هزینه آزمون رگرسیون را کاهش می‌دهد و بهبود بهره‌وری نرم‌افزار، با مجموعه آزمون بهینه‌شده است.

Acharya و همکاران [۱] از داده‌های تاریخی روی خطاهایی که قبلاً رخداده‌اند برای اولویت‌بندی موارد آزمون استفاده کرده‌اند، فعالیت‌های سیستم را با استفاده از نمودار فعالیت و گراف فعالیت مدل کردند. گره‌های تحت تأثیر با استفاده از الگوریتم Forward slicing به دست آمدند و الگوهای مکرر گراف توسط قوانین وابستگی استخراج می‌شوند، که به صور مؤثری در اولویت‌بندی کمک کرده است و مجموعه آزمون[۳] اولویت‌بندی شده در این روش ارزش APFD[۴] بالایی در مقایسه با موارد آزمون اولویت‌بندی نشده دارد و ارزش APFD بر نیازمندی‌های تابعی سیستم نشان داده که یک تغییر در مرحله تعمیر و نگهداری بر رفتارهای غیر تابعی سیستم نیز تأثیر می‌گذارد.

Saini و همکاران [۴] الگوریتم اولویت‌بندی موارد آزمون (MTCPA[۵]) را پیشنهاد کردند که بر اساس دو تابع می‌باشد. توابع شامل پوشش دستورالعمل و زمان اجرایی موارد آزمون با استفاده از الگوریتم ژنتیک می‌باشد. روش پیشنهادی با تکنیک‌های مختلف اولویت‌بندی موجود به‌منظور پیدا کردن جواب بهینه مقایسه شده است. در نتایج تجربی مربوط به این روش نشان داده‌شده است که الگوریتم پیشنهادی، دنباله‌ای از موارد آزمون با حداکثر پوشش خطا و حداقل زمان اجرایی همراه با معیار میانگین درصد شناسایی خطا (APFD) حداکثر به‌عنوان جواب می‌دهد.

Wang و همکاران [۴۱] اثرات پروفایل‌های مختلف در کاهش مورد آزمون رگرسیون ارائه کردند. در این روش با تجزیه‌وتحلیل خوشه‌ای از سه نوع مختلف پروفایل‌های ساختاری شامل توالی اجرای تابع (FES[۶])، توالی فراخوانی تابع (FCS[۷]) و درخت فراخوانی تابع (FCT[۸]) ارائه کردند و همچنین پنج برنامه متوسط برای اعتبار اثربخشی پروفایل‌های مختلف در کاهش مورد آزمون رگرسیون طراحی و انجام داده‌اند.

Shahid و Ibrahim[۴۴] روش جدیدی برای تعیین اولویت‌بندی آزمون نرم افزار مبتنی بر کد پیشنهاد دادند. رویکرد ارائه‌شده، موارد آزمون نرم افزار را بر اساس کد تحت پوشش آزمون اولویت‌بندی می‌کند. موارد آزمون که حداکثر روش‌ها را پوشش می‌دهند؛ بیشترین احتمال تشخیص خطا را دارا هستند. Abdullah و همکاران [۵] یافته‌های بررسی سیستماتیک

[1] Representative set of test cases
[2] Varying chromosome lengh
[3] Test Suit
[4] Average Percent Fault Detection (APFD)
[5] Multi-Objective test case prioritization algorithm (MTCPA)
[6] Function Execution Sequence (FES)
[7] Call Sequence (FCS)
[8] Function Call Tree (FCT)

انجام‌شـده برای جمع‌آوری شـواهد برآورد پایایی طراحی شـیءگرا را ارائه دادند. آن‌ها نتیجه گرفتند که آزمون پذیری عاملی است که پیش‌بینی می‌کند چه مقدار تلاش برای آزمون نرم افزار لازم است.

Huda و همکاران [۲۴] یک مدل مؤثر اندازه‌گیری طراحی شـیءگرا پیشـنهاد کردند. مدل پیشنهادی آن‌ها از روش رگرسـیون خطی چندگانه بین عوامل مؤثر و معیارها بهره می‌گیرد. اطلاعات سـاختاری و عملکردی نرم‌افزار شـیءگرا برای ارزیابی اثربخشی عوامل مورداستفاده قرارگرفته است. این مدل بین اثربخشی و سازه‌های طراحی شیءگرا ارتباط برقرارمی‌کند. توانایی تعیین کمیت مدل ازلحاظ تجربی مورد تائید قرار گرفت.

Chikarara و Chhillar[۱۴] مجموعه‌ای از معیارها را ارائه دادند. معیارهای ارائه‌شـده برای دسـتور برنامه‌ها مبتنی بر مقادیر پیچیدگی آن‌ها به کار رفت. آن‌ها نتیجه گرفتند که باید بین ویژگی‌های داخلی نرم‌افزار برای ایجاد درجه بالاتر قابلیت استفاده مجدد، سازگاری وجود داشته باشد.

Patwa و همکاران [۳۶] عوامل فاز برنامه‌نویسـی را ارائه کردند که بر آزمون نرم افزار شـیءگرا تأثیر می‌گذارد. این عوامل عبارت‌اند از مهارت برنامه‌نویس و آزمونر، برنامه‌نویس و آزمونر سـازمان، اندازه گروه توسـعه، کاربار برنامه[1] (استرس)، دانش دامنه و ماهیت انسانی (اشتباه یا از قلم انداختن). تجزیه‌وتحلیل عوامل و مکان این عوامل با توجه به تأثیر آن‌ها بر نرم‌افزار، با استفاده از روش وزن نسبی و آزمون آنواَ انجام شد.

Brunt ink و همکاران [۹] رابطه بین رده‌ها و موارد آزمون JUnit آن‌ها را تحلیل کردند. نتایج مطالعه آن‌ها حاکی از وجود همبسـتگی معناداری بین معیارهای سـطح- کلاس و معیارهای سـطح- آزمون اسـت. آن‌ها همچنین در مورد اینکه چگونه معیارهای مختلف می‌توانند در آزمون‌پذیری نقش داشـته باشـند، بحث کرد. آزمایشـات با اسـتفاده از GQM و MEDEA انجام شدند. نتایج با استفاده از ضریب همبستگی رتبه‌ای اسپیرمن، ارزیابی‌شده است.

Malviya و Singh[۳۱] مشاهداتی در مورد مدل تخمین قابلیت نگهداری برای نرم‌افزار شیءگرا در مرحله‌های ضروری، طراحی، برنامه‌نویسـی و آزمون ارائه کردند. کار آن‌ها در مورد افزایش قابلیت‌های تعمیر و نگهداری شـاخص‌های MOOD است.

Hao و همکاران [۲۱] یک رویکرد اولویت‌بندی یکپارچه مورد آزمون ارائه کردند. رویکرد پیشـنهادی آن‌ها شـامل دو مدل اسـت. آن‌ها نشـان دادند که طیف وسـیعی از تکنیک‌های اولویت‌بندی آزمون وجود دارد. این طیف توسط مدل ساخته‌شده و بین تکنیک‌ها با اسـتفاده از استراتژی‌های صرفاً کلی یا صرفاً اضافی، مستقر می‌باشد. آن‌ها مواردی را پیشنهاد دادند تا امکان اسـتفاده از احتمالاتی که موارد آزمون قادر به تشـخیص اشتباهات برای روش‌ها باشند و از اطلاعات پوشش دینامیک به‌جای اطلاعات پوشش استاتیک استفاده کنند، فراهم آید.

۴-روش‌های آزمون نرم افزار

٤-۱- آزمون رگرسیون

آزمون رگرسـیون، روند بازرسـی مجدد قسـمت‌های اصـلاح‌شده و آسیب‌دیده‌ی نرم‌افزار است و اطمینان از اینکه هیچ‌کدام از خطاهای جدید در کد قبلی قبلاً معرفی نشده باشند [۴۵]. بنابراین، پس از اضافه کردن برخی ویژگی‌ها یا انجام برخی تغییرات در نرم‌افزار، محصـول اصـلاح‌شـده خطای commission اسـت، و سـپس انجام آزمون رگرسیون ضروری است. در آزمون رگرسـیون، وجود test suite ها و test plan ها مفید هسـتند. در این روش، آزمون تنها روی اجزای اصـلاح‌شـده یا تحت تأثیر انجام می‌شود، نه کل نرم‌افزار. در انجام آزمون رگرسیون محدودیتی وجود ندارد به همین خاطر می‌توان این آزمون را هر تعداد دفعه که لازم باشد انجام داد [۳۸ و ۴۵]. اما schedule هیچ زمانی برای آزمون رگرسیون نمی‌دهد چراکه آن را تحت محدودیت‌های زمانی بزرگ‌تر انجام می‌دهد.

[1] Program workload

آزمون نرم افزار یک رویکرد مهم برای اطمینان از کیفیت و قابلیت اطمینان نرم‌افزار است. از زمان توسعه فناوری شیءگرا (OO[1])، نرم‌افزار شیءگرا (OOS[2]) به‌طور گسترده‌ای مورداستفاده قرارگرفته است. بااین‌حال، با توجه به ویژگی‌های خاص زبان نرم‌افزارهای شیءگرا مانند خلاصه‌سازی، ارث و پلی مورفیسم، آزمونرها ممکن است با چالش‌هایی مواجه شوند و این مسئله باعث شود که از رویکرد آزمون نرم افزارهای سنتی استفاده کنند [۳۷،۴۰،۱۳]. بسیاری از روش‌های آزمون نرم افزار شیءگرا موردبررسی قرارگرفته‌اند، ازجمله آزمون تصادفی[3] [۶]، آزمون‌های مبتنی بر حالت[4] [۲۳] و آزمون‌های مبتنی بر توالی[5] [۲۵]. در بین این رویکردها، آزمون تصادفی اغلب به دلیل سادگی در صنعت به‌کاررفته است [۲۰و۱۱]. سایر روش‌های آزمون به‌طورکلی نیاز به مهارت‌های آزمون حرفه‌ای‌تر دارند و اغلب روی برخی از انواع نرم‌افزارهای خاص تمرکز می‌کنند. یک مسئله در تکامل نرم‌افزارهای شیءگرا این است که موارد آزمون تولیدشده توسط این روش‌های آزمون معمولاً تعداد بسیار زیادی از موارد آزمون را دربر می‌گیرند و ازاین‌رو اجرای هر یک از آن‌ها ممکن است هزینه بسیار بالایی داشته باشد [۳۳،۲۷،۱۵].

۴-۲- تکنیک‌های آزمون شیءگرایی

اصول شیءگرایی برمبنای مفاهیمی همچون Object , Class , Message , Interfaces, Inheritance, Polymorphism و غیره می‌باشد. آزمون‌های سنتی با استفاده از روش‌های زیر قابل‌استفاده در شیءگرایی می‌باشند[۱]:

٤-۲-۱- Method testing

در این روش هر متد توسط برنامه‌نویس آزمون می‌شود و اطمینان حاصل می‌شود که تمام حالات در نظر گرفته‌شده است.

٤-۲-۲- Class testing

در اینجا تمرکز بر روی متدهای یک کلاس است که به روش‌های زیر انجام می‌گیرد:

- آزمون هر متد در یک کلاس
- آزمون خصوصیات کلاس بین متدها

٤-۲-۳- Interaction testing

آزمون ادغام می‌تواند با استراتژی‌های مختلف انجام گیرد:

Thread-based testing: کلاس‌های موردنیاز برای پاسخ به یک ورودی یا رخداد را ادغام می‌کند.

Use-based testing: کلاس‌های موردنیاز را برای یک Use Case ادغام می‌کند.

Cluster testing: کلاس‌های موردنیاز را برای یک همکاری را ادغام می‌کند.

٤-۲-٤- System testing

تمام روش‌های System test در روش‌های سنتی قابل‌استفاده برای شیءگرایی نیز می‌باشند. برخی از این آزمون‌ها عبارت‌اند از: Performance testing، Stress testing، Security testing، Recovery testing.

٤-۲-٥- Acceptance testing

آزمون محیط تولید و آزمون محیط مصرف‌کننده می‌باشند.

۵- اولویت‌بندی موارد آزمون

[1] Object-oriented
[2] Object-oriented software
[3] Random testing
[4] State-based testing
[5] Sequence-based testing

اولویت‌بندی موارد، آزمون برنامه‌ریزی موارد آزمون در مواردی است که خطاها را بسیار سریع و راحت و در دستوری خاص به‌جای برخی از موارد تصادفی تشخیص می‌دهند [۳۲،۳۸،۴۶]. همان‌طور که شناخته‌شده است، چنین بودجه و زمان مشخصی برای آزمون رگرسیون وجود ندارد، بنابراین هدف یافتن حداکثر خطا در حداقل زمان است؛ برای این آرایش موارد آزمون یا test suite ها در برخی موارد محدود هستند. این حالت حتی برخی از معیارها مانند شناسایی خطا و پوشش کد را مطابق با موارد آزمون اولویت‌بندی شده، تعریف می‌کند.

برای بهبود کار آیی آزمون نرم افزار شیءگرا در آزمون رگرسیون، لازم است تا موارد آزمون را اولویت‌بندی کنیم تا خطاها را با سرعت بیشتری پیدا کنیم. به‌طورکلی، ازآنجاکه تنها برخی از ورودی‌های آزمون می‌توانند خطاها را شناسایی کنند، درصورتی‌که این ورودی‌های خاص می‌توانند برای اجرای زودهنگام اولویت‌بندی شوند. سپس راندمان آزمون می‌تواند تا حد زیادی بهبود یابد. این نوع اولویت‌بندی مورد آزمون (TCP[1]) باید امکان تشخیص خطاها را فراهم کند [۱۸].

تکنیک‌های اولویت‌بندی مورد آزمون در حال حاضر مبتنی بر اطلاعات جعبه سفید[2] یا جعبه سیاه[3] توسعه‌یافته‌اند [۴۹]. اطلاعات جعبه سفید اغلب شامل پوشش سورس کد برنامه، یک مدل برنامه و سابقه تشخیص خطا هستند؛ اما اطلاعات جعبه سیاه معمولاً شامل اطلاعات ورودی آزمون هستند. در آزمون رگرسیون اطلاعات جعبه سفید معمولاً بر اساس نسخه‌های قبلی برنامه است، اما آزمون روی نسخه فعلی انجام می‌شود [۲۹]. تکنیک‌های اولویت‌بندی مورد آزمون با استفاده از اطلاعات جعبه سیاه این مشکل را ندارند.

نمونه‌گیری تصادفی[4] یک روش اولویت‌بندی جعبه سیاه است و معمولاً به‌عنوان معیار ارزیابی اثربخشی سایر تکنیک‌های اولویت‌بندی مورداستفاده قرارمی‌گیرد. به‌منظور بهبود کار آیی توالی‌های تصادفی و ارزیابی بهتر اولویت‌بندی در آزمون رگرسیون، تحقیقاتی درزمینه‌ی روش اولویت‌بندی با استفاده از توالی‌های تصادفی تطبیقی (ARS[5]) انجام‌شده است [۴۸و۴۹]. ARS ها را می‌توان به‌عنوان توالی تصادفی جایگزین[6] در نظر گرفت که در آن موارد آزمون به‌منظور بهبود عملکرد توالی تصادفی در دامنه ورودی به‌طور یکنواخت گسترش می‌یابند. ARS ها از مفهوم آزمون تصادفی تطبیقی [۷،۱۰،۱۲،۱۳] حاصل‌شده‌اند که نسخه پیشرفته‌ی آزمون تصادفی است و تلاش می‌کند تا اثربخشی تشخیص خطای آزمون تصادفی را با انتشار برابر ورودی‌های آزمون در کل دامنه ورودی گسترش دهد. نمونه‌گیری تصادفی تطبیقی قادر به ایجاد ARSها برای انتخاب موارد آزمون به‌صورت متنوع در دامنه ورودی تا حد امکان است [۲۸]. ARS ها برای نرم‌افزار پردازش گرا به اولویت‌بندی مورد آزمون مبتنی بر تکنیک‌های ART اعمال‌شده‌اند [۷،۱۰،۴۸،۴۹].

۶- تکنیک‌های اولویت‌بندی و بهینه‌سازی موارد آزمون

ازآنجایی‌که قسمت عمده‌ای از هزینه و زمان تولید و نگهداری نرم‌افزارها صرف آزمون نرم افزارها می‌گردد، یافتن روش‌های بهینه‌تر و کارآمدتر برای آزمون نرم افزارها بسیار ضروری می‌نماید. ازاین‌رو در چند دهه اخیر، پژوهش‌های زیادی درزمینه‌ی ارائه‌ی روش‌ها و ابزارهای گوناگون برای آزمون انجام‌شده است و محققان سناریوهای مختلفی برای اولویت‌بندی و کاهش موارد آزمون در آزمون رگرسیون نرم‌افزار ارائه کرده‌اند [۲۹].

در طول مرحله توسعه و نگهداری نرم‌افزار، زمانی که موارد آزمون جدید هنگام ویرایش نرم‌افزار تولید می‌شوند، اندازه مجموعه آزمون ممکن است افزایش یابد. مجموعه‌های آزمون ، اغلب به‌اندازه‌ای می‌رسند که بیشتر هزینه‌ها برای اجرای مجموعه آزمون اختصاص داده می‌شود [۱۶]. اجرای کل مجموعه آزمون زمان‌بر و هزینه‌بر است؛ بنابراین روش‌هایی مانند

[1] test case prioritization
[2] White-box
[3] Black-box
[4] Random sampling
[5] Adaptive Random Sequences
[6] Alternative random sequence

انتخاب موارد آزمون[1]، اولویت‌بندی[2]، کاهش مجموعه موارد آزمون[3]، انجام برخی از معیارهای انتخاب، پوشش همه‌ی خطاهای احتمالی در حداقل زمان و... اعمال شود [۳۹].

اولویت‌بندی موارد آزمون، فرایند بسیار مفید و ضروری است، چراکه ما موارد آزمون را در دستور مبتنی بر اولویت مرتب می‌کنیم، که خروجی و نتایج آزمون‌های کارآمدی را به ما می‌دهد [۳۸]. روش‌هایی برای اولویت‌بندی مورد آزمون وجود دارد و یکی از آن‌ها رویکرد حریصانه است که در آن برخی از معیارها ثابت هستند و عنصر با حداکثر وزن انتخاب می‌شود. اشکال عمده‌ی رویکرد حریصانه این است که راه‌حل بهینه محلی به‌جای بهینه جهانی در مسئله در نظر گرفته‌شده داده می‌شود. جدا از این، مسئله‌ی دیگر این است که الگوریتم ژنتیک اضافی یا دو الگوریتم بهینه وجود دارد؛ این همانند حریص است اما از استراتژی‌های مختلفی استفاده می‌کند.

حتی تکنیک‌های جستجوی فرا ابتکاری دریافتن راه‌حل برای یک مسئله‌ی خاص با هزینه محاسباتی منطقی استفاده می‌شوند. بنابراین برخی از تکنیک‌های اولویت‌بندی در سناریوی موازی برای چندین صف[4] (PSMQ) [46]، بهینه‌ساز ازدحام ذرات چند هدفه برای بهینه‌سازی مورد آزمون، الگوریتم بهینه‌سازی کلنی مورچگان (ACO[5]) برای پوشش خطا استفاده‌شده است و از اولویت‌بندی مورد آزمون رگرسیون [۱۷]، الگوریتم ژنتیک و الگوریتم حریصانه، اولویت‌بندی آزمون چند هدفه (MOTCP[6]) [26]، بهینه‌سازی آزمون رگرسیون چند هدفه (MORTO[7]) [34]، آزمون اهمیت رویکرد ماژول (TIM[8]) استفاده‌شده است. معیارهای به‌کاررفته در روش‌های آزمون اولویت‌بندی برای محاسبه بهره‌وری استفاده می‌شوند. بعضی از آن‌ها درصد متوسط پوشش بلوک (APBC[9])، درصد متوسط پوشش طراحی (APDC[10])، درصد متوسط خطاهای شناسایی‌شده (APFD[11])، درصد متوسط پوشش دستور (APSC[12]) و درصد متوسط ماژول‌های آسیب‌دیده بر اساس خطا در هر مورد آزمون (APMC[13]) هستند [۱۲].

۷- انواع رویکردهای اولویت‌بندی

انواع رویکرد های اولویت بندی به شرح زیر می باشد :

۷-۱- رویکرد مبتنی بر تغییرات و تاریخچه

داده‌های تاریخچه‌ای مانند تاریخچه اجرایی یا تغییر اطلاعات موجود در این رویکرد مورداستفاده قرارمی‌گیرند.

- روش پوشش جدید: بخش‌هایی از کد که تغییر کرده‌اند شناسایی می‌شوند و پوشش‌ها فقط روی همان‌ها حساب می‌شوند.
- روش تعمیم‌یافته: بخش‌هایی از کد که در هر کامیت ها به‌صورت هم‌زمان تغییر کرده‌اند، شناسایی می‌شوند. سپس پوشش نه فقط روی بخش‌ها تغییریافته، بلکه روی کدهای مرتبط با آن‌ها حساب می‌شود.

---

[1] Select Test Case
[2] Prioritization
[3] Test Case Minimized
[4] Parallel scenario for multiple queues
[5] Ant colony optimization algorithm
[6] Multi-objective test case prioritization
[7] Multi-objective regression test optimization
[8] Testing importance of module approach
[9] Average percentage block coverage
[10] Average percentage decision coverage
[11] Average percentage of faults detected
[12] Average percentage statement coverage
[13] Average percentage-fault-affected module

- رویکرد سابقه آزمون: نمونه آزمون‌هایی که در اولویت‌بندی قبل، اولویت بالاتری داشته‌اند در این دور آزمون نیز دارای اولویت بیشتری هستند [۲۵].

۷-۲- رویکرد مبتنی بر نیازمندی

از ویژگی و خصوصیات نیازمندی‌ها استفاده می‌شود. نیازمندی‌های نرم‌افزار کار استخراج، تحلیل، تعیین و اعتبارسنجی نیازمندی‌ها برای نرم‌افزار را دارد.

۷-۳- رویکرد مبتنی بر مدل

مدل‌های مختلفی ازجمله نمودارهای توالی و یا UML نمودار فعالیت به‌جای بلاک‌های کد استفاده می‌شود. هدف UML نمایش انواع مختلف نمودارهای بصری به‌قصد نمایش جنبه‌های مختلف سیستم است. نمودار فعالیت برای توصیف قدم‌به‌قدم گردش کار تجاری و عملیاتی مؤلفه‌های سیستم استفاده می‌شود [۳۳].

جدول ۱- مرور روش های اولویت بندی و تست نرم افزار

عنوان	پارامترهای مورد استفاده	ملاک ارزیابی	الگوریتم های استفاده شده	نتایج
تولید و کاهش موارد آزمون با تجزیه و تحلیل ورودی و خروجی خودکار ۲۰۰۳ [۳۵].	لیست منظم شده از ویژگی ها و کلاسهای هم ارزی برای خصوصیات ورودی کد- روابط ورودی و خروجی	ارزیابی توصیفی	الگوریتم استخراج قوانین، شبکه عصبی	صرفه جویی در منابع تست
تأثیر تست نرم افزار بر اولویت بندی و کاهش موارد آزمون ۲۰۰۹ [۴۳].	تاریخچه موارد آزمون، پوشش	درصد تشخیص خطا، درصد اندازه کاهش	-	بهبود اثربخشی تست
تکنیک چندهدفه برای اولویت بندی موارد آزمون براساس اندیس گذاری معنایی نهان ۲۰۱۲ [۳۰].	پوشش نیازمندیها و پوشش کد و هزینه اجرا	میانگین درصد شناسایی خطا (APFD)	الگوریتم ژنتیک	عملکرد بهتر تکنیک های پایه
کاهش موارد آزمون مبتنی بر قواعین جدول تصمیم ۲۰۱۴ [۳].	نیازمندیهای تابعی	مقایسه با دیگر روش های ایجاد جدول تصمیم، خصوصیات نیازها	-	بیشتر از ۳۳ درصد موارد آزمون کاهش یافته، صرفه جویی در هزینه و زمان
یافتن مجموعه نماینده از مجموعه تست جهت کاهش موارد آزمون در آزمون رگرسیون ۲۰۱۵ [۴۲].	ماتریس نیازمندیهای تست (TR)، پوشش، تابع	ارزش تابع تناسب کروموزوم F(x)، هزینه بهینه، اندازه مجموعه موارد آزمون	الگوریتم ژنتیک	کاهش هزینه و بهبود بهره وری نرم افزار با مجموعه آزمون، بهینه شده است.
اولویت بندی مورد آزمون مبتنی بر مدل با استفاده از	تاریخچه موارد آزمون، نمودار فعالیت (AD) و گراف (GD)	میانگین درصد شناسایی خطا (APFD)	الگوریتم برش	این روش در درصد تشخیص خطا نتایج بهتری می دهد.

استخراج قانون وابستگی ۲۰۱۵ [۱]				
الگوریتم اولویت بندی موارد آزمون چندهدفه با الگوریتم ژنتیک ۲۰۱۵ [٤]	**تابع پوشش دستورالعمل اجرایی و تابع زمان اجرایی**	**میانگین درصد شناسایی خطا (APFD)**	**الگوریتم ژنتیک**	**این روش موثرتر از تکنیک های دیگر است.**
اثرات پروفایل های مختلف در کاهش موارد آزمون رگرسیون ۲۰۱۵ [٤١]	**تاریخچه موارد آزمون (اطلاعاتی در مورد ساختار و روابط) و تابع های FCS،FCT،FES**	**درصد موثر تشخیص خطا، درصد کاهش اندازه موارد آزمون، فاصله خوشه ها**	**الگوریتم خوشه بندی K-mean**	**آنالیز خوشه ای FCS با استراتژی حداقل و حداکثر فاصله تشابه ساختاری نسبت به الگوریتم های پوشش مبتنی بر تابع می تواند خطاهای بیشتری را کشف کند.**

۸–نتیجه‌گیری

آزمون نرم افزار یا آزمایش نرم‌افزار به‌منظور کشف خطاهای نرم‌افزار پیش از تحویل نهایی آن به کار می‌رود. آزمون نرم افزار فرآیندی پیچیده است و بیش از نیمی از بودجه پروژه را به خود اختصاص می‌دهد. به‌منظور کاهش هزینه‌ها و افزایش تعداد خطاهای کشف‌شده امروزه بحثی به نام آزمون خودکار نرم‌افزار معرفی‌شده است که در آن تلاش می‌شود فرآیند آزمون نرم افزار به‌صورت خودکار و به کمک الگوریتم‌های تکاملی مانند الگوریتم ژنتیک، سردسازی شبیه‌سازی‌شده، جستجوی ممنوعه، سردسازی کلونی مورچگان و یا روش‌های یادگیری ماشین مانند شبکه‌های عصبی، الگوریتم‌های فازی و غیره انجام شود. استفاده ازاین‌روش‌ها به میزان زیادی کار را راحت می‌کند، اما مسئله دیگر این است که ممکن است زمان اجرای کار زیاد شود، چراکه این الگوریتم‌ها باید تمامی حالت‌های ممکن را آزمایش کنند. به همین خاطر باید آزمون کیس ها را اولویت‌بندی کرد و کمترین تعداد آزمون کیس ها را که بیشترین تعداد خطاها را کشف می‌کنند انتخاب کرد.

تقدیر وتشکر:

با تشکر و سپاس از استاد دانشمند و پر مایه ام سرکار خانم دکتر آجودانیان که از محضر پر فیض تدریس شان ، بهره ها برده ام.

۱۰–مراجع

1. Acharya, P. Mahali and D. P. Mohapatra, **Model Based Test Case Prioritization Using Association Rule Mining**, Springer Smart Innovation, Systems and Technologies, vol. 3, no. 33, pp. 429-440, 2015.
2. Bertolino, **Software testing research: Achievements, challenges, dreams**, in 2007 Future of Software Engineering, 2007, pp. 85-103.
3. Gupta, N. Mishra and D. S. Kushwaha, **Rule Based Test Case Reduction Technique using Decision Table**, in IEEE International Advance Computing Conference (IACC), Allahabad,India, 2014.
4. Saini and D. Tyagi, **MTCPA: Multi-Objective Test Case Prioritization Algorithm Using Genetic Algorithm**, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 6, pp. 1093-1097 2015.
5. Abdullah, D., Srivastava, R., Khan, M.H.: **Testability estimation of bject oriented design: a revisit**. Int. J. Adv. Res. Comput. Commun. Eng. 2(8) (2013).

6. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.j., Mcminn, P., Bertolino, A., 2013. **An orchestrated survey of methodologies for automated software test case generation**. J. Syst. Softw. 86 (8), 1978-2001.
7. Barus, A.C., Chen, T.Y., Kuo. F.C .. Liu. H., Merkel. R., Rothermel, G., 2016. **A cost-effective random testing method for programs with non-numeric inputs**. IEEE Trans. Comput. 65 (12), 3509-3523.
8. Binder, R.V .. 1996. **Testing object-oriented software: a survey**. Softw. Test. Verification Reliab. 6 (3), 125-252.
9. Bruntink, M., Van Deursen **A.: Predicting class testability using object oriented metrics**. In: 4th IEEE International Workshop on Source Code Analysis and Manipulation, pp. 136-145. Chicago, lL, US (2004) .
10. Chen, T.Y .. Kuo, F.C .. Merkel, R.G., Tse, T.H., 2010. **Adaptive random testing: the art of test case diversity**. J. Syst. Softw. 83 (1). 60-66.
11. Chen, T.Y., Fei-Ching, K., Dave, T., Quan, Z.Z., 2015. **A revisit of three studies related to random testing**. Sci. China Inf. Sci. 58 (5), 1-9.
12. Chen, T.Y., Kuo, F.C., Liu, H., Wong, W.E., 2013. **Code coverage of adaptive random testing**. IEEE Trans. Reliab. 62 (1), 226-237.
13. Chen, T.Y., Leung, H., Mak, I.I., 2004. **Adaptive random testing**. In: Proceedings of the 9th Asian Computing Science Conference (ASIAN 2004), Thailand. Springer LNCS, pp. 320-329.
14. Chhikara, A., Chhillar, R.S.: **Analyzing complexity of java program using object oriented software metrics.** IJCSI Int. J. Comput. Sci. 9(1),3 (2012). ISSN:1694-0814 .
15. Ciupa, I., Leitner, A., Oriol, M., Meyer, B., 2008. **ARTOO: adaptive random testing for object-oriented software**. In: ACM/IEEE 30th International Conference on Software Engineering (ICSE 2008), IEEE, New York, USA, pp. 71-80.
16. D. Jeffrey and N. Gupta, **Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction**, TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 33, no. 2, pp. 108-123, 2007.
17. Elanthiraiyan, N., Arumugam, C.: **Parallelized ACO algorithm for regression testing prioritization in hadoop framework.** In: International Conference on Advanced Communication Control and Computing Technologies (TCACCCT), pp. 1568-1571. IEEE (2014) .
18. Elbaum, S., Malishevsky, A.G., Rothermel, G., 2002. **Test case prioritization: a family of empirical studies**. IEEE Trans. Softw. Eng. 28 (2), 159-182.
19. G. Suganya and S. Neduncheliyan, **A study of Object Oriented testing techniques: Survey and Challenges**, in Innovative Computing Technologies (ICICT), 2010 International Conference on, 2010, pp. 1-5.
20. Hamlet, R., 2002. **Random testing. Encyclopedia of Software Engineering**. John Wiley and Sons. Harman, M., McMinn, P., Souza, J . Yoo, S .. 2012. Search based software engineering: techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification, pp. 1-59.
21. Hao, D., Zhang, L., Zhang, L., Rothermel, G., Mei, H.: **A unified test case prioritization approach**. In: ACM Trans. Softw. Eng. Methodol. 9(4) (2010).
22. Hla, K.H.S., Choi, Y. Park, J.S. **Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting**. In: 8th International Conference on Computer and Information Technology Workshops, pp. 527-532. IEEE (2008).
23. Holt, N.E., Briand, LC., Torkar, R., 2014. **Empirical evaluations on the cost-effectiveness of state-based testing: an industrial case study**. Inf. Softw. Technol. 56 (8), 890-910.
24. Huda, M., Arya, Y.D.S., Khan, M.H.: **Evaluating effectiveness factor of object oriented design: a testability perspective**. Int. J. Softw. Eng. Appl. (IJSEA) 6(1), 41 (2015).
25. Hui, S.U., Zhang, Y., Yao, H .. Fei, R .. 2005. **Object-oriented software cluster-level testing based on uml sequence diagram.** Comput. Eng. 31 (24), 78-80.
26. Islam, M.M., Scanniello, G.: **MOTCP: a tool for the prioritization of test cases based on a sorting genetic algorithm and latent semantic indexing**. In: 28th IEEE International Conference on Software Maintenance (ICSM), pp. 654-657. IEEE (2012).
27. Ledru, Y .. Petrenko, A., Boroday, s .. Mandran, N .. 2012. **Prioritizing test cases with string distances**. Automated Softw. Eng. 19 (1). 65-95.
28. Liu, H., Kuo, F.C., Towey, D., Chen, T.Y., 2014. **How effectively does metamorphic testing alleviate the oracle problem?** IEEE Trans. Software Eng. 40 (1). 4-22.

29. Luo, Q .. Moran, K., Poshyvanyk, D., 2016. **A large-scale empirical comparison of static and dynamic test case prioritization techniques.** In: ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2016), ACM, Washington, USA, pp. 559-570.
30. M. M. Islam, A. Marchetto, A. Susi and G. Scanniello, **A Multi-Objective Technique to Prioritize Test Cases Based on Latent Semantic Indexing**, Szeged, 2012.
31. Malviya, A.K., Singh V.: **Some observation on maintainability estimation model for object oriented software in requirement, design, coding and testing phases**. Int. J. Adv. Res. Comput. Sci. Softw. Eng. 5(3) (2015). ISSN: 2277 128X .
32. Nagar, R., Kumar, A., Kumar, S., Baghel, A.S.: **Implementing test case selection and reduction techniques using meta-heuristics**. In: Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference, pp. 837-842. IEEE (2014).
33. Nie, C., Wu, H., Niu, X., Kuo, F.C., Leung, H., Colbourn, C.J.. 2015. **Combinatorial testing, random testing, and adaptive random testing for detecting interaction triggered failures**. Inf. Softw. Technol. 62, 198-213.
34. Noguchi, T., Sato, A.: **History-based test case prioritization for black box testing using ant colony optimization**. In: IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1-2. IEEE (2015).
35. P. Saraph, M. Last and A. Kandel, **Test Case Generation and Reduction by Automated Input-Output Analysis**, 2003.
36. Patwa, S., Malviya A.K.: **Impact of coding phase on object oriented software testing.** Covenanat J. Inf. Commun. Technol. (CJICT) 2(1) (2014).
37. Pezze, M .. Young, M., 2004. **Testing object-oriented software**. In: 26th International Conference on Software Engineering Proceedings (ICSE 2004),United Kingdom. IEEE, pp. 739- 740.
38. Qu, B., Nie, C., Xu, B.: **Test case prioritization for multiple processing queues**. In: ISISE'08 International Symposium on Information Science and Engineering, vol. 2, pp. 646-649. IEEE (2008).
39. R. Nagar, A. Kumar, S. Kumar and A. S. Baghel, **Implementing Test Case Selection and Reduction Techniques using Meta-Heuristics**, in Confluence The Next Generation Information Technology Summit (Confluence), Noida, India, 2014 5th International Conference.
40. R. V. Binder, **Testing object-oriented systems :models, patterns, and tools**: Addison-Wesley Professional, 2000.
41. R. Wang, B. Qu and Y. Lu, **Empirical study of the effects of different profiles on regression test case reduction**, The Institution of Engineering and Technology, vol. 9, no. 2, p. 29–38, 2015.
42. S. K. Mohapatra and M. Pradhan, **Finding Representative Test Suit for Test Case Reduction in Regression Testing**, in IEEE International Conference on Computer, Communication and Control (IC4-2015), Odisha , India, 2015.
43. S. u. R. Khan, I. ur Rehman and S. U. R. Malik, **The Impact of Test Case Reduction and Prioritization on Software Testing Effectiveness**, in International Conference on Emerging Technologies, Islamabad, Pakistan, 2009.
44. Shahid, M., Ibrahim, S.: **A new code based test case prioritization technique.** Int. J. Softw. Eng. Appl. 8(6), 31-38 (2014).
45. Simons, C., Paraiso, E.C.: **Regression test cases prioritization using failure pursuit sampling**. In: 10th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 923-928. IEEE (2010).
46. Tyagi, M., Malhotra, S.: **Test case prioritization using multi objective particle swarm optimizer**. In: International Conference on Signal Propagation and Computer Technology (ICSPCT), pp. 390-395. **IEEE (2014).**
47. X. Wang and . H. Zeng, **Dynamic test case prioritization based on multi-objective**, in Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Las Vegas, NV , 2014.
48. Zhang, X., Chen, T.Y., Liu, H., 2014. **An application of adaptive random sequence in test case prioritization**. In: International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), IEEE, Canada, pp. 126-131.
49. Zhang, X., Xie, X., Chen, T.Y., 2016. **Test case prioritization using adaptive random sequence with category-partition-based distance.** In: IEEE International Confer-ence on Software Quality, Reliability and Security (QRS 2016), IEEE, Washington, USA, pp. 374-385.
50. A Multi-factored Cost- andCode Coverage-Based Test Case Prioritization Technique for Object-Oriented Software **Vedpal and Naresh Chauhan: Elsevier**2019

Pavi Saraswat, Abhishek:A Review of Test Case Prioritization and Optimization Techniques