بررسی استانداردهای کدنویسی ایمن در زبان C++

محمدمحسن امیری[1]، مرتضی معمر*[2]، موسی محمدنیا[3] و مسعود عسگری‌مهر[4]

[1] دانشگاه آزاد واحد علوم و تحقیقات
mohammadmohsenamiri@Gmail.com

[2] دانشگاه غیرانتفاعی ایوانکی
Moammer.Morteza@Gmail.com

[3] مربی دانشگاه جامع امام حسین (ع)
Mohammadniya.Mousa@sharif.edu.ac.ir

[4] عضو هیئت علمی دانشگاه غیرانتفاعی ایوانکی
Asgarimehr.Masuod@eyc.ac.ir

چکیده

با گسترش روزافزون به‌کارگیری سامانه‌های رایانه‌ای در کاربردهای بحرانی-ایمن، استفاده از روش‌های ارتقای ایمنی و قابلیت اطمینان در فازهای اولیه طراحی و تولید، اهمیت به‌سزایی پیدا کرده است. چون رخداد اشکال یا بروز شکست در این سامانه‌های بحرانی نه‌تنها هزینه‌های بسیاری را به تولیدکننده تحمیل می‌کند، بلکه می‌تواند جان و مال انسان‌ها و نیز محیط‌زیست را به مخاطره بیندازد. در این مقاله چهار استاندارد مطرح برنامه‌نویسی به زبان C++ به نام‌های MISRA C++، JSF AV C++، HI C++، ESCR C++ مورد بررسی قرار گرفته است؛ که قادر خواهند بود، معیار ایمنی را در کدهای نوشته‌شده در مرحله طراحی کد، افزایش دهند. ازاین‌رو در این مقاله، ابتدا میزان هم‌پوشانی این استانداردها به جهت یافتن جامع‌ترین استاندارد ارزیابی‌شده و در ادامه به میزان غنای این استانداردها از شش جهت قابلیت اطمینان، تعمیر، خوانایی، آزمون‌پذیری، کارایی و ایمنی پرداخته شده است. در پایان ابزارهایی را که توانایی وارسی این استانداردها را در حین تولید دارند، مورد بررسی قرار خواهیم داد.

واژگان کلیدی: استانداردهای کدنویسی ایمن C++، ابزارهای وارسی ایمنی در کد، هم‌پوشانی استانداردهای کدنویسی C++، ایمنی در کدهای C++

۱- مقدمه

سامانه‌های رایانه‌ای به‌عنوان بخش مهمی از زندگی روزانه محسوب می‌شوند و کاربردهای گسترده آن‌ها از وسایل الکترونیکی متحرک گرفته تا کاربردهای پزشکی، فضایی و صنعتی، بر کسی پوشیده نیست. هر یک از این سامانه‌ها بسته به نوع کاربرد و محیط مورد استفاده، نیازمند ویژگی‌های خاصی خواهند بود. ازجمله برخی از مهم‌ترین ویژگی‌ها می‌توان به سرعت بالای پردازش، مصرف انرژی پایین، توان بالا، بهینه‌بودن هزینهٔ ساخت، مساحت، کارایی، اتکاپذیری و رفتار بی‌درنگ اشاره کرد.

سامانه‌های رایانه‌ای در کاربردهای بحرانی به سه دسته اصلی تقسیم می‌شوند که عبارت اند از:

- سامانه‌های بحرانی-ایمن: سامانه‌ای که شکست آن موجب آسیب‌ها و خسارات جانی و مالی و نیز تخریب محیط‌زیست می‌شود. نمونه‌ای از سامانه‌های بحرانی ایمن سامانه کنترلی یک کارخانه تولید مواد شیمیایی است.
- سامانه‌های بحرانی-مأموریتی: سامانه‌ای که شکست آن موجب شکست در اهداف و فعالیت‌های آن خواهد شد. یک نمونه از سامانه‌های بحرانی-مأموریتی عملیات ناوبری یک سفینه فضایی است.

* نویسندهٔ مسؤول

- سامانه‌های بحرانی-تجاری: سامانه‌ای که شکست آن می‌تواند منجر به ازدست‌رفتن هزینه بسیار برای کسب و کارهایی که از آن سامانه استفاده می‌کنند، شوند. یک نمونه از این سامانه‌های بحرانی-تجاری، سامانهٔ حسابداری مشتریان در یک بانک است.

ازآن‌جاکه سامانه‌های نهفته و سامانه‌های رایانه‌ای در کاربردهای بحرانی-ایمن و بحرانی-مأموریتی به‌صورت گسترده‌ای وارد شده‌اند و یا در بسیاری از موارد انتظار می‌رود سال‌ها بدون هیچ خطایی کار کنند و درصورت رخداد خطا خود را ترمیم کنند، مؤلفه‌هایی چون ایمنی، امنیت، قابلیت اطمینان، اتکاپذیری، قابلیت تعمیر و نگه داشت اهمیت به‌سزایی می‌یابند. به‌عنوان نمونه از کاربردهای بحرانی-ایمن می‌توان به سامانه‌های کنترل پرواز، سامانه‌های کنترل نیروگاه اتمی و سامانه‌های تجهیزات پزشکی اشاره کرد. ازکارافتادن و یا ایجاد وقفه در این سامانه‌ها می‌تواند خطرات جبران‌ناپذیری به‌همراه داشته باشد. وقوع خرابی در کاربردهای بحرانی-ایمن می‌تواند جان انسان‌ها را به مخاطره بیندازد و خسارات جبران‌ناپذیر مالی یا آسیب‌های جدی به محیط‌زیست وارد کند. در سامانه‌های بحرانی-تجاری، بروز خرابی پی‌درپی به سبب کارکرد نادرست سامانه، می‌تواند منجر به بی‌اعتبارشدن شرکت سازنده شده و بازار فروش محصولات آینده آن را نیز تهدید کند.

برای افزایش ایمنی در این سامانه‌ها روش‌های گوناگونی وجود دارد که می‌توانند در سطوح مختلف به آن‌ها اعمال شوند. گستره این سطوح می‌تواند شامل اعمال آزمون‌ها و روش‌های تحمل‌پذیری در ابتدای طراحی تا زمان اجرا را در بر گیرد و به‌طور کلی به سه دسته اصلی تقسیم می‌شوند: هنگام کدنویسی و طراحی، بعد از طراحی، بعد از فروش و حین اجرا. از سوی دیگر، همواره پیش‌گیری از وقوع اتفاقات ناگوار از مقابله با آن کم‌هزینه‌تر است؛ بنابراین باید تلاش شود تا نرم‌افزارها به‌صورت پیش‌گیرانه‌ای از قرارگیری در شرایط رخ داد اشکال نجات یابند؛ چون که هزینه برطرف‌کردن اشکال رخداد با پیشروی در مراحل تولید افزایش می یابد. ازاین‌رو یک فعالیت مناسب ایمنی در نرم‌افزار ایجاب می‌کند که به مسئله ایمنی و قابلیت اطمینان در ابتدای چرخه تولید نرم‌افزار پرداخته شود. هم‌چنین درصورتی‌که روش‌های آزمون برای کشف نواقص در مرحلهٔ نخست و در حین طراحی و کدنویسی صورت گیرد، زمان لازم برای رفع آن‌ها وجود خواهد داشت. بدین ترتیب چرخهٔ تولید نرم‌افزار روند روبه‌جلو خواهد داشت. این موارد در دو سطح دیگر (بعد از فروش و حین اجرا) به‌ترتیب زمان بیش‌تری صرف خواهند کرد و منجر به روند رو به عقب در طراحی خواهند شد که نه‌تنها وجود زمان کافی برای رفع نواقص را تضمین نمی‌کند بلکه رفع نواقص نیز در بسیاری از موارد به‌دلیل پیچیده‌ترشدن سطح و روند رو به عقب صورت نخواهد گرفت. ازاین‌رو طی سالیان گذشته استانداردهای مطرح شده‌اند که می‌توانند ایمنی کد را در همان فازهای ابتدایی تولید برقرار سازند. بدین ترتیب می‌توان از آسیب‌های جانی و مالی زیادی که به افراد و شرکت‌ها وارد می‌شود، جلوگیری کرد. از سوی دیگر در دهه‌های اخیر، ابزارهای گوناگونی برای وارسی کدهای برنامه در حین طراحی آن‌ها، ارائه‌شده‌اند. برخی از این ابزارها قابلیت وارسی بر اساس سازگاری با قوانین استانداردهای کد نویسی را نیز دارند.

