بررسی اشکالات نرم افزار روی برد ماهواره نمونه

شاهرخ جلیلیان[۱*]، فاطمه سالار کالجی[۲]، علیرضا خانی[۳]، ابوالفضل دیانی[۴]، سید علیرضا عمرانیان[۵]، فرهاد باقر اسکویی[۶]

۱- مربی، پژوهشگاه فضایی ایران، پژوهشکده سامانه های ماهواره، sh.jalilian@isrc.ac.ir

۲،۳،۴،۵،۶- پژوهشگر، پژوهشگاه فضایی ایران، پژوهشکده سامانه های ماهواره

*نویسنده مخاطب

چکیده

توسعه نرم افزار بخش فضایی (نرم افزار روی برد) برای سامانه های فضایی از جمله ماهواره ها به دلیل ویژگیهای منحصر به فرد در محیط عملیاتی و محدودیتهای دسترسی به سامانه در زمان عملیاتی، دارای چالشهای منحصر به فردی هست. به علاوه معمولا سامانه های فضایی داری ویژگی بحرانی-ماموریتی نیز هستند. لذا سامانه های فضایی لازم است از اتکاپذیری نسبتا بالایی برخوردار باشند. سهم نرم افزار روی برد در تامین اتکاپذیری آن بسیار بالاست. به همین منظور لازم است در فرآیند توسعه نرم افزار روی برد (به ویژه فرآیند پیاده سازی) بخشهایی که آسیب پذیرتر هستند شناسایی شود و با ملاحظات لازم تولید شوند. در پژوهش حاضر دو پروژه فضایی مورد بررسی قرار گرفت و از دو جنبه به آنها پرداخته شد. در جنبه نخست، مطالعات میدانی با نظرسنجی از توسعه دهندگان و ذینفعانی که یا مستقیم و یا غیر مستقیم در تولید محصول نهایی نرم افزار روی برد نقش داشتند انجام گرفت. در یک جنبه دیگر، نرم افزارهای مورد بررسی، تحت آزمون تحلیل استاتیک با استفاده از نرم افزار آزمونگر خودکار قرار گرفتند که نتایج میدانی به دست آمده را به خوبی تایید کردند.

واژه های کلیدی: *نقص نرم افزار، آزمون نرم افزاری، بررسی میدانی، آزمون استاتیک، نرم افزار روی برد*

۱- مقدمه

در تاریخچه نقش نرم افزار سامانه های فضایی دو مورد بیش از بقیه به چشم می خورد که تاثیر بروز نقصهای نرم افزاری در موفقیت یا شکست ماموریت فضایی را به روشنی نشان می دهد. از دست دادن مدارگرد آب و هوای مریخ در سال ۱۹۹۹ و انفجار موشک آریان ۵ در حین پرتاب در سال ۱۹۹۶ دو رویداد مهمی بود که در پی بروز نقص نرم افزاری رخ داد.

مدارگرد آب و هوای مریخ یکی از دو کاوشگری بود که مأموریت آنها مطالعات هواشناسی و سنجش میزان آب و دی اکسید کربن سیاره مریخ بود. فضاپیما در ۱۱ دسامبر ۱۹۹۸ از کیپ کاناورال در فلوریدا پرتاب شد و در سپتامبر ۱۹۹۹ به مریخ رسید. پس از مانور اولیه که امکان قرار دادن در مدار را فراهم می کرد، قرار بود فضاپیما تماس رادیویی را دوباره برقرار کند. با زمین چند دقیقه بعد، اما این هرگز اتفاق نیفتاد. تحقیقاتی که پس از آن انجام شد مشخص کرد که کاوشگر در هنگام ورود به جو به دلیل ضریب تبدیل نادرست برای مقدار تغییر سرعت به سطح مریخ برخورد کرده است. نرم افزار روی برد و نرم افزار ایستگاه زمینی از دو سیستم اندازه گیری متفاوتی (سیستم بریتانیا و سیستم متریک) استفاده می کرد.

در موشک آریان ۵ به دلیل نقص نرم افزاری باعث بروز خطا در محاسبه موقعیت و سرعت پرتابگر در حدود ۳۰ ثانیه پس از پرتاب شد. مقدار موقعیت اشتباه باعث شد که رایانه داخلی دستوراتی را برای تغییر جهت پرتابگر به موتورها ارسال کند. به طوری که سازه در معرض تنش های زیادی قرار گرفت. ماژول نرم افزاری مربوطه به طور خاص برای آریان ۵ طراحی نشده بود بلکه از آریان ۴ گرفته شده بود. در نسخه قبلی برای ذخیره سازی یک متغیر تاثیرگذار از محل حافظه ۱۶ بیتی استفاده شده بود در حالی که محاسبات آریان ۵ نیاز به متغیر ۶۴ بیتی داشت. این در حالی است که برای صرفه جویی در هزینه، از آزمونهای بیشتر هم صرف نظر شده بود که در نهایت منجر به انفجار موشک شد.

