نشریه علمی پژوهش در ایمنی سلامت و محیط زیست

سال اول، شماره ۳، پاییز ۱۴۰۲: (پیاپی ۳) ۴۳-۵۳

علمی

ارزیابی ریسک کد نرم‌افزار سیستم‌های نهفته مبتنی بر چارجوب سیستم مدیریت ایمنی

علی اصفهانی[۱]، دکتر رضا اصفهانی[۲]، دکتر عبدالرحمن کشوری[۲]

تاریخ دریافت: ۱۴۰۲/۰۴/۱۳

تاریخ پذیرش:۱۴۰۲/۰۶/۲۱

چکیده

امروزه ایمنی در سیستم‌های فناوری اطلاعات اهمیت زیادی دارد و عدم تضمین ایمنی این سیستم‌ها و نقص‌های نرم‌افزاری و همین‌طور خطاها و اشتباهات برنامه‌نویسی می‌تواند باعث خسارت‌های سنگین مالی، روانی و حتی جانی در کشور شود. پیاده‌سازی کد بدون نقص و خطا، تقریباً کاری غیرممکن است، تعداد بسیار کمی از این اشتباهات برنامه‌نویسی به هنگام Compile شدن برنامه مشخص می‌شود؛ اما بیشتر آن‌ها پنهان می‌مانند. نکته حائز اهمیت آن است که بتوان کد پرریسک را قبل از تولید محصول مشخص کرد و مانع از انتشار نسخه جدید شد. در این مقاله با استفاده از ابزار Source Monitor، پیچیدگی کد سیستم‌های نهفته، تعداد خطوط کد و تعداد دستورات در هر تابع مشخص می‌شود. بعد از این مرحله با بهره‌مندی از چارچوب سیستم مدیریت ایمنی یا SMS، ریسک پروژه با استفاده از ماتریس ارزیابی ریسک، آنالیز می‌شود. همچنین خطاهای شایع و پرتکرار در سیستم‌های نهفته (سیستم‌های به زبان C) مورد بررسی قرار می‌گیرند.

کلیدواژه‌ها: ایمنی نرم‌افزار، سیستم‌های نهفته، پیچیدگی کد، چارچوب سیستم مدیریت ایمنی، ماتریس ارزیابی ریسک

۱ کارشناسی ارشد دانشگاه آزاد اسلامی واحد تهران شمال (alisf۲۰۰۰@yahoo.com)

۲استادیار دانشکده و پژوهشکده پدافندغیرعامل، دانشگاه جامع امام حسین (ع)

۱- مقدمه

حفظ ایمنی در سیستم‌های فناوری اطلاعات بسیار حائز اهمیت می‌باشد چرا که می‌تواند باعث جلوگیری از حوادث ناگوار و تلفات سنگین شود. برای مثال در صنعت خودروسازی بسیار مهم است که سیستم‌های نهفته به‌خوبی کار کنند و نرم‌افزار آن‌ها دچار اشکال نشوند چرا که می‌تواند حتی باعث تهدید جان افراد سرنشین شود.

نرم‌افزار سیستم‌های نهفته، دستگاه‌ها و سیستم‌های سخت-افزاری مختلف را مدیریت می‌کند. هدف اصلی نرم‌افزار سیستم‌های نهفته، کنترل عملکرد مجموعه‌ای از دستگاه‌های سخت‌افزاری بدون به خطر انداختن هدف یا کارایی است. در حالت ایده‌آل، این نرم‌افزارها نیازی به ورودی کاربر ندارند و می‌توانند به طور مستقل بر روی پارامترهای از پیش تعیین‌شده عمل کنند. سیستم‌های نهفته به دستگاه‌هایی گفته می‌شود که قرار است یک یا چند کار کاملاً مشخص شده از قبل را انجام دهد. یکی از پرکاربردترین زبان‌های برنامه‌نویسی در سیستم‌های نهفته، زبان C می‌باشد.

امروزه ایمنی نرم‌افزار اهمیت زیادی دارد؛ زیرا اشکالات نرم-افزاری می‌توانند موجب تحمیل خسارت‌های فراوانی شوند. به‌ویژه در سال‌های گذشته نمونه‌های فراوانی از فجایعی که به دلیل اشکالات نرم‌افزاری به بار آمده قابل‌ملاحظه هستند:

- در آوریل ۲۰۱۵ ترمینال بلومبرگ در لندن به‌دلیل مشکل نرم‌افزاری، سیستم از کار افتاد و بیش از ۳۰۰,۰۰۰ معامله‌گر را در بازارهای مالی تحت تاثیر قرار داد. این اتفاق دولت را مجبور کرد تا خرید ۳ میلیارد پوند اوراق را به تاخیر بیاندازد[۲۱].
- در سال ۲۰۱۳ شرکت نیسان مجبور به جمع‌آوری یک میلیون خودرو به خاطر مشکل نرم‌افزاری مربوط به حسگر کیسه هوا شد. گزارش شده بود که دو تصادف به خاطر این مشکل نرم‌افزاری اتفاق افتاده است [۲۲].
- برند استارباکس (شرکت تولید قهوه در سیاتل) مجبور شد به دلیل مشکل نرم‌افزاری در سیستم پایانه فروش خود، ۶۰ درصد فروشگاه‌های خود را در آمریکا و کانادا موقتاً تعطیل کند. در یکی از مغازه‌ها قهوه به‌صورت رایگان سرو شد چون سیستم قادر به پردازش تراکنش نبود [۲۳].
- در سال ۲۰۱۵ هواپیمای جنگی F-۳۵ قربانی یک نقص نرم‌افزاری شد و نتوانست اهداف را به درستی مورد هدف قرار دهد [۲٤].
- ایرباس A۳۰۰ هواپیمایی چین در آوریل ۱۹۹۴ به‌دلیل نقص نرم‌افزاری سقوط کرد و ۲۶۴ انسان بی‌گناه کشته شدند [۲۵].
- در آوریل ۱۹۹۹ یک نقص نرم‌افزاری موجب شکست در پرتاب ماهواره نظامی شد تا یکی از گرانترین اتفاقات در این زمینه با هزینه ۱.۲ میلیارد دلار رقم بخورد [۲٦].

بنابراین، نیاز است که خطاهای سیستم‌های فناوری اطلاعات در همان مراحل اولیه توسعه نرم‌افزار، شناسایی و آنالیز شده یا با ارزیابی ریسک پروژه، مانع از ادامه انجام پروژه شد.

در بخش ۲، کارهای مرتبط و در بخش ۳، پیچیدگی کد بیان شده است. در بخش ۴، به چارچوب سیستم مدیریت ایمنی اشاره شده و بخش ۵، روش پیشنهادی ارائه شده است.

۱-۲- کارهای مرتبط

با توجه به این‌که مشکل کمبود حافظه یکی از شایع‌ترین نقص‌ها در سیستم‌های نهفته است، در مقاله [۱] نیز ذکر شده که تمام نقص‌های سرریز عدد صحیح[1] مشخص شده توسط ابزارهای تجزیه و تحلیل ایستا نقاط ضعف هستند و در نهایت باید اصلاح شوند؛ همچنین آن‌ها از مجموعه آزمون Juliet test suite استفاده کردند. در [۲] سه ابزار برای پشتیبانی از تحلیل کد با استفاده از روش DESMET تحلیل و ارزیابی شدند. DESMET برای ارزیابی ابزارها یا روش‌ها است که شامل ۹ روش ارزیابی مناسب برای تجزیه و تحلیل ابزارهای تحلیل کد مختلف است. در [۳] هدف کار آن‌ها مطالعه ترکیب ابزارهای تجزیه‌وتحلیل خودکار کد در برنامه‌های کاربردی وب به‌عنوان راهی برای بهبود عملکرد تشخیص آسیب‌پذیری است.

همان‌طور که در [٤] بیان شده، اکثر کدهای سیستم به زبان-های C و C++ نوشته شده‌اند، خطاهای اساسی مورد بررسی آن‌ها مربوط به حافظه، حسابگرها، عملیات منطقی، خطاهای اشاره‌گر و غیره است. مقاله [٤] شامل بررسی ابزارها، رفتار ابزارها در برابر مشکلات ورودی، مشکلات توابع، حلقه‌ها، متغیرها، اشاره‌گرها و برخی مشکلات اساسی کدگذاری است.

در [۵] استفاده از تحلیل ایستا در استاندارد ایمنی خودرو (ISO۲۶۲۶۲) با استفاده از قوانین AUTOSAR برای کاهش خطاهای زمان اجرا نرم‌افزار توصیه می‌شود (که بالاتر اهمیت آن ذکر شد). هدف مقاله [۵] شناسایی بهترین ترکیب از ابزارهای تجاری برای به حداقل رساندن نقص‌ها است. در [۱۸] سنجه‌های مختلف پیچیدگی کد در متدها و کلاس‌ها مورد ارزیابی و بررسی قرار گرفته است.

۲- پیچیدگی کد

اندازه‌گیری پیچیدگی کد بهترین راه برای پیش‌بینی احتمال نقص است[٦]. یک ضرب‌المثل قدیمی در فناوری وجود دارد که می-گوید: «شما نمی‌توانید آنچه را که نمی‌توانید اندازه‌گیری کنید،

[1] Integer overflow

مدیریت کنید.

بر کسی پوشیده نیست که نوشتن، اشکال‌زدایی و نگهداری کد کاری دشوار است. این جنبه‌ها همراه با ادغام برنامه‌های کاربردی، کد را با کیفیت بالا نگه می‌دارد. هزینه رفع خطاها همیشه در مراحل ابتدایی تولید و توسعه کم است. با وجود رابط‌های سیستمی متعدد، الزامات متعدد، پیچیدگی سیستم‌های نرم‌افزاری گاهی از کنترل خارج می‌شود. ارائه برخی برنامه‌ها برای نگهداری از این کدها، بسیار گران و برای بهبود آن بسیار خطرناک است. اگر به کیفیت کد توجه نشود، پیچیدگی کد می‌تواند در پروژه‌های ارائه شده، بسیار زیاد شود به‌طوری‌که از آن نقطه به بعد، نه می‌توان کد را بهبود بخشید و نه می‌توان به دلیل محدودیت بودجه و زمان از ابتدا شروع کرد [۱۹]. از این‌رو اخیراً مدیران توسعه علاقه زیادی به اندازه‌گیری کیفیت کد نشان داده‌اند. بنابراین، ابزارهای تجزیه و تحلیل کد بسیار محبوب شده‌اند.

پیچیدگی کد، تعداد مسیرهایی که کد برنامه، می‌تواند طی کند را اندازه‌گیری می‌کند. مغز انسان محدود است و نمی‌تواند تمام پیامدهای صدها مسیر مختلف را در یک عملکرد مشخص درک کند. وقتی توسعه‌دهنده، نویسنده کد نیست و سعی می‌کند مسیرهای منطقی درون یک تابع را بفهمد، این امر حتی دشوارتر می‌شود. با چندین مسیر تودرتو، ایجاد یک مدل ذهنی از همه حالت‌های برنامه غیرممکن است. به همین دلیل است که واحدهای کد باید کوچک و پیچیدگی کمی داشته باشند. در مقاله‌ای که در کنفرانس بین‌المللی معتبر مهندسی نرم‌افزار (ICSE) که در سال ۲۰۰۸ منتشر شد، محققان مایکروسافت دریافتند که مسائل سازمانی، پیچیدگی کد[۱]، انحراف کد[۲](زمانی است که توسعه‌دهنده کد را در یک دوره مشخص تغییر می‌دهد.) و پوشش کد[۳] مهم‌ترین پیش‌بینی‌کننده‌های اشکالات نرم‌افزاری هستند. این مطالعه در سال ۲۰۱۵ نیز تکرار شد [۷] (همچنین در ICSE منتشر شد).

پیچیدگی کد معیار مناسبی برای پیش‌بینی اشکالات می‌باشد. در حالی که علاقه فزاینده‌ای به افزایش پوشش کد و کاهش پیچیدگی کد وجود دارد، اما همچنان به اندازه کافی مورد توجه قرار نمی‌گیرند.

۲-۱- مزایای اندازگیری پیچیدگی کد

اندازه‌گیری پیچیدگی نرم‌افزار بر اساس الگوریتم‌های تعریف شده، ارزیابی جامعی از کد ارائه می‌دهد. برخی از مهم‌ترین مزایای اندازه‌گیری پیچیدگی نرم‌افزار طبق [۸] و [۹] به شرح زیر است:

- آزمون قابل اعتماد

اندازه‌گیری پیچیدگی کد، به توسعه‌دهنده می‌گوید که چند مسیر در کد وجود دارد. بنابراین، توسعه‌دهنده می‌داند که چند مسیر برای آزمایش وجود دارد. این موضوع به آن‌ها کمک می‌کند تا حداقل تعداد آزمون‌های مورد نیاز برای پوشش کل کد را محاسبه کنند.

- خوانایی کد

یکی از معروف‌ترین گفته‌ها در صنعت IT عبارت است از: «*خواندن کد از نوشتن آن سخت‌تر است.*»

از آن‌جایی که کد، بسیار بیشتر از آن‌چه نوشته شده، خوانده می‌شود، این جمله درست است. پیچیدگی بالای کد می‌تواند به این معنی باشد که خواندن کد و درک عملکرد آن حتی با مستندات مناسب دشوار است. این موضوع، عواقب قابل توجهی برای مهندسان و مدیران دارد. کد قابل خواندن که به جلوگیری از تکرار روش‌های مشابه کمک می‌کند، کمتر مستعد خطا است زیرا تشخیص آن آسان است و بهبود یا رفع اشکال در آن سریع‌تر است.

- کاهش خطر نقص

با کاهش پیچیدگی کد، توسعه‌دهندگان می‌توانند ریسک معرفی خطاهای بیشتر را کاهش دهند. از این گذشته، یک توسعه‌دهنده خوب هرگز با خطوط کدی که نوشته است ارزیابی نمی‌شود، بلکه کیفیتی که در کد حفظ کرده است اهمیت دارد.

- زمان

یکی دیگر از تأثیرات داشتن سیستم‌هایی با کد پیچیده این است که می‌تواند مستقیماً بر زمان‌های Compile نرم‌افزار تأثیر بگذارد که به معنای افزایش منابع محاسباتی مصرف شده و اتلاف وقت برای مهندسان است.

- هزینه نگهداری کمتر

با کاهش پیچیدگی، احتمال ایجاد نقص کاهش داده می‌شود. هنگامی که خطر اشکالات احتمالی کاهش می‌یابد، نقص‌های کمتری برای یافتن وجود دارد. بنابراین هزینه نگهداری به‌میزان قابل توجهی کاهش خواهد یافت.

- پیش‌بینی‌پذیری بیشتر

اندازه‌گیری پیچیدگی کد، به توسعه‌دهندگان کمک می‌کند تا زمان تکمیل یک بخش از کد یا ویرایش آن را پیش‌بینی کنند. همچنین این دانش به سازمان‌ها اجازه می‌دهد تا مدت زمان ارائه نسخه را بهتر پیش‌بینی کنند. از این رو، مشاغل وابسته به چنین نرم‌افزارهایی، می‌توانند اهداف و انتظارات خود را بهتر تعیین کنند. یکی دیگر از مزایای بسیار مهم این است که به تیم‌های توسعه اجازه می‌دهد تا بودجه‌های واقعی‌تری را تنظیم کنند.

۲-۲- برخی خطاهای رایج سیستم‌های نهفته

یافتن و از بین بردن اشکالات نهفته در نرم‌افزار سیستم‌های نهفته،

[۱] code complexity

[۲] code churn

[۳] code coverage

یک تجارت دشوار است. تلاش‌های زیاد و ابزارهای گران قیمت، اغلب برای ردیابی رفتارهای برنامه‌ریزی نشده در زمان اجرا مورد نیاز است. در بدترین حالت، علت اصلی به‌گونه‌ای نامحسوس به کد یا داده‌ها آسیب می‌رساند که به نظر می‌رسد سیستم عمدتاً خوب کار می‌کند. اغلب مهندسان از تلاش برای کشف علت ناهنجاری‌های نادر دست می‌کشند زیرا نمی‌توان آن‌ها را به راحتی در آزمایشگاه بازتولید کرد. بنابراین در بخش طبق سه مرجع [۱۰]، [۱۱] و [۱۲] برخی از رایج‌ترین اشکالات نرم‌افزاری در اکثر سیستم‌های نهفته آورده شده است.

- نشت حافظه[۱]

نشت حافظه [۱۳]زمانی رخ می‌دهد که حافظه تخصیص داده شود و پس از استفاده، آزاد نشود؛ یا زمانی که اشاره‌گر به تخصیص حافظه حذف می‌شود و حافظه دیگر قابل استفاده نیست. نشت حافظه عملکرد را کاهش می‌دهد و به مرور زمان باعث می‌شود که حافظه تمام شود و برنامه از کار بیفتد. طراحی و برنامه‌نویسی یک نرم‌افزار نهفته نیاز به دقت زیادی دارد. برنامه باید به اندازه کافی قوی باشد تا بتواند هر خطای احتمالی را که ممکن است رخ دهد، مدیریت کند. باید بتوان این خطاها را پیش‌بینی کرد و با آن‌ها برخورد کرد (به ویژه در حوزه حافظه). اغلب یک برنامه می‌تواند برای مدتی اجرا شود، قبل از اینکه به‌طور مرموزی خودش یا سیستم را بر اثر تخصیص حافظه‌ای که هرگز آزاد نمی‌شود از کار بیاندازد.

- سرریز پشته[۲]

سرریز پشته، یک وضعیت نامطلوب است که در آن یک برنامه کامپیوتری خاص سعی می‌کند از فضای حافظه بیشتری نسبت به پشته در دسترس، استفاده کند. هر چند اثر سرریز هر پشته متفاوت است اما ماهیت آسیب و زمان‌بندی رفتار نادرست برنامه کاملاً به این بستگی دارد که کدام داده‌ها یا دستورالعمل‌ها مورد استفاده قرار می‌گیرند و چگونه از آن‌ها استفاده می‌شود. متأسفانه، سرریز پشته به‌مراتب بیشتر از کامپیوترهای رومیزی به سیستم‌های نهفته آسیب می‌زند که این معمولاً به‌دلیل مقدار RAM پایین موجود در سیستم‌های نهفته است.

- عدم وجود کلمه کلیدی فرار[۳]

کلمه کلیدی فرار در C توسط برنامه‌نویس زمانی که یک متغیر را در کد منبع تعریف می‌کند، استفاده می‌شود و از آن برای اطلاع Compiler استفاده می‌شود که مقدار متغیر می‌تواند در هر زمان تغییر کند. هدف جلوگیری از بهینه‌سازی[۴] در خواندن و نوشتن آن متغیر است.

- تکه تکه شدن پشته[۵]

تخصیص حافظه پویا به‌طور گسترده توسط توسعه‌دهندگان نرم‌افزار نهفته، استفاده نمی‌شود. یکی از دلایل آن مشکل تکه تکه شدن پشته است. مدیریت حافظه پویا می‌تواند مشکل‌ساز و ناکارآمد باشد. برای برنامه‌های دسکتاپ، که حافظه به‌صورت رایگان در دسترس است، می‌توان این مشکلات را نادیده گرفت. اما برای برنامه‌های نهفته این‌گونه نیست. داده‌های ذخیره‌شده در RAM که دیگر مورد نیاز نیستند را می‌توان با فراخوانی تابع free یا استفاده از کلمه کلیدی delete به حالت اول برگرداند. در تئوری این باعث می‌شود که فضای ذخیره‌سازی برای استفاده مجدد در دسترس باشد. سوال این است که Heap Fragmentation چگونه می‌تواند مشکل‌ساز باشد؟ هنگامی که برای آزاد کردن یک بلوک، (free) را فراخوانی می‌کنید، یک حفره از حافظه استفاده نشده ایجاد می‌کنید. پس از مدتی حافظه همانند یک پنیر سوئیسی با سوراخ‌های زیاد تبدیل می‌شود.

- وضعیت مسابقه[۶]

وضعیت مسابقه زمانی رخ می‌دهد که یک دستگاه یا سیستم سعی می‌کند دو یا چند عملیات را همزمان انجام دهد. در واقع زمانی اتفاق می‌افتند که دو پردازش برنامه کامپیوتری یا رشته تلاش می‌کنند به یک منبع به‌طور همزمان دسترسی پیدا کنند و باعث ایجاد مشکل در سیستم شوند.

- بن بست[۷]

در محاسبات همزمان، بن‌بست به هر وضعیتی گفته می‌شود که در آن هیچ یک از رشته‌ها یا نخ‌ها[۸] نمی‌توانند به کار خود ادامه دهند، زیرا هر یک از اعضا منتظر است تا اقدامی توسط دیگری انجام شود. مانند ارسال یک پیام یا به‌طور معمول، آزاد کردن قفل. در واقع در یک سیستم عامل، بن‌بست زمانی رخ می‌دهد که یک فرایند یا رشته وارد حالت انتظار می‌شود، زیرا منتظر منبع یا رشته دیگری است که فرایند آن انجام شود.

۲-۳- ابزار SourceMonitor

سورس مانیتور[۹] [۱۴]که توسط جیمز وانر[۱۰] از Campwood Softwareنوشته شده است، ابزاری ساده است که آمار پیچیدگی چرخه‌ای و سایر معیارها را برای بسیاری از زبان‌ها از جمله #C، C++، C، جاوا، Delphi، Visual Basic و حتی HTML

---

۱ memory leak
۲ stack overflow
۳ missing volatile keyword
۴ optimization
۵ heap fragmentation
۶ race condition
۷ dead lock
۸ threads
۹ SourceMonitor
۱۰ James Wanner

جمع‌آوری می‌کند. سورس مانیتور همچنین از ۱۱ معیار مفید، از جمله تعداد خطوط کل در فایل‌های کد، تعداد کلاس‌ها، میانگین عبارات در هر تابع و غیره را گزارش می‌دهد. یکی از بزرگترین ویژگی‌های سورس مانیتور ردیابی تغییرات کد است. یک پروژه ایجاد شده در سورس مانیتور که حاوی تمام کد منبع است، پس از هر تغییرات می‌توان پیچیدگی و معیارهای دیگر را دوباره بررسی کرد. این مورد، کمک می‌کند تا هنگام افزودن قابلیت به نرم‌افزار، منحنی پیچیدگی در مقابل ویژگی‌های اضافه شده، مشاهده شود. این ابزار رایگان، این امکان را می‌دهد که پیچیدگی نسبی ماژول‌ها شناسایی شود. به‌عنوان مثال، می‌توان از سورس مانیتور برای شناسایی کدهایی استفاده کرد که به احتمال زیاد دارای نقص هستند و بنابراین نیاز به بازبینی دارند. سورس مانیتور که به زبان C++ نوشته شده است، کد را با سرعت بالا اجرا می‌کند (معمولاً حداقل ۱۰۰۰۰ خط کد را در ثانیه بررسی می‌کند).

۲-۳-۱- معیارهای مورد بررسی در ابزار Source Monitor

در این بخش مواردی که ابزار Source Monitor بررسی می‌کند، بیان می‌شوند. این موارد عبارت‌اند از: تعداد خطوط کد، تعداد دستورات[۱]، درصد شاخه‌ها[۲]، درصد خطوط با نظرات[۳]، تعداد توابع[۴]، میانگین دستورات در هر تابع[۵]، بیشترین پیچیدگی[۶]، میانگین پیچیدگی[۷]، بیشترین عمق قطعه[۸] و میانگین عمق قطعه[۹].

به طور کلی مقادیر بالا بسیار حائز اهمیت می‌باشند و آزمونگر نرم‌افزار می‌تواند با بررسی آن‌ها ارزیابی کلی از سطح کیفیت کد انجام دهد. تعداد خطوط کد به طور کلی باعث افزایش پیچیدگی نمی‌شود و فقط نشان‌دهنده حجم پروژه می‌باشد. در ادامه موارد بالا با استفاده از راهنمای ابزار توضیح داده می‌شود.

- دستورات:

در زبان C، دستورات با یک کاراکتر نقطه ویرگول خاتمه می‌یابند. به‌طور کلی شاخه‌هایی مانند if، for، while و goto نیز به‌عنوان دستورات شمرده می‌شوند. دستورالعمل‌های پیش‌پردازنده #include، #define و#undef به‌عنوان دستورات شمرده می‌شوند و همه دستورالعمل‌های پیش‌پردازنده دیگر نادیده گرفته می‌شوند. علاوه بر این، تمام عبارت‌های بین دستور #else یا #elif و دستور پایانی #endif نادیده گرفته می‌شوند. این معیار تمام دستورات موجود در یک فایل یا در تمام فایل‌ها را شمارش می‌کند.

- درصد شاخه‌ها:

عبارت‌هایی که باعث وقفه در اجرای متوالی دستورات می‌شوند، به‌طور جداگانه شمارش می‌شوند. که شامل if، else، for، while، goto، break، continue، switch، case، default و return هستند و به‌عنوان شاخه‌ها شمارش می‌شوند. توجه داشته باشید که do محاسبه نمی‌شود زیرا همیشه به دنبال while می‌آید.

- درصد خطوط با نظرات:

خطوطی که حاوی نظرات به سبک C (/*...*/) یا سبک C++ (//...)هستند، شمارش می‌شوند و برای محاسبه این معیار با تعداد کل خطوط موجود در فایل مقایسه می‌شوند. اگر گزینه نادیده گرفتن سرصفحه‌ها و پاورقی‌ها انتخاب شود، نظرات موجود در ابتدا و انتهای فایل‌ها محاسبه نمی‌شوند.

- تعداد توابع:

تعداد توابع یافت شده در یک فایل یا در تمام فایل‌های موجود است.

- میانگین دستورات در هر تابع:

تعداد کل دستورات یافت شده در داخل توابع تقسیم بر تعداد توابع یافت شده.

- پیچیدگی:

معیار پیچیدگی تقریباً همان‌طور که توسط استیو مک‌کانل در کتاب Code Complete [۱۵] در ۱۹۹۳ تعریف شده است، محاسبه می‌شود. معیار پیچیدگی تعداد مسیرهای اجرا را از طریق یک تابع اندازه‌گیری می‌کند. هر تابع دارای پیچیدگی ۱ است، بعلاوه ۱ پیچیدگی برای هر دستور مانند if، else، for، foreach یا while محاسبه می‌شود. عبارت‌های حسابی « MyBoolean ? ValueIfTrue: ValueIfFalse» هر کدام ۱ عدد به مجموع پیچیدگی اضافه می‌کنند. همچنین ۱ عدد پیچیدگی برای هر &&و || اضافه می‌شود.

عبارت‌های سوئیچ تعداد پیچیدگی را برای هر خروج از یک case، ۱ عدد و هر دستور catch یا except در یک بلوک try هر کدام ۱ عدد به پیچیدگی اضافه می‌کنند.

  - بیشترین پیچیدگی:

مقدار پیچیدگی در پیچیده‌ترین تابع در فایل کد منبع است.

  - میانگین پیچیدگی:

مقیاسی از میانگین پیچیدگی کلی اندازه‌گیری شده در توابع است.

- عمق قطعه:

---

[۱] Statements
[۲] Branches percent
[۳] Percent Lines with Comments
[۴] Functions
[۵] Average Statements per Function
[۶] Max Complexity
[۷] Average Complexity
[۸] Max Block Depth
[۹] Average Block Depth

عمق بلوک تودرتو یافت شده که در ابتدای هر فایل صفر است. این عدد تا عمق ۹ ثبت می‌شود و تمام دستورات در سطوح عمیق‌تر به‌عنوان عمق ۹ حساب می‌شوند و با برچسب «۹+» نشان داده می‌شود. اگر گزینه «نمایش حداکثر عمق بلوک اندازه‌گیری شده» در گزینه‌ها زده شود، حداکثر عمق واقعی را (به جای «۹+») تا عدد ۳۲ می‌تواند نشان دهد.

- بیشترین عمق قطعه:

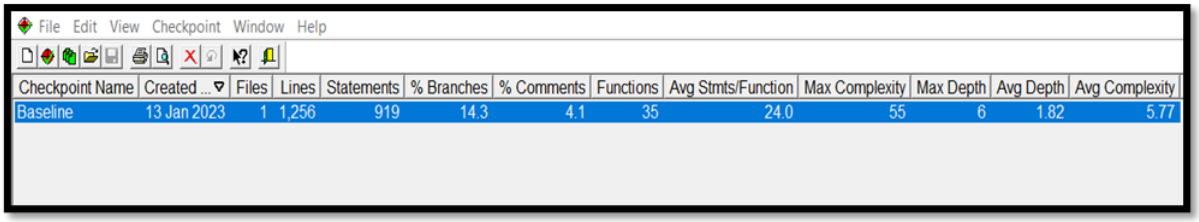
حداکثر عمق بلوک تودرتو یافت شده در کد است.

- میانگین عمق قطعه:

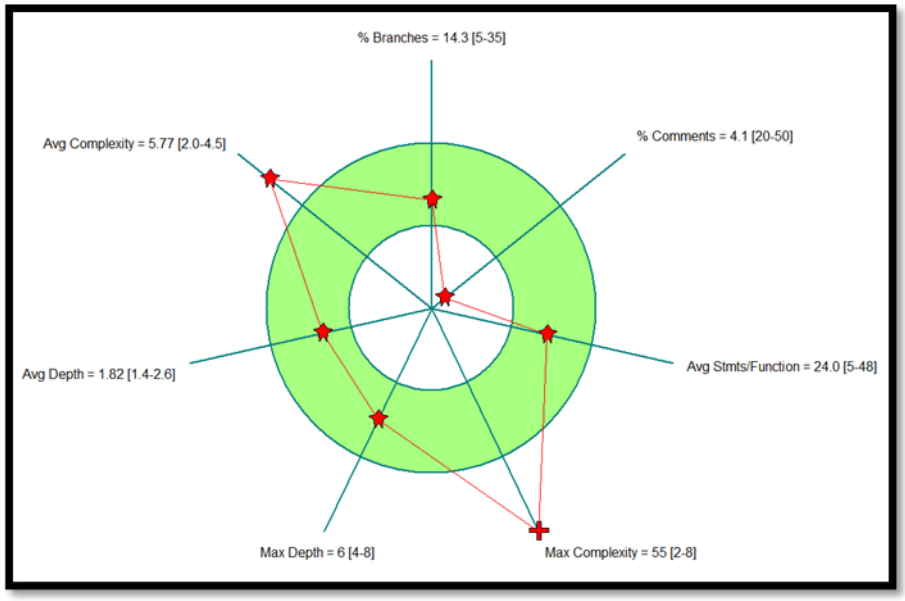
میانگین وزنی عمق بلوک تمام دستورات است. نحوه محاسبه آن به این شکل است که عدد عمق تمام دستورات را با هم جمع می‌کنیم و تقسیم بر تعداد کل دستورات می‌کنیم.

۲-۳-۲- محاسبه پیچیدگی کد، تعداد خطوط و میانگین دستورات در هر تابع با ابزار Source Monitor

در این بخش یک مثال، برای مشاهده خروجی ابزار Source Monitor، آورده شده است. یک کد آزمایشی به زبان C به‌عنوان ورودی، به ابزار داده شده است (شکل ۱ و ۲).



شکل(۱) : خروجی ابزار Source Monitor



شکل (۲) : نمودار کیویات ابزار Source Monitor

در این مقاله برای اندازه‌گیری پیچیدگی کد، از معیار میانگین پیچیدگی کد محاسبه شده توسط ابزار Source Monitor استفاده شده است. همچنین برای اندازه‌گیری معیار میانگین دستورات در هر تابع و تعداد خطوط کد از این ابزار استفاده شده است. میانگین دستورات در هر تابع در هر دو شکل ۱ و ۲، قابل مشاهده است اما تعداد خطوط کد در نمودار کیویات نشان داده نمی‌شود و در خروجی شکل ۱ قابل مشاهده است.

۳- چارچوب سیستم مدیریت ایمنی (SMS)

سیستم مدیریت ایمنی[۱] [۲۰]، چارچوب یا رویکردی مدیریتی است که برای کاهش و کنترل ریسک ایمنی در محل کار، کاربرد دارد. SMS به کلیه رویه‌ها، سیاست‌ها و فعالیت‌هایی اشاره دارد که سازمان برای رساندن خطرات ایمنی در سطوح قابل‌تحمل انجام می‌دهد. سیستم‌های مدیریت ایمنی پایه و اساس تصمیم‌گیری مبتنی بر ریسک را فراهم می‌کنند و به سازمان‌ها اجازه می‌دهند تا از ایمنی عملیات اطمینان حاصل کنند.

امروزه مدیریت ایمنی بر اساس پیاده سازی اقداماتی پیشگیرانه برای جلوگیری از وقوع حادثه بنا شده است. به منظور جلوگیری از

[۱] Safety Management System

حوادث باید فرهنگ ایمنی در سازمان شکل گیرد. فرهنگ ایمنی چیزی جز ارزش‌ها، باورها، تفکرات و رفتارهای افراد در خصوص ایمنی نیست. فرهنگ ایمنی در سطوح مختلف سازمان قابل تعریف است. استاندارد سیستم مدیریت ایمنی OHSAS ۱۸۰۰۱ در سال ۲۰۰۷ توسط موسسه استاندارد بریتانیا منتشر شد. سیستم مدیریت ایمنی یا به اختصارSMS، سازمان‌ها را برای کنترل و به حداقل رساندن ریسک‌های ایمنی یاری می‌کند و عملکرد آن‌ها را در این زمینه ارتقا می‌دهد.

۳-۱- اهمیت سیستم مدیریت ایمنی

ایمنی کارکنان از اهمیت بالایی برخوردار است و به شرکت‌ها کمک می‌کند تا از خسارات مالی و انسانی جلوگیری کنند. در ادامه برخی از اهمیت‌های سیستم مدیریت ایمنی توضیح داده خواهد شد [۲۰]:

۱) حفظ ایمنی کارکنان

یکی از جوانمردانه‌ترین دلایل برای اهمیت سیستم مدیریت ایمنی، حفظ سلامت و ایمنی کارکنان است. ایمنی کارکنان به‌عنوان دارایی ارزشمند سازمان در نظر گرفته می‌شود و هر گونه حادثه یا صدمه به آن‌ها می‌تواند عواقب جبران‌ناپذیری داشته باشد.

۲) کاهش هزینه‌ها

حوادث و صدمات کاری نه‌تنها به کارکنان بلکه به سازمان هم هزینه‌های مالی بسیاری تحمیل می‌کنند. از جمله هزینه‌های پزشکی، جبران خسارت‌ها و هزینه‌های قانونی. یک سیستم مدیریت ایمنی قوی می‌تواند به کاهش این هزینه‌ها کمک کند.

۳) افزایش بهره‌وری

محیط کاری امن و بدون حادثه می‌تواند به افزایش بهره‌وری کارکنان منجر شود. افرادی که در محیطی ایمن کار می‌کنند، بهترین عملکرد را از خود ارائه می‌دهند و به بهبود عملکرد کل سازمان کمک می‌کنند.

۴) افزایش اعتماد عمومی

سازمان‌هایی که به ایمنی کارکنان و محیط کاری خود توجه کافی می‌کنند، اغلب اعتماد عمومی بیشتری دارند. مشتریان و جامعه به‌طورکلی به سازمان‌های ایمن‌سازی شده بیشتر اعتماد می‌کنند.

۵) اطمینان از پایداری سازمان

سیستم مدیریت ایمنی، به سازمان اطمینان می‌دهد که می‌تواند با چالش‌ها و مخاطرات مختلف مواجه شود و پایداری خود را حفظ کند. این امر به معنای حفظ تجارت و رشد سازمان است.

۴- روش پیشنهادی

۴-۱- آماده‌سازی

پس از مشخص شدن پیچیدگی کد، تعداد خطوط و میانگین تعداد دستورات در هر تابع، می‌توان یک آنالیز از ریسک موجود با استفاده از چارچوب SMS انجام داد. ابتدا هر ۳ معیار گفته شده در ۵ سطح طبقه‌بندی می‌شوند.

برای میانگین پیچیدگی کد عدد خروجی ابزار سورس‌مانیتور، ۱ پایین‌ترین سطح و ۵ بالاترین سطح میانگین پیچیدگی کد می‌باشد. باتوجه به اینکه حداقل میانگین پیچیدگی کد عدد ۱ می‌باشد. بنابراین اگر عدد مربوط به میانگین پیچیدگی کد که در این مقاله همان پیچیدگی کد در نظر گرفته می‌شود، در ابزار Source monitor بین ۱ تا ۳ باشد پیچیدگی خیلی کم یا سطح ۱؛ اگر بین ۳ تا ۵ باشد کم یا سطح ۲، بین ۵ تا ۷ باشد متوسط یا سطح ۳، بین ۷ تا ۹ باشد زیاد یا سطح ۴ و ۹ یا بیشتر باشد به معنی پیچیدگی خیلی زیاد یا سطح ۵ تلقی می‌شود.

برای طبقه‌بندی تعداد دستورات به ازای توابع، اگر عدد مربوط به میانگین دستورات هر تابع در ابزار Source monitor بین ۱ تا ۲۰ باشد پیچیدگی خیلی کم یا سطح E؛ اگر بین ۲۰ تا ۴۰ باشد کم یا سطح D، بین ۴۰ تا ۶۰ باشد متوسط یا سطح C، بین ۶۰ تا ۸۰ باشد زیاد یا سطح B و ۸۰ یا بیشتر باشد به معنی خیلی زیاد یا سطح A تلقی می‌شود.

طبق کتاب code-complete ویرایش دوم [۱۶] با افزایش تعداد خطوط کد در پروژه احتمال وجود خطا بالاتر می‌رود (جدول ۱).

جدول(۱) : اندازه پروژه و تراکم خطا [۱۶]

تراکم خطا	اندازه پروژه (تعدادخطوط کد)
۰ تا ۲۵ خطا در یک‌صد خط کد (KLOC[1])	کمتر از ۲K
۰ تا ۴۰ خطا در KLOC	۲K تا ۱۶K
۰.۵ تا ۵۰ خطا در KLOC	۱۶K تا ۶۴K
۲ تا ۷۰ خطا در KLOC	۶۴K تا ۵۱۲K
۴ تا ۱۰۰ خطا در KLOC	۵۱۲K یا بیشتر

همان‌طور که در جدول ۱ مشاهده می‌شود، تعداد خطای موجود در یک پروژه با کمتر از ۲۰۰۰ خط کد، به ازای هر ۱۰۰۰ خط حدودا بین ۰ تا ۲۵ می‌باشد. همچنین در یک پروژه با ۲۰۰۰ تا ۱۶۰۰۰ خط کد بین ۰ تا ۴۰، ۱۶۰۰۰ تا ۶۴۰۰۰ خط کد بین ۰.۵ تا ۵۰، ۶۴۰۰۰ تا ۵۱۲۰۰۰ خط کد بین ۲ تا ۷۰ و ۵۱۲۰۰۰ یا بیشتر بین ۴ تا ۱۰۰ می‌باشد. واضح است که با افزایش تعداد خطوط کد ریسک وجود خطا در کد بالاتر است.

[1] Thousand lines of code

با توجه به موارد گفته شده می‌توان تعداد خطوط کد را نیز به ۵ سطح تقسیم‌بندی کرد. به طوری‌که اگر تعداد خطوط کمتر از ۲۰۰۰ خط باشد، سطح ۱، ۲۰۰۰ تا ۱۶۰۰۰ خط کد سطح ۲، ۱۶۰۰۰ تا ۶۴۰۰۰ خط کد سطح ۳، ۶۴۰۰۰ تا ۵۱۲۰۰۰ خط کد سطح ۴ و ۵۱۲۰۰۰ یا بیشتر سطح ۵ در نظر گرفته می‌شود.

۴-۲- ارزیابی ریسک کد با استفاده از چارچوب سیستم مدیریت ایمنی

با توجه به ماتریس ارزیابی ریسک در چارچوب سیستم مدیریت ایمنی (SMS) که به معنی رتبه‌بندی رویدادها استفاده می‌شود و تصمیم می‌گیرد که آیا باید ریسک پروژه پذیرفته شود یا میزان ریسک بالا بوده و پروژه ایمنی مورد نظر را ندارد و باید آن را کاهش داد. (جدول ۲)

جدول (۲) : ماتریس ارزیابی ریسک در چارچوبSMS [۱۷]

Risk Likelihood		Risk Severity				
		Catastrophic A	Hazardous B	Major C	Minor D	Negligible E
Frequent	5	5A	5B	5C	5D	5E
Occasional	4	4A	4B	4C	4D	4E
Remote	3	3A	3B	3C	3D	3E
Improbable	2	2A	2B	2C	2D	2E
Extremely Improbable	1	1A	1B	1C	1D	1E

در دو جدول زیر مقادیر احتمال ریسک و شدت ریسک مربوط به ماتریس گفته شده، توضیح داده شده است (جدول ۳ و ۴).

جدول (۳) : شدت ریسک [۱۷]

ارزش	شدت (severity)	توضیحات
A	مصیبت‌بار – فاجعه‌آمیز (CATASTROPHIC)	○ نابودی تجهیزات ○ خسارت‌های جانی و مالی شدید
B	خطرناک (HAZARDOUS)	○ کاهش شدید حاشیه‌های ایمنی به طوری که نمی‌توان به سیستم برای انجام دقیق یا کامل وظایف خود اعتماد کرد ○ مصدومیت شدید ○ آسیب عمده تجهیزات
C	عمده (MAJOR)	○ کاهش قابل توجه حاشیه‌های ایمنی، کاهش توانایی سیستم برای مقابله با شرایط نامطلوب عملیاتی ○ حادثه جدی ○ صدمه به افراد
D	جزئی (MINOR)	○ مزاحمت در عملیات ○ ایجاد محدودیت‌ها ○ حادثه جزئی
E	قابل اغماض (NEGLIGIBLE)	○ عواقب کم

جدول (۴) : احتمال ریسک [۱۷]

ارزش	احتمال (likelihood)	توضیحات
۱	به شدت غیرممکن (EXTREMELY IMPROBABLE)	تقریباً غیر قابل تصور است که این رویداد رخ دهد
۲	بسیار نامحتمل (IMPROBABLE/Extremely Remote)	بسیار بعید است اتفاق بیفتد (معلوم نیست که رخ داده باشد)
۳	نامحتمل	بعید است اتفاق بیفتد، اما ممکن است (به ندرت رخ داده است)

	(REMOTE)	
۴	گاه به گاه (OCCASIONAL)	به احتمال زیاد گاهی اوقات رخ می‌دهد (به ندرت رخ داده است)
۵	زود به زود (FREQUENT)	به احتمال زیاد بارها رخ می‌دهد (اغلب رخ داده است)

با توجه به اینکه با افزایش پیچیدگی کد[۱](CC) و تعداد خطوط کد[۲] (LOC) احتمال ایجاد خطا بالا می‌رود وهمچنین با توجه به اینکه در یک پروژه توابع دارای دستورات زیاد معمولا توابع اصلی و حیاتی سیستم هستند، می‌توان اینطور در نظر گرفت که اگر خطایی در این توابع (که توابع اصلی موجود در کد هستند) ایجاد شود شدت بالایی خواهد داشت و می‌تواند خسارت بیشتری نسبت به توابع کوچکتر وارد کند. بنابراین معیار میانگین دستورات در هر تابع[۳] (SpF) که به ۵ سطح تقسیم شد، نشان‌دهنده همین موضوع می‌باشد و اگر سطح آن (که قبلا بررسی شد) ۱ باشد، یعنی توابع آن پروژه به نسبت دارای دستورات کمی بوده و شدت خطا در آن کم‌تر می‌باشد و اگر سطح آن ۵ باشد یعنی توابع آن پروژه به نسبت دارای دستورات زیادی بوده و شدت خطا در آن بسیار بالا می‌باشد.

در این مقاله با توجه به موارد گفته شده و استفاده از ماتریس ارزیابی ریسک در چارچوب SMS، ماتریس ارزیابی ریسک کد ارائه شده است؛ که به برای شدت ریسک کد[۴] (CRS)، سطح میانگین تعداد دستورات در هر تابع قرار می‌گیرد و برای محاسبه احتمال ریسک کد[۵] (CRL) از فرمول زیر استفاده می‌شود:

$$CRL = \frac{CC + LOC}{۲} \qquad (۱)$$

عدد مربوط به احتمال ریسک کد بین ۱ تا ۵ است. اگر ۱ باشد نشان‌دهنده کمترین احتمال ریسک در کد و اگر ۵ باشد نشان‌دهنده بیشترین احتمال ریسک در کد است. ماتریس ارزیابی ریسک کد ارائه شده در این مقاله، مطابق جدول ۵ است:

جدول (۵) : ماتریس ارزیابی ریسک کد ارائه شده در این مقاله

احتمال ریسک کد (CRL)		شدت ریسک کد (CRS)				
		مصیبت‌بار – فاجعه‌آمیز SpF = A	خطرناک SpF = B	عمده SpF = C	جزئی SpF = D	قابل اغماض SpF = E
زود به زود	CRL = ۵	۵A	۵B	۵C	۵D	۵E
گاه به گاه	CRL = ٤	۴A	۴B	۴C	۴D	۴E
نامحتمل	CRL = ۳	۳A	۳B	۳C	۳D	۳E
بسیار نامحتمل	CRL = ۲	۲A	۲B	۲C	۲D	۲E
به شدت غیرممکن	CRL = ۱	۱A	۱B	۱C	۱D	۱E

۵- نتیجه گیری

همانطور که ذکر شد حفظ ایمنی در سیستم‌های فناوری اطلاعات بسیار حائز اهمیت می‌باشد زیرا باعث جلوگیری از حوادث ناگوار و تلفات مالی و حتی جانی می‌شود. در بخش مقدمه ۶ مورد از فجایعی که در گذشته بر اثر نقص نرم‌افزاری رخ داده است، ذکر شد. این نقص‌ها در برخی موارد خسارت‌های سنگین مالی و در برخی دیگر باعث تلفات جانی شده‌اند. ریشه اکثر این نقص‌های نرم‌افزاری در کد پروژه قرار دارد و به همین دلیل پیچیدگی کد در سیستم‌های نهفته و همچنین خطاهای شایع در آن‌ها بررسی شد.

در این مقاله روشی نوین برای آنالیز کد پروژه ارائه شده است. در این روش با استفاده از چارچوب جهانی سیستم مدیریت ایمنی یا SMS به آنالیز احتمال ریسک و شدت ریسک در کد پرداخته شد. یکی از مزایای این روش جلوگیری از راه‌اندازی پروژه‌های ناایمن و بازنویسی کدهای پر ریسک در سیستم‌های نهفته و حیاتی ایمن می‌باشد. همچنین می‌توان طبق ماتریس ارزیابی ریسک کد، به تعیین سطح ریسک کد پرداخته و در نهایت کدهای ناایمن قبل از راه‌اندازی سیستم‌ها شناسایی شده و مانع از تولید محصول شد.

[۱] Code complexity
[۲] Line of code
[۳] Statements per function
[۴] Code risk severity
[۵] Code risk likelihood

۵- مراجع

[۱] Choi, H. J., Lee, H., & Choi, J.-Y. (۲۰۲۱). Is a False Positive really False Positive? ۲۰۲۱ ۲۳rd International Conference on Advanced Communication Technology (ICACT), ۱٤٥–۱٤۹. https://doi.org/۱۰,۲۳۹۱۹/ICACT٥۱۲۳٤,۲۰۲۱,۹۳۷۰٤٤۲

[۲] Nikolić, D., Stefanović, D., Dakić, D., Sladojević, S., & Ristić, S. (۲۰۲۱). Analysis of the Tools for Static Code Analysis. IEEE. https://ieeexplore.ieee.org/abstract/document/۹٤۰۰٦۸۸

[۳] Nunes, P., Medeiros, I., Fonseca, J., Neves, N., Correia, · Miguel, Vieira, M., & Correia, M. (n.d.). An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. https://link.springer.com/article/۱۰,۱۰۰۷/s۰۰٦۰۷-۰۱۸-۰٦٦٤-z

[٤] Fatima, A., Bibi, S., & Hanif, R. (۲۰۱۸). Comparative Study on Static Code Analysis Tools for C/C++. IEEE https://ieeexplore.ieee.org/abstract/document/۸۳۱۲۲٦٥

[٥] Imparato, A., Maietta, R. R., Scala, S., & Vacca, V. (۲۰۱۷). A comparative study of static analysis tools for AUTOSAR automotive software components development. Proceedings - ۲۰۱۷ IEEE ۲۸th International Symposium on Software Reliability Engineering Workshops, ISSREW ۲۰۱۷, ٦٥–٦۸. https://ieeexplore.ieee.org/abstract/document/۸۱۰۹۲٥۳

[٦] A Simple Understanding of Code Complexity - Codegrip. (n.d.). Codegrip. Retrieved April ۲٤, ۲۰۲۳, from https://www.codegrip.tech/productivity/a-simple-understanding-of-code-complexity/

[۷] Caglayan, B., Turhan, B., Bener, A., Habayeb, M., Miransky, A., & Cialini, E. (۲۰۱٥). Merits of Organizational Metrics in Defect Prediction: An Industrial Replication. IEEE/ACM. https://ieeexplore.ieee.org/abstract/document/۷۲۰۲۹٥۳

[۸] A Simple Understanding of Code Complexity - Codegrip. (n.d.). Codegrip. Retrieved April ۲٤, ۲۰۲۳, from https://www.codegrip.tech/productivity/a-simple-understanding-of-code-complexity/

[۹] Kumar, N. (n.d.). Code complexity - a comprehensive guide - Hatica. Hatica. Retrieved April ۲٤, ۲۰۲۳, from https://www.hatica.io/blog/code-complexity/

[۱۰] Barr, M. (۲۰۱۰, April ۱). Five top causes of nasty embedded software bugs - Embedded.com. Embedded Staff. https://www.embedded.com/five-top-causes-of-nasty-embedded-software-bugs/

[۱۱] Most common Errors in embedded systems. (n.d.). Firmcodes. Retrieved April ۲۳, ۲۰۲۳, from http://www.firmcodes.com/common-errors-embedded-systems/

[۱۲] Barr, M. (۲۰۱٦, May ٥). Top ۱۰ Causes of Nasty Embedded Software Bugs. Barrgroup. https://barrgroup.com/embedded-systems/how-to/top-ten-nasty-firmware-bugs

[۱۳] Erickson, C. (۲۰۰۲, August ۳۱). Memory Leak Detection in Embedded Systems | Linux Journal. Linux Journal. https://www.linuxjournal.com/article/٦۰٥۹

[۱٤] James Wanner. (۲۰۲۱). SourceMonitor (Version ۳,٥). Campwood Software. https://www.campwoodsw.com/SourceMonitor.html

[۱٥] MacConnell, S. (۱۹۹۳). Code Complete: A practical handbook of software construction. Microsoft Press Redmont, WA

[۱٦] McConnell, S. (۲۰۰٤). Code complete. Pearson Education.

[۱۷] Chakib, M. (۲۰۱۸). SAFETY MANAGEMENT SYSTEM.

[۱۸] وحیدی اصل, م., دهقانی تفتی, م & ., خلیلیان, ع. (۲۰۲۰). رویکردی جدید مبتنی‌بر سنجه‌های نرم‌افزاری جهت افزایش سودمندی آزمون بازگشت .مجله مهندسی برق دانشگاه تبریز , ٥۰(۱), ۴۶۳–۴۷۶.

[۱۹] چگونه پیچیدگی کد را در یک شرکت نرم‌افزار مدیریت کنیم؟(۲۰۱۷/۰٥/۰۹). خط جدید. از چگونه پیچیدگی کد را در یک شرکت نرم‌افزار مدیریت کنیم؟(khatedid.com)

[۲۰] شاکری فر، پرشان. ۲۰۲۳. سیستم مدیریت ایمنی (SMS) چیست؟. eiqm. از سیستم مدیریت ایمنی (SMS) چیست؟ (eiqm.ir)

[۲۱] J. Kollewe and G. Wearden, “Bloomberg IT meltdown leaves financial world in the dark,” the Guardian, Apr. ۱۷, ۲۰۱٥. https://www.theguardian.com/business/۲۰۱٥/apr/۱۷/uk-halts-bond-sale-bloomberg-terminals-crash-worldwide

[۲۲] J. Hirsch, “Nissan recalls ۱ million cars to fix air bags that may not trigger,” Los Angeles Times, Mar. ۲٦, ۲۰۱٤. https://www.latimes.com/business/la-xpm-۲۰۱٤-mar-۲٦-la-fi-hy-nissan-recalls-altima-sentra-pathfinder-leaf-۲۰۱٤۰۳۲٦-story.html (accessed Dec. ۱۷, ۲۰۲۳).

[۲۳] T. Soper, “Starbucks lost millions in sales because of a ‘system refresh’ computer problem,” GeekWire, Apr. ۲٤, ۲۰۱٥. https://www.geekwire.com/۲۰۱٥/starbucks-lost-millions-in-sales-because-of-a-system-refresh-computer-problem/

[۲٤] K. O. Osborn, “Software Glitch Causes F-۳٥ to Incorrectly Detect Targets in Formation,” Military.com, Mar. ۲٤, ۲۰۱٥. https://www.military.com/defensetech/۲۰۱٥/۰۳/۲٤/software-glitch-causes-f-۳٥-to-incorrectly-detect-targets-in-formation

[۲٥] “Crash of an Airbus A۳۰۰-٦۲۲R in Nagoya: ۲٦٤ killed | Bureau of Aircraft Accidents Archives,” www.baaa-acro.com. https://www.baaa-acro.com/crash/crash-airbus-a۳۰۰-٦۲۲r-nagoya-۲٦٤-killed (accessed Dec. ۱۷, ۲۰۲۳).

[۲٦] R. Bajpai, “What are some recent major computer system failures caused by software bugs?,” www.siliconindia.com. https://www.siliconindia.com/online-courses/tutorials/What-are-some-recent-major-computer-system-failures-caused-by-software-bugs-id-۷٤.html (accessed Dec. ۱۷, ۲۰۲۳).

of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. ۹۹-۰۲, ۱۹۹۹.

Risk evaluation of software code of embedded systems based on safety management system

Ali Esfahani [۱]*, Reza Esfahani [۲], Abulrahman Keshvari[۲]

۱- Master's degree, Islamic Azad University, North Tehran branch (alisf۲۰۰۰@yahoo.com)

۲- Assistant Professor of the Faculty and Research Institute of Non-Professional Defense, Imam Hossein University

Abstract

Today, safety in information technology systems is very important, and failure to guarantee the safety of these systems and software defects, as well as programming errors and mistakes, can cause heavy financial, psychological, and even life-threatening damages. be in the country It is almost impossible to implement the code without defects and errors, very few of these programming errors are identified when the program is compiled; But most of them remain hidden. It is important to be able to determine the risky code before the production of the product and prevent the release of a new version. In this article, the code complexity of embedded systems, the number of code lines, and the number of instructions in each function are determined using the Source Monitor tool. After this step, with the benefit of the safety management system or SMS framework, project risk is analyzed using the risk assessment matrix. Also, common and frequent errors in embedded systems (systems in C language) are investigated.

Keywords: Software safety, embedded systems, code complexity, safety management system framework, risk assessment matrix