هدف این پژوهش بررسی استانداردهایی است که رعایت آن‌ها در کد نویسی نه‌تنها موجب می‌شود تمام کدها در یک سبک و قالب مشخص باشند، بلکه منجر به عملکرد درست آن‌ها نیز می‌شود. بدین ترتیب ابتدا به شرح استانداردهای کدنویسی ایمن در زبان C++ خواهیم پرداخت؛ سپس در بخش سه میزان هم‌پوشانی بین استانداردهای مطرح نیز بر اساس قوانین آن‌ها عنوان شده و در بخش چهار ابزارهای رایج و قدرتمند برای وارسی کد معرفی می‌شود. در انتها بر اساس بررسی‌های صورت‌گرفته یک استاندارد و ابزار فراهم‌آورندهٔ آن، در جهت بهبود ایمنی در کدهای نوشته‌شده به زبان C++ معرفی خواهد شد.

۲- استانداردهای کدنویسی ایمن

استانداردهای کد نویسی به برنامه‌نویسان کمک می‌کنند تا کد خود را منطبق با سیاست‌های ایمنی توسعه دهند. به‌عنوان نمونه کدهایی ارائه دهند که عاری از هرگونه نقص و اشکالی باشد که منجر به شکست‌های جدی می‌شوند. هدف قوانین و توصیه‌های یادشده به معنای تعریف یک سبک برنامه‌نویسی C++ است که برنامه‌نویسان را توانمند می‌سازد تا کدهایشان را به‌صورت صحیح، قابل‌اطمینان و قابل ترمیم گسترش دهند.

با تمام این تفاسیر، استانداردها یک محصول خالی از اشکال و به‌طورکامل ایمن را تضمین نمی‌کنند. پایبندی به این استانداردها به برنامه‌نویسان کمک خواهد کرد تا کدهای منظمی ارائه دهند که اشکالات و خطاها را کاهش می‌دهند. معیارهای اصلی برای کاربردهای بحرانی-ایمن به‌اختصار شامل موارد زیر هستند:

- **قابلیت اطمینان و ایمنی**: توانایی سامانه یا عنصری از سامانه، برای عملکرد صحیح تحت شرایط معین در زمان مشخص است.
- **قابلیت انتقال**: قابلیت نرم‌افزار یا برنامه برای انتقال از یک محیط به محیط دیگر، بدون تغییر در برنامه یا نرم‌افزار، است.
- **قابلیت تعمیر و ترمیم**: کد باید به شیوه‌ای نوشته شده باشد که دربردارنده سبکی سازگار، خوانا، ساده برای طراحی و آسان برای اشکال زدایی باشد.
- **قابلیت آزمون‌پذیری**: کد باید طوری نوشته شده باشد که آزمون آن آسان باشد. کمینه‌کردن مشخصه‌های زیر برای هر نرم‌افزاری آن را برای آزمون و تعمیر ساده‌تر خواهد کرد:
  - اندازهٔ کد
  - پیچیدگی کد
  - تعداد مسیرهای ایستا (تعداد مسیرها در قطعه‌ای از کد)
- **کارایی**: توانایی محصول در ارائه کارایی مناسب، بسته به متوسط منابع مصرفی تحت شرایط معین، است.
- **خوانایی**: کد باید به شیوه‌ای نوشته شده باشد که برای خواندن، فهمیدن و درک‌کردن آسان باشد.

دو معیار مهم دیگری که باید در طراحی سامانه‌های نهفته مورد توجه قرار گیرد و استانداردها نیز به‌طورکلی در راستای کمک به بهبود این معیارها ارائه شده‌اند، عبارت‌اند از:

قابلیت استفادهٔ مجدد: طراحی اجزایی با قابلیت استفاده مجدد به‌شدت توصیه می‌شود. این کار می‌تواند توسعهٔ تکراری و افزونه را حذف کند و فعالیت‌های مربوط به آزمون را کاهش دهد درنتیجه مانع افزایش هزینه نیز خواهد بود.

قابلیت تمدید و توسعه‌پذیری: انتظار می‌رود نیازمندی‌ها در طول چرخه حیات محصول تکامل یابند؛ بنابراین یک سامانه باید به روشی قابل توسعه طراحی شود.

بدین ترتیب در طول سالیان گذشته استانداردهای گوناگونی ارائه شده است که برخی یا تمام معیارهای یادشده را برای سامانه‌های مختلف بسته به کاربرد و نوع سامانه ارائه می‌دهند. در ادامهٔ این بخش به شرح چهار استاندارد مهم و پر کاربرد جهانی خواهیم پرداخت. در انتها قوانین هر استاندارد را بر اساس معیارهای شش‌گانهٔ اصلی که به توصیف آن‌ها پرداختیم، دسته‌بندی خواهیم کرد. بدین ترتیب می توان برای اهداف گوناگون استاندارد مناسب‌تری را انتخاب کرد.

۲-۱- MISRA C++

MISRA C (Motor Industry Software Reliability Association guidelines) برای نرم‌افزارهای مبتنی بر وسایل نقلیه ارائه شد که شامل مجموعه کاملی از معیارهایی است که باید در توسعهٔ نرم‌افزارهای نهفته مبتنی بر وسایل نقلیه استفاده شود. این استاندارد به‌طور اختصاصی برای استفاده در سامانه‌هایی که شامل جنبه‌های ایمنی هستند به کار می‌رود. MISRA C++ برای پشتیبانی از برنامه‌های نوشته‌شده به زبان C++ ارائه شد؛ به‌طوری‌که ویژگی‌های زبان C++ را که به‌طور بالقوه غیر ایمن هستند، شناسایی و یک سری قوانین برنامه‌نویسی برای اجتناب از آن‌ها ارائه کرده است. انتخاب زبان، مترجم و ویژگی‌های زبان استفاده‌شده بسته به سطح ایمنی نیازمند توجه زیادی است. یکی از معیارهای توصیه‌شده استفاده از زیرمجموعه‌ای استاندارد، از زبان است که در خودروسازی، هوافضا، ناوبری و صنایع دفاع کاربرد دارند. این استاندارد شامل سه دسته قانون، به‌شرح زیر است:

- **قوانین مورد نیاز**: نیازمندی‌های اجباری برای توسعه‌دهنده، کد C++ باید با این قوانین MISRA مطابقت داشته باشد؛ چون اگر رعایت نشوند، نقص‌ها افزایش خواهند یافت.
- **قوانین پیشنهادی**: قوانینی که به‌طورمعمول باید در نظر گرفته شوند، گرچه وضعیت اجباری ندارند. این به این معنا نیست که می‌توان آن‌ها را نادیده گرفت؛ بلکه باید تا حد امکان اعمال شوند.
- **قوانین مستندسازی**: نیازمندی‌های اجباری برای توسعه‌دهنده، زمانی که ویژگی اشاره شده در کد به کار رفته باشد.

برای پروژه‌هایی که یک استاندارد ایمنی را پشتیبانی و از سطوح گوناگون ایمنی پیروی می‌کنند، تمام قوانین ارائه‌شده در این استاندارد باید بدون توجه به سطح ایمنی اعمال شوند.

۲-۲- JSF AV C++

JSF AV C++ (Joint Strike Fighter Air Vehicle) استانداردی است که با هدف کمک به برنامه‌نویسان در جهت توسعه برنامه‌ها همراه با سیاست‌های نرم‌افزاری بحرانی-ایمن ارائه شده است. به‌طورکلی طبق این استاندارد کد تولیدشده باید شامل ویژگی‌های مهمی چون: ساختار سازگار، قابلیت انتقال، قابلیت ترمیم و نیز قابلیت فهم بالا باشد. این قوانین برای توسعه C++ در صنایع هوایی و نیز دیگر کاربردها و به‌طور خاص برای شناسایی رفتار غیر قابل پیش‌بینی طراحی

شده است. این استاندارد برای صنایع هوافضا نیز گسترش یافته و امروزه با تعداد زیادی از صنایع مطابقت یافته و کنترل و نظارت بر آن‌ها را ترفیع بخشیده است.

این استاندارد شامل ۲۲۱ قانون است و به‌طور کلی مجموعه قوانین این استاندارد به سه دسته اصلی تقسیم می‌شوند:

- دستورهایی تحت عنوان should که به‌عنوان پیشنهادهایی برای بهبود برنامه است و راه‌های توصیه‌شده برای انجام یک کار را پیشنهاد می‌کند.
- دستورهای دوم دستورهای will نام دارند که به‌عنوان قوانین اجباری در نظر گرفته می‌شوند. انتظار می‌رود که این قوانین اعمال شوند اما؛ نیاز به اعتبارسنجی ندارند. این قوانین مربوط به نیازمندی‌های غیر بحرانی ایمن که نیاز به اعتبارسنجی ندارند، باز می‌گردند؛ مانند قرارداد نام‌گذاری.
- دستورهای دسته سوم که شامل دستورهایی تحت عنوان shall هستند. این قوانین جز نیازمندی‌های برنامه هستند که به‌حتم باید اعمال شوند و نیازمند اعتبارسنجی هستند. چه به‌صورت خودکار و چه به‌صورت دستی.

HI C++ -۳-۲

استاندارد HIC++ (High Integrated C++) برای نخستین‌بار در سال ۲۰۰۳ ، حدود یک دهه قبل، بر اساس موضوعات و استانداردهای کد نویسی C++ ارائه و در سال ۲۰۱۳ نسخه تکمیل‌شدهٔ آن ارائه شد. سیاست‌های راهنما برای این استاندارد نیز همانند دو استاندارد پیشین قابلیت انتقال، ایمنی، قابلیت ترمیم و خوانایی است. هدف این استاندارد ارائه کد منبع با بالاترین کیفیت با ترجمه کد منبع با این سیاست‌هاست. این مستند مجموعه‌ای از قوانین را برای تولید کد C++ با بالاترین کیفیت ارائه کرده است. سیاست‌هایی که در این استاندارد به آن‌ها راهنمایی شده، عبارت است از: قابلیت ترمیم، قابلیت انتقال، خوانایی و نیرومندی . در این مستند دلیل هر یک از قوانین یادشده همراه با یک مثال مناسب به‌خوبی توصیف شده است. تمام این قوانین باید به برنامه اعمال شوند؛ مگر این‌که یک انحراف و یا خطا در کد نویسی به‌دلیل کاربرد آن قانون گزارش شود. این استاندارد قوانینی را تصویب کرده است که باید برای محدودکردن استفاده از زبان C++ اعمال شوند؛ بدون این‌که هسته انعطاف‌پذیر آن را تحت تأثیر قرار دهند. بدین شیوه می‌توان برنامه‌هایی ایمن و آسان برای ترمیم داشت. مسائل و مشکلات ایجادشده چه به‌دلیل تنوع مترجم‌ها، سبک‌های مختلف برنامه‌نویسی و نیز جنبه‌های خطرناک و مبهم زبان، ممکن است، موجب شوند بدون اعمال استانداردهای کد نویسی، برنامه‌نویسان کدی بنویسند که مستعد خطا و یا برای دیگر برنامه‌نویسان جهت بهبود و یا ترمیم سخت باشد. ترکیبی از این قوانین باید اعمال شود تا بتوان به جامعیتی بالا در برنامه‌نویسی دست یافت. به‌عنوان مثال مدیریت نیازمندی‌ها و آزمون پوشش[1]. درصورتی‌که این قوانین با دقت بالا اعمال و با فنون برنامه‌نویسی همراه شوند، می‌توانند تولید کد با جامعیت بالا را آسان سازند. هدف این استاندارد کد نویسی، اعمال بهترین عملیات در توسعه C++ به‌وسیله اعمال توصیه‌های سبکی و معنایی برای محدودکردن استفاده از ویژگی‌های زبان برنامه‌نویسی است که ممکن است، منجر به برداشت نادرست و یا خطا شود. این استاندارد کدنویسی به‌صورت منظم مرور شده و به‌روز می‌شود به‌طوری‌که بتوان کد قابل‌اطمینان داشت. استاندارد پایه برای این مستند C++ Standard (ISO/IEC 14882) است. استفاده از رفتارهای نامشخص و تعریف‌نشده طبق آمار به‌دست‌آمده در فهرست ISO C++ غیرمجاز است. این قوانین همچنین عملیاتی را که تعریف شده‌اند، اما منجر به خطا می‌شوند، ممنوع کرده است. این مجموعه قوانین به هیجده دسته کلی شامل قوانین مربوط به عبارات، جملات، تعاریف، طبقه‌ها، کنترل دسترسی اعضا و توابع مخصوص اعضا و ... تقسیم می‌شوند.

ESCR C++ -۴-۲

استاندارد ESCR C++ (Embedded System development Coding Reference) برای نخستین‌بار در سال ۲۰۰۶ با اهداف زیر ارائه و در سال ۲۰۱۳ نسخه به‌روز آن منتشر شد:

- کمینه‌کردن مشکلات رخداده در حین استفاده از نرم‌افزار (قابلیت اطمینان)،
- افزایش قابلیت تحمل در برابر اشکال‌ها (قابلیت ترمیم)
  - آسان‌سازی و قابل‌فهم‌کردن کد،
  - کمینه‌کردن اصلاح در کد داخلی،
  - ساده‌سازی کد برای وارسی و آزمون
- عملیاتی برای انتقال برنامه نرم‌افزار به‌طوری‌که با تغییر از یک محیط به محیط دیگر تا حد امکان کارا باشد و منجر به بروز اشکال نشود. (قابلیت انتقال)

[1] Test Coverage

- کدنویسی که از کم‌ترین زمان و حافظه ممکن استفاده کند.(کارایی)

در این استاندارد قوانین به‌تفکیک در چهار دسته قابلیت اطمینان، قابلیت ترمیم، قابلیت انتقال و کارایی توصیف شده‌اند. قابلیت اطمینان در این استاندارد خود به سه دسته تقسیم می‌شود:

- R1: تعریف متغیرها، اشیا، توابع و غیره استفاده از آن‌ها بسته به نوع و اندازه؛
- R2: استفاده از داده با توجه به محدوده اندازه و نمایش داخلی آن؛
- R3: کدنویسی به‌نحوی که آگاه از اندازه محدوده‌ها باشد.

قابلیت تعمیر نیز به پنج دسته تقسیم می‌شود:

- M1: همواره باید توجه داشت که دیگر برنامه‌نویسان امکان دارد برنامه را بخوانند؛
- M2: کدنویسی به سبکی که می‌تواند اشکالات اصلاح و تغییر را واضح سازد؛
- M3: کدنویسی برنامه‌ها در ساده‌ترین حالت ممکن؛
- M4: کدنویسی در یک سبک واحد؛
- M5: کدنویسی به سبکی که فرایند وارسی و آزمون را ساده کند.

قابلیت انتقال به دو زیر قانون اصلی: ۱-کدنویسی به سبکی مستقل از مترجم کد و ۲- شناسایی محلی از کد که با انتقال مشکل دارد، تقسیم می‌شوند.

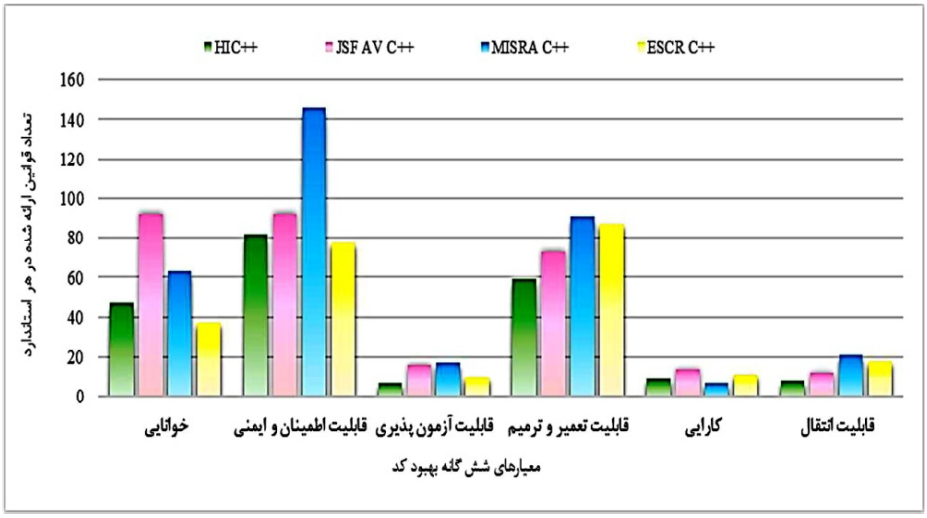
درنهایت کارایی به یک زیرقانون اکتفا می‌کند و آن کدنویسی به سبکی است که کم‌ترین میزان استفاده از زمان و منابع را داشته باشد.

قوانین در این استاندارد در سه دسته اصلی جای می‌گیرند.

- قوانینی که بسته به نوع پروژه می‌توانند مورد استفاده قرار گیرند؛
- قوانین غیر ضروری که از دید کاربران زبان بدیهی هستند؛
- قوانین بسیار ضروری که اگر در نظر گرفته نشوند، کیفیت کد را پایین می‌آورند.

۲-۵- دسته‌بندی قوانین استانداردها بر اساس معیارها

در این بخش تأثیر قوانین هر استاندارد بر معیارهای شش‌گانهٔ مطرح‌شدهٔ ابتدای بخش، یعنی خوانایی، قابلیت اطمینان و ایمنی، قابلیت تعمیر و ترمیم، قابلیت انتقال، قابلیت آزمون‌پذیری و کارایی گردآوری شده است. شکل (۱) نتیجه این گردآوری را بر اساس تعداد قوانین هر استاندارد نسبت به معیارهای شش‌گانهٔ یادشده نشان می‌دهد. با توجه به این شکل می‌توان گفت در زمینه بهبود قابلیت اطمینان و ایمنی، استاندارد MISRA C++ نسبت به سه استاندارد دیگر، قوانین بیشتری ارائه داده است. همچنین در زمینه قابلیت تعمیر و ترمیم، قابلیت انتقال و نیز قابلیت آزمون‌پذیری، این استاندارد نسبت به سه استاندارد دیگر، عملکرد مطلوب‌تری را دارد. در معیار خوانایی و کارایی، JSF از سایر استانداردها پیشی گرفته است؛ بدین ترتیب بسته به نوع پروژه و محدودیت‌های مطرح‌شدهٔ آن، می‌توان استاندارد مناسبی را برای اعمال در برنامه‌ها انتخاب کرد.



(شکل-۱) : تعداد قوانین ارائه شده در هر استاندارد بر اساس معیارهای شش‌گانه بهبود کد

۳- هم‌پوشانی استانداردها

هر کدام از استانداردها قوانینی ارائه می‌دهند که ممکن است، برخی از آن‌ها با قوانین ارائه‌شده در دیگر استانداردها یکسان باشند.

قوانینی که تنها در یک استاندارد ظاهر شده‌اند، تحت تأثیر دو عامل خواهند بود:

- تعداد زیاد قوانین غیرمعمول، فرایند انتخاب و اعمال آن‌ها را طولانی و زمان‌بر خواهد کرد.
- تعداد زیادی از قوانین با تحلیل ایستا قابل‌اعمال نیستند؛ بنابراین نیاز به روش‌های دیگری برای اعمال آن‌ها هست؛ مانند اعمال دستی.

قوانینی که بین چند استاندارد یکسان هستند، برای اعمال با ابزار خودکار کاربردی خواهند بود. به‌عنوان نمونه یکی از قوانینی که بین MISRA C++ و JSF C++ مشترک است؛ به این صورت است که "آرایه‌ها نباید به‌صورت پارامتر ارسال شوند".بنابراین از نقص در کاربرد آرایه اجتناب خواهد شد. برخی قوانین استانداردها به دلایلی قابل اعمال به‌صورت ایستا نیستند. برخی قوانین مربوط به مستندسازی و یا وابسته به فرایند به‌جای کد منبع هستند؛ بنابراین این قوانین قابل خودکارسازی نخواهند بود. به‌عنوان نمونه قانون شماره ۲-۰-۱ در MISRA بدین صورت است که مترجم‌های چندگانه تنها در صورتی باید استفاده شوند که واسط مناسب و تعریف‌شده داشته باشند. در قانون شماره ۲۱۸ در JSF سطوح هشدار در مترجم با سطوح سیاست‌های پروژه منطبق خواهد بود.

در برخی موارد استانداردها شامل قوانین بسیار سنگینی هستند که قابلیت اعمال به‌صورت خودکار را در سطح بالا ندارند. به‌عنوان نمونه در استاندارد MISRA ، قانون ۱-۰-۱۵ بدین معنی است که استثناها فقط باید برای کنترل خطا به کار روند؛ و یا قانون ۲۱۶ در JSF بدین معنی است که برنامه‌نویسان نباید کد را به‌صورت نابه‌هنگام و زودتر از وقت معلوم بهینه کنند.

در حیطه اعمال قوانین، به‌لحاظ ضروری یا غیر ضروری‌بودن قانون‌ها، استانداردها با یکدیگر تفاوت خواهند داشت. علت اصلی طبقه‌بندی قوانین به دسته‌هایی با ضریب ضروریت اعمال متفاوت، این است که درصورت رعایت‌نشدن قوانین ضروری احتمال رخداد خطا در کدها بالاتر خواهد رفت.

با توجه به هم‌پوشانی که این استانداردها ارائه می‌دهند و نیز معیارهایی که هر استاندارد ارائه می‌دهد می‌توان تصمیم گرفت کدام استاندارد طبق آرمان‌ها و اهداف سازمانی مناسب هستند. می‌توان این استانداردها را جمع‌بندی و یک استاندارد جامع مطابق با اهداف سازمان ارائه کرد.

برای این‌که بتوان تحلیل و نتیجه جامعی درباره استانداردها داشت، هم‌پوشانی استانداردها با یکدیگر در شکل (۲) نمایش داده شده است. همچنین برای تحلیل بهتر، میزان هم‌پوشانی هر استاندارد با استانداردهای دیگر نیز در جدول (۱) نمایش داده شده است.

(جدول-۱): هم‌پوشانی قوانین هر استاندارد با دیگر استانداردها

مجموعه استاندارد	میزان هم‌پوشانی کل
MISRA C++	۶۷,۳۹٪
JSFAV C ++	۶۴,۶۹٪
ESCR C++	۶۰,۳٪
HIC++	۵۴,۵۱٪

۴- ابزارهای تحلیل کد

در دهه‌های اخیر، توجه زیادی به امنیت و ایمنی برنامه شده است. به‌طوری‌که تحلیل کد، یک روش استاندارد در معرفی نرم‌افزار ایمن است و خطرات ذاتی نرم‌افزار را کاهش می‌دهد. ابزار تحلیل کد، برای تحلیل کد منبع و یا کد ترجمه‌شده در جهت کمک به پیداکردن نقص‌های ایمنی طراحی شده‌اند. برخی ابزارها برای تعبیه‌شدن در یک محیط IDE طراحی شده‌اند. برای آن دسته از اشکالاتی که می‌توانند در مرحلهٔ طراحی نرم‌افزار کشف شوند، این ابزارها مرحلهٔ قدرتمندی در چرخه تولید نرم‌افزارها فراهم خواهند کرد؛ چون بلافاصله یک بازخورد به توسعه‌دهنده در حین گسترش و نوشتن نرم‌افزار ارائه می‌دهند. این بازخورد بسیار مناسب و کاربردی خواهد بود. به‌خصوص برای آسیب‌هایی که در زمانی بسیار بعدتر از زمان توسعه کد یافت می‌شوند. مزایای استفاده از این ابزارها عبارت‌اند از:

- مقیاس‌پذیری بالا، بدین معنی که می‌توانند روی تعداد زیادی از نرم‌افزارها به‌صورت چندباره اجرا شوند. هم‌چنین امکان استفاده از آزمایش‌های قبلی برای صرفه‌جویی در زمان، در برخی ابزارها موجود است؛
- مناسب برای مواردی که این ابزارها می‌توانند به‌صورت خودکار ضریب برخورد بالا به‌دست آورند؛ مانند سرریز بافر، تزریق نادرست در پایگاه داده و ...؛
- خروجی این ابزارها برای توسعه‌دهنده مفید خواهد بود. چون فایل‌ها و خطوط معیوب و دارای نقص را نشان خواهد داد؛

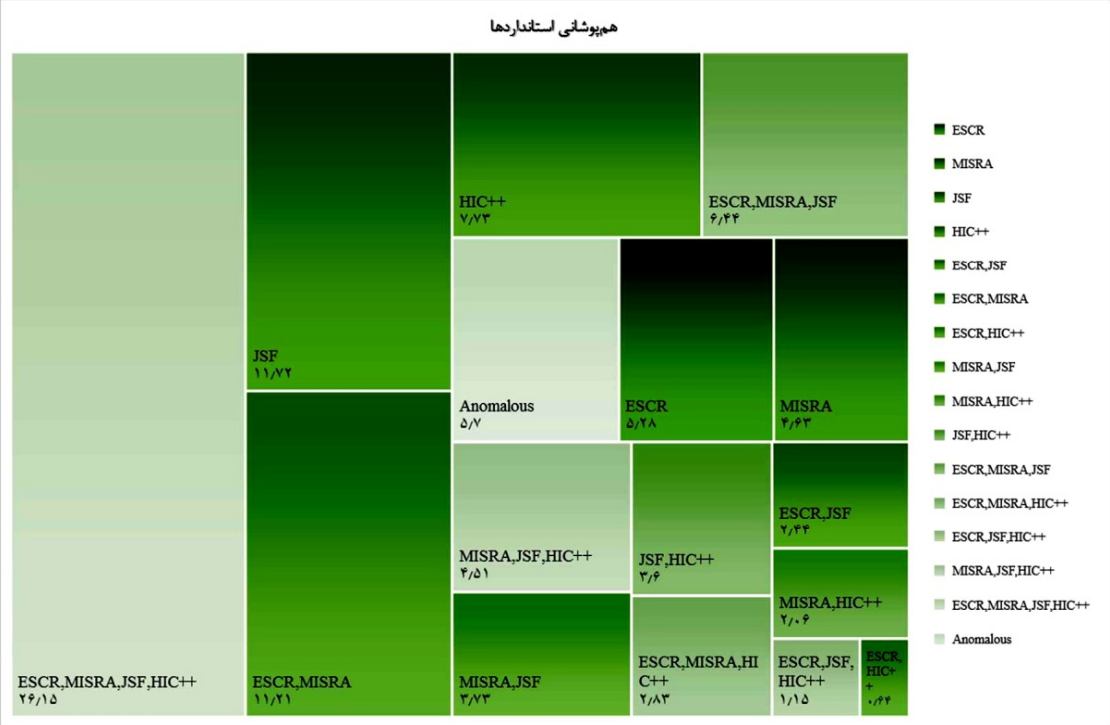
- بررسی خودکار و سریع بسیاری از معیارها در کد بدون اجرای واقعی آن؛
- یک همراه مناسب برای چشم انسان هستند؛ چون برای کشف اشکالاتی که به‌سختی به‌صورت دستی کشف می‌شوند، کار می‌کنند.

معایبی که استفاده از این ابزارها به‌دنبال دارند نیز به‌شرح زیر است:

- بسیاری از آسیب‌های ایمنی برای کشف به‌صورت خودکار بسیار پیچیده هستند؛ مانند احراز هویت، مسائل کنترل دسترسی، رمزنگاری ناامن و... ؛
- تعداد زیاد اشکالات غیر واقعی؛
- خطاهای پیکربندی به‌طورمعمول یافت نخواهند شد؛ چون در کد نشان داده نشده‌اند؛
- مشکل‌بودن اثبات این‌که یک مسئله ایمنی شناخته‌شده به‌طورواقعی یک آسیب‌پذیری واقعی است؛
- بسیاری از این ابزارها کدهایی را که ترجمه نمی‌شوند، به‌سختی تحلیل می‌کنند. تحلیل‌گران به‌طورمعمول نمی‌توانند کدها را ترجمه کنند؛ چون کتابخانه‌های صحیح در دسترس نیستند.

آنچه که در انتخاب یک ابزار مناسب بسیار مهم است و باید به‌عنوان محدودیت‌هایی در انتخاب در نظر گرفته شود، شامل معیارهای زیر است:

- چه نوع آسیب‌پذیری‌هایی را می‌تواند کشف کند؟
- آیا به یک مجموعه از منابع ساخت‌یافته نیاز دارد؟
- می‌تواند تحلیل را به‌جای کد منبع روی کد دودویی انجام دهد؟
- زمان و یا هزینه برای توسعه یادگیری و خصوصی‌سازی این ابزار تا چه اندازه محدود است؟
- توسعه‌دهنده چه اندازه پوشش از ابزار انتظار دارد؟
- هزینه تأیید و تصدیق ابزار چه اندازه است؟
- نرخ خطای ابزار چه میزان است؟
- چه محدوده‌ای از دانش با این ابزار پوشش داده می‌شود؟
- آیا با سایر ابزارهای توسعه‌دهنده در یک راستا هست و در جامعیت با آن‌ها کار می‌کند یا خیر؟



(شکل-۲): هم‌پوشانی استانداردها

(جدول-۲): مقایسه ۱۰ ابزار برتر تحلیل کد

نام ابزار	ارائه‌دهنده	استاندارد پشتیبانی شده				کدهای مورد قبول برای وارسی*				سیستم‌عامل‌های پشتیبانی شده			اشکالات قابل کشف
		MISRA C++	HI C++	JSF AV C++	ESCR C++	Visual Studio	Eclips	CMakc	Jenkins	Windows	Mac	Linux	
LDRA_testbed	LDRA	✓	✓	✓	✓	×	✓	×	×	✓	×	✓	سرریز حافظه، نشتی منابع، سرریز بافر، اشاره‌گر تهی متفاوت، اشکالات هم‌زمانی، اشکالات حسابی، اشکال انتقال، کد مرده، تقسیم‌بر صفر، کتاب‌خانه‌ها ناامن و غیره
ECLAIR	BugSeng	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	سرریز بافر، دسترسی خارج از محدوده به آرایه، اشاره‌گرهای نامعتبر، کدهای افزونه
QAC++	PRQA	✓	✓	✓	×	✓	✓	×	✓	✓	×	✓	اشکالات انتقال، کاربرد، عدم تعریف مناسب و نیز اشکالات بلوک‌های پایه بر اساس استانداردهای معین، سرریز بافر، تقسیم‌بر صفر، کد مرده، کد غیرقابل دسترسی، اشکال جریان داده، اشکالات معنایی
SonarQube	SonarSource	✓	×	×	×	×	✓	×	✓	✓	✓	✓	کدهای افزونه و تکراری، نقض استانداردهای کدنویسی، کدهای پیچیده، اشکالات، توصیه‌ها و معماری و طراحی بالقوه
PolySpace	MathWorks	✓	×	✓	×	×	✓	×	×	✓	✓	✓	اشکالات زمان اجرا، کد غیرقابل‌دسترس، سرریز حسابی، سرریز بافر، تقسیم بر صفر
C++Test	Parasoft	✓	×	✓	×	✓	✓	×	×	✓	✓	✓	نشتی حافظه، سرریز بافر، مشکلات حسابی، تزریق در پایگاه داده، محافظت از داده‌های حساس، دسترسی غیرمجاز به حافظه
Coverity	Synopsys	✓	×	×	×	✓	✓	×	×	✓	✓	✓	عدم مقداردهی اولیه، تزریق نادرست در پایگاه داده، پیکربندی اشتباه امنیتی، تناقض با قوانین، شرایط رقابتی، از کار افتادن برنامه، کارایی نامناسب، دسترسی غیرقانونی به حافظه، تخریب حافظه، سرریز اعداد حسابی، کنترل داده‌های غیر ایمن، عبارات غیر صحیح، قفل‌های مرده، مشکلات جریان کنترل، دسترسی هم‌زمان به داده‌ها، عدم سازگاری در سلسله‌مراتب کلاس‌ها، مشکلات تعمیر، سرریز بافر و غیره
Kiuwan	Optimyth	✓	×	×	×	×	✓	×	✓	✓	✓	✓	کدهای افزونه، اشکالات در ارسال داده، ناسازگاری‌ها، عبارات غیرقابل انتقال و تعاریف استفاده نشده و متغیرهایی که به‌صورت محلی تعریف نشده‌اند.
Semmle	Oxford University	✓	×	×	×	✓	✓	×	×	✓	✓	✓	از کار افتادن سیستم، تخریب حافظه، آسیب امنیتی، فاش شدن اطلاعات، تقسیم بر صفر، اشاره‌گر نادرست، سرریز بافر، کد غیرقابل دسترسی، دسترسی و یا رونوشت غیرقانونی به حافظه
Understand	Sci Tools	✓	×	×	×	✓	✓	✓	×	✓	✓	✓	سرریز بافر، ورودی نامعتبر، تزریق نادرست در پایگاه داده، تزریق مسیر یا فایل نادرست، نشت اطلاعات، رمزگذاری ضعیف، عملیات کدنویسی آسیب‌پذیر، نقض محدوده آرایه‌ها

* ابزارها محیط های دیگری را نیز پشتبانی می کنند که پرکاربردترین این محیط ها در جدول یادشده است.

۱- متن‌باز (برای نسخه C++ هزینه برابر است با $۷۰۰۰)

۲- برای استفاده از این ابزار کدها باید روی یک cloud قرار گیرند.

۳- نسخه آزمایشی و نیز نسخه crack شده موجود است.

با توجه به معیارهای یادشدهٔ بالا برای انتخاب ابزار در تحلیل برنامه‌ها، مجموعه‌ای از ابزارهای ارائه‌شده در این زمینه با یکدیگر مقایسه شده‌اند. نتیجه این مقایسه در جدول (۲) یادشده است.

این ابزارها قادر به کشف اشکالات بسیاری که در اثر کدنویسی ظاهر می‌شوند، می‌باشند. برخی از این اشکالات عبارت‌اند از: سرریز بافر، تقسیم‌بر صفر، کد مرده، کد غیرقابل دسترسی، اشکال جریان داده، اشکالات معنایی، نشتی حافظه، مشکلات حسابی، محافظت از داده‌های حساس، دسترسی غیرمجاز به حافظه، عدم مقداردهی اولیه، تزریق نادرست در پایگاه داده، پیکربندی اشتباه امنیتی، تناقض با قوانین، ازکارافتادن برنامه، کارایی نامناسب، دسترسی غیرقانونی به حافظه، تخریب حافظه، سرریز اعداد حسابی، کنترل داده‌های غیر ایمن، عبارات غیر صحیح، قفل‌های مرده، مشکلات جریان کنترل، دسترسی هم‌زمان به داده‌ها، عدم سازگاری در سلسله‌مراتب طبقه‌ها، مشکلات تعمیر، نوشتن در حافظه فقط خواندنی، خواندن متغیرهای تعریف‌نشده، شکست در انتساب مقدار بازگشتی از زیر روال و غیره.

۵- نتیجه‌گیری

با توجه به آن‌چه تاکنون یادشده است انتخاب یک استاندارد مناسب برای هم‌شکل‌کردن کدهای یک شرکت و یا یک پروژه، ارتباط به‌طورکامل مستقیمی با اهداف شرکت و یا پروژه دارد. طبق بررسی‌های صورت‌گرفته استانداردهای MISRA C++، ESCR C++، HI C++ و JSF++ استانداردهای بسیار پرکاربرد در حوزهٔ کدنویسی برای سامانه‌های بحرانی-ایمن و نیز نهفته به حساب می‌آیند. در جدول (۳) خلاصه‌ای از استانداردهای مطرح‌شده در بخش‌های پیشین قابل مشاهده است.

همان‌طورکه پیش از این یاد شد، استانداردی مانند JSF بسیار اختصاصی‌تر و به عنوان استانداردی برای کدنویسی در سامانه‌های بحرانی_ایمن هوافضایی ارائه شده است. استانداردهایی مانند ESCR و HIC++ بسیار عمومی‌تر هستند و طیف وسیعی از کاربردها را چه به‌صورت بحرانی و چه به‌صورت غیر بحرانی در بر می‌گیرند. ازاین‌رو به‌دلیل کاربرد گسترده‌تر و رایج استاندارد MISRA C++ و نیز دامنه کاربرد آن که سامانه‌های بحرانی (چه نهفته و چه غیر نهفته) را در بر می‌گیرد، این استاندارد در این مقاله مورد توجه بیشتری قرار گرفته است. دلیل دیگر انتخاب این استاندارد این است که استاندارد MISRA به‌عنوان پایه و اساسی برای سه استاندارد دیگر نیز به کار رفته است. بدین معنی که هر سه استاندارد دیگر از نسخه‌های گوناگون استاندارد MISRA بهره برده و در بسیاری از قوانین خود، به این استاندارد ارجاع داده‌اند. از سوی دیگر همان‌طورکه در شکل (۲) مشاهده می‌شود، قوانین موجود در این استاندارد به‌خوبی با قوانین دیگر استانداردها هم‌پوشانی دارند. به‌عبارت دقیق‌تر همان‌طورکه در جدول (۱) نشان داده شده است، میزان هم‌پوشانی کل برای قوانین استاندارد MISRA با دیگر قوانین در مجموعه قوانین چهار استاندارد برابر ٪۶۷٫۳۹ است که درصد هم‌پوشانی بالایی نسبت به سایر استانداردها به حساب می آید؛ بنابراین با انتخاب این استاندارد بخش قابل توجهی از قوانین دیگر استانداردها نیز تحت پوشش قرار خواهد گرفت.

(جدول-۳): مروری بر استانداردهای مطرح جهانی

استاندارد	سال ارائه آخرین نسخه	تعداد قوانین	انواع قوانین	کاربرد
MISRA C++	۲۰۰۸	۲۲۷	ضروری پیشنهادی مستندسازی	سامانه‌های بحرانی
JSF AV C++	۲۰۰۵	۲۲۱	*shall should will*	سامانه‌های هوافضایی
HIC++	۲۰۱۳	۱۵۵	---	افزایش کیفیت کدهای *C++*
ESCR C++	۲۰۱۳	۱۶۷	وابسته به نوع پروژه بدیهی بسیار ضروری	سامانه‌های نهفته

پس از انتخاب استاندارد برگزیده نیازمند ابزاری خواهیم بود تا بتواند روند تجزیه و تحلیل کد را بر اساس استاندارد انتخابی، آسان و دقیق سازد. ابزارهای گوناگونی برای وارسی کدهای برنامه در حین طراحی و نوشتن کد ارائه شده‌اند. برخی از این ابزارها قابلیت وارسی بر اساس سازگاری با قوانین استانداردهای کدنویسی را نیز دارند. در بخش پیشین، ده مورد از ابزارهای برتر در این زمینه که کاربردهای فراوان و مهمی در سطح جهانی دارند مورد بررسی قرار گرفتند. از آنجا که این ابزارها بسیار قدرتمند بوده و در زمینه‌های حساس و وسیع مورد استفاده قرار می‌گیرند، اغلب با مبالغ بالایی به فروش می‌رسند. از سوی دیگر برخی ابزارها تنها در محدوده‌های خاص جغرافیایی-کشوری ارائه می‌شوند. بدین ترتیب، با توجه به بررسی‌های صورت‌گرفته و شرایط موجود، برای بررسی استاندارد MISRA C++ بهترین ابزار در دسترس، ابزار Understand از شرکت Sci Tools است. Understand در سال ۱۹۹۶ توسط شرکت Sci Tools ارائه شد. ابزار Understand، یک ابزار تجزیه‌وتحلیل ایستای کد است که بر درک کد منبع، متریک‌ها و آزمون‌های استاندارد تمرکز دارد. این ابزار برای کمک به تعمیر، نگهداری و درک

کدهای بزرگ طراحی شده است و یک پلتفرم عبوری چندزبانه و مبتنی بر تعمیر و نگهداری ارائه می‌دهد. این ابزار:

- کدهای پیچیده را به آسانی تعمیر می‌کند؛
- معماری کد منبع را برای بهینه‌سازی طراحی نرم‌افزار تصویرسازی می‌کند؛
- کد منبع مستندنشده را قابل درک می‌سازد؛
- مهندسی معکوس و تجزیه‌وتحلیل اثرات را برای کدهای تغییریافته فراهم می‌آورد؛
- نرم‌افزار برای بررسی جریان اجرا و نحوه عملکرد بررسی می‌کند؛
- مرور و حرکت در کد را امکان‌پذیر می‌سازد؛
- مستندات خودکار را همانند مستندات واردشده ایجاد می‌کند؛
- تجزیه‌وتحلیل ایستا را برای کد منبع متن‌باز یا کد شخص ثالث فراهم می‌آورد؛
- ابزارهای تجزیه‌وتحلیل کد موروثی برای درک کد منبع را ارائه می‌دهد؛
- از ویرایش‌گر چندابزاری قدرتمند بهره می‌برد؛
- اشکال‌زدایی کدهای بحرانی-ایمن را هنگامی که ابزارهای اشکال‌زدایی سنتی با شکست مواجه می‌شوند، انجام می‌دهد؛
- پشتیبانی گسترده‌ای از زبان‌ها فراهم می‌آورد.

کد منبع برای استفاده از این ابزار، می‌تواند به هر یک از زبان‌های زیر نوشته شود:

C, C++, C#, Objective C/Objective C++, Ada, Java, Pascal/Delphi, COBOL, JOVIAL, VHDL, FORTRAN, PL/M, Python, PHP, HTML, CSS, JavaScript , XML.

همچنین دارای ویژگی‌های منحصربه‌فردی است که آن را نسبت به سایر ابزارهای مطرح‌شده متمایز می‌کند. این ویژگی‌ها به گزارش‌های ارائه‌شده توسط این ابزار بر اساس استاندارد منتخب باز می‌گردد.

۶- جمع‌بندی

یک مهارت کلیدی برای ارزیابی کد و شناسایی سریع اشکالات به‌منظور جلوگیری از بروز مسائل احتمالی قبل از رویداد آن‌ها، اگرچه که هنوز رخ نداده‌اند، شناسایی مشکلات رایج کدنویسی است. بسیاری از این اشکالات به‌صورت تکراری در برنامه ظاهر می‌شوند. پس از شناسایی اشکالات رایج باید برای مقابله با آن‌ها اقداماتی صورت گیرد. این اقدامات می‌توانند در مراحل مختلف طراحی و تولید محصول تا ارائه آن به بازار مشتری و استفاده از محصول ادامه یابند. لیکن هزینه و زمان لازم برای مقابله با اشکالات در طول این مراحل به‌مرور افزایش می‌یابد؛ به‌طوری که اگر اشکالات در مرحله تولید هم کشف شوند، نه‌تنها هزینه زیادی برای رفع آن‌ها مورد نیاز خواهد بود، بلکه امکان دارد، زمان لازم برای برطرف‌کردن آن اشکالات نیز در اختیار نباشد. ازاین‌رو هر قدر زمان کشف و برطرف ساختن اشکالات به زمان طراحی و پیش از تولید نزدیک‌تر باشد از نظر زمانی و هزینه مقرون‌به‌صرفه‌تر خواهد بود.

هدف این مقاله، افزایش ایمنی سامانه‌های رایانه‌ای با به‌کارگیری استانداردهای مطرح‌شده برای کدنویسی ایمن است. استانداردهای کدنویسی به برنامه‌نویسان کمک می‌کنند تا کد خود را منطبق با سیاست‌های امنیتی و به‌صورت مجموعه‌ای واحد توسعه دهند. به‌عنوان نمونه کدهایی که عاری از هرگونه نقص و اشکالی که منجر به شکست‌های جدی که در نتیجه آن‌ها آسیب‌های زیادی به افراد و شرکت‌ها وارد می‌شود، باشد. علاوه‌براین ازآنجاکه در حین کدنویسی و طراحی برنامه‌ها و پیش از تولید اعمال می‌شوند با هزینه و صرف زمان کم‌تری از وقوع اشکالات تکراری و شناخته‌شده پیش‌گیری می‌کنند. مورد دیگری که در این مقاله به آن پرداخته شد، بررسی چند مورد از برترین ابزارهای جهانی در زمینه آزمون خودکار برنامه‌های نوشته‌شده به زبان C++ بر اساس این استانداردها بود. بدین ترتیب بر اساس معیارهای مطرح‌شده برای بهبود کیفیت کد و نیز گسترهٔ کاربرد، استانداردی به‌عنوان استاندارد مرجع انتخاب شد و در همین راستا برای اعمال دقیق‌تر و آسان‌تر قوانین این استاندارد، ابزاری که بتوان به آن دسترسی آسان‌تری نیز داشت، معرفی شد.

۷- مراجع

[1] P. Lokuciejewski, P. Marwedel, "Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems", Germany, Springer, 2011.

[2] Kaelbling, Leslie P. "A formal framework for learning in embedded systems." Proc. 6th Intern. Workshop on Machine Learning. 2016.

[3] Yim, Keun Soo. "Fault tolerance model, methods, and apparatuses and their validation techniques." U.S. Patent No. 9,317,254. 19 Apr. 2016.

[4] Seacord, Robert C. The CERT C Coding Standard: 98 Rules for Developing Safe, Reliable, and Secure Systems. Pearson Education, 2014.

[5] Martins, Luiz Eduardo G., and Tony Gorschek. "Requirements engineering for safety-critical systems:

[6] A systematic literature review." Information and Software Technology 75 (2016): 71-89.

[7] Smith, David J., and Kenneth GL Simpson. The Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety: IEC 61508 (2010 Edition), IEC 61511 (2015 Edition) and Related Guidance. Butterworth-Heinemann, 2016.

[8] Scott Meyers. Effective C++: 50 Specific Ways to Improve Your Programs and Design, 2nd Edition. Addison-Wesley, 1998

[9] MISRA C++:2008 Guidelines for the use of the C++ language in critical systems, June 2008, MIRA Limited

[10] Joint Strike Fighter Air Vehicle C++ Coding Standards, December 2005, Lockheed Martin Corporation

[11] High Integrity C++ Coding Standard Manual - Version 4.0, October 2013, Programming Research

[12] Embedded System Development Coding Reference C++, Version 1.0, March 2013, Software Engineering Center

[13] http://www.ldra.com

[14] http://bugseng.com/products/eclair

[15] http://www.programmingresearch.com

[16] http://www.sonarqube.org

[17] http://www.polyspace.com

[18] https://www.parasoft.com

[19] http://www.coverity.com

[20] https://www.kiuwan.com

[21] https://semmle.com

[22] https://scitools.com

محمد محسن امیری درجهٔ کارشناسی ارشد خود را در رشتهٔ مهندسی کامپیوتر گرایش نرم‌افزار از دانشگاه آزاد علوم و تحقیقات در سال ۱۳۹۷ اخذ کرد. وی از سال ۱۳۸۹ در عناوینی همچون دبیر دانشجویی المپیاد رباتیک دانشجویی پیام‌نور، دبیرعلمی بزرگ‌ترین مسابقات رباتیک آموزش پرورش و دبیر انجمن رباتیک کل دانشگاه پیام نور فعالیت داشته است. از ایشان سه کتاب در حوزه‌های مهندسی نرم‌افزار به چاپ رسیده است. برنامه‌نویسی، رباتیک، نرم‌افزارهای تحمل‌پذیر خطا، مهندسی معکوس، سامانه‌های هوشمند و برنامه‌نویسی ایمن ازجمله زمینه‌های پژوهشی ایشان است.

مرتضی معمر درجهٔ کارشناسی خود را در رشتهٔ مهندسی برق مخابرات گرایش فناوری اطلاعات و ارتباطات از دانشگاه قم در سال ۱۳۸۹ و همچنین درجه کارشناسی ارشد خود را در رشته مهندسی فناوری اطلاعات گرایش تست نرم‌افزار از دانشگاه ایوانکی در سال ۱۳۹۷ اخذ کرد. از ایشان بیش از سی مقاله علمی در کنفرانس‌ها و مجلات داخلی و بین‌المللی به چاپ رسیده است. تولید و استقرار محصولات نرم‌افزاری، امنیت سخت‌افزار، برنامه‌نویسی ایمن، امنیت اطلاعات، تست و تضمین کیفیت نرم‌افزار و تست برنامه‌های کاربردی موبایل از جمله زمینه‌های پژوهشی ایشان است.

موسی محمدنیا درجه کارشناسی خود را در رشته مهندسی برق گرایش الکترونیک از دانشگاه جامع امام حسین (ع) در سال ۱۳۷۷ و همچنین درجه کارشناسی ارشد خود را در رشته مهندسی صنایع از دانشگاه جامع امام حسین (ع) در سال ۱۳۹۷ اخذ کرد. از ایشان بیش از بیست مقاله علمی در کنفرانس‌ها و مجلات داخلی و بین‌المللی و نیز چهار کتاب در حوزه‌های مهندسی کامپیوتر به چاپ رسیده است. تحلیل پروتکل‌های امنیتی، تئوری اطلاعات، سیستم عامل، برنامه‌نویسی سامانه‌های بی‌درنگ و نهفته، شبیه‌سازی، مدل‌سازی و امنیت از جمله زمینه‌های پژوهشی ایشان است. وی در حال حاضر نیز به‌عنوان مربی در دانشگاه تدریس می‌کند.

مسعود عسگری مهر درجهٔ کارشناسی ارشد خود را در رشتهٔ مهندسی فناوری اطلاعات گرایش مدیریت سامانه‌های اطلاعاتی از دانشگاه علوم و فنون مازندران در سال ۱۳۹۰ و همچنین درجه دکترای خود را در رشتهٔ مهندسی فناوری اطلاعات گرایش ITSM از دانشگاه علامه طباطبائی در سال ۱۳۹۶ اخذ کرد. وی در سال‌های ۱۳۹۳ تا ۱۳۹۴ به‌عنوان یکی از مدیران راهبردی در اداره کل فناوری اطلاعات بانک ملت فعالیت داشته است. نامبرده در حال حاضر استادیار دانشکده کامپیوتر و فناوری اطلاعات دانشگاه ایوانکی است. از ایشان بیش از پانزده مقاله علمی در کنفرانس‌ها و مجلات بین‌المللی به چاپ رسیده است. تولید و استقرار محصولات نرم‌افزاری، معماری سازمانی، مدیریت ریسک، تست نرم‌افزار، و تضمین کیفیت نرم‌افزار از جمله زمینه‌های پژوهشی ایشان است.