علاوه بر دو مورد مذکور، در پروژه ماه نورد آپولو هم شرایط پیش بینی نشده ای پیش آمده که این بار بر خلاف دو رویداد قبلی، با واکنش درست نرم افزار، از شکست ماموریت آن جلوگیری شد. [1] اینها موردهایی بودند که به خوبی نقش و اهمیت نرم افزار در انجام ماموریت سامانه های فضایی را نشان می دهند. خرابی ناشی از نرم افزار به دلیل بروز خطای نرم افزاری رخ می دهد و خطای نرم افزاری می تواند منشا گوناگونی داشته باشد. در ادامه ابتدا به بررسی میدانی از نقش آفرینان حوزه نرم افزار در دو پروژه فضایی در خصوص نقصهای نرم افزاری پرداخته می شود و سپس نتایج حاصل از آزمونهای استاتیک خودکار مورد بررسی قرار می گیرد و در انتها جمع بندی بحث و بررسی های انجام شده ارائه می گردد.

۲- بررسی میدانی نقصهای محتمل

منشا بروز خرابی های سامانه های فضایی که ناشی از بد عمل کردن نرم افزار باشد، بروز خطاهای نرم افزاری است. خطای نرم افزار می تواند از نقصهای نرم افزاری یا نقصهای سخت افزاری ایجاد شود. نقصهای سخت افزاری در اینجا موضوع بحث ما نیست. اما نقصهای نرم افزاری خود ریشه های مختلفی دارد از جمله، نقصهای تحلیل (الزامات)، طراحی و پیاده سازی که ناشی از خطای تحلیل گر، طراح و پیاده ساز است. همچنین ذینفعان دیگری هم هستند مانند اپراتور، ناظر، کارفرما که با انجام کار نادرست و یا انجام ندادن کار اصلاحی به موقع، ممکن است در ایجاد نقص تاثیرگذار باشند. لذا برای بررسی نقصها به صورت میدانی از افراد با نقشهای مختلف که در دسترس بودند نظرسنجی و جمع آوری اطلاعات انجام شده است.

۲-۱- پرسشنامه

برای طراحی سوالات نظرسنجی، دو نوع سوال تهیه شد که بخش اول سوالات برای تمام مخاطبین و بخش دوم صرفا برای افراد درگیر با توسعه نرم افزار بود. سوالات به صورت انتخاب گزینه ای و در فرم الکترونیکی طراحی شد که مخاطبین امکان افزودن گزینه جدید را هم داشتند. گزینه‌ها در فرم الکترونیکی به صورت انتخاب از لیست بودند که امکان گزیش به ترتیب اولویت و اهمیت هم برای افراد مشارکت کننده وجود داشت. سوالات در زیر آورده شده است.

۱. نقش شما در ارتباط با زیرسیستم نرم افزار کدام مورد بود؟

تحلیلگر، طراح، پیاده کننده، آزمونگر، مدیر ذیربط

۲. بیشترین مشکلات (مانند باگ، ناپایداری و ...) و صرف زمان را در کدام یک از موارد زیر داشتید؟

الزامات نرم افزار و تحلیل سیستمی مرتبط با نرم افزار، طراحی سطح بالا (معماری)، طراحی دقیق، پیاده سازی، آزمون واحد، آزمون تجمیع زیرسیستمی، آزمون تجمیع سیستمی، آزمون پذیرش کارفرما

۳. در کدام یک از موارد زیر نیاز به تکمیل و تقویت را مشاهده کرده اید؟

فعالیتهای مهندسی سیستم (الزامات، یکپارچه سازی ...)، تامین به موقع سخت افزار و بستر توسعه، تهیه تجهیزات و ابزارهای کارآمد و به روز برای توسعه و آزمون، آموزشهای فنی، انسانی برای افراد ذیربط، تکمیل نیروی انسانی ذیربط، تامین بستر شبیه ساز برای توسعه نرم افزار، فعالیتهای PMO، فعالیتهای تضمین محصول

۴. مهمترین چالشهای شما در رابطه با صحه گذاری و اعتبار سنجی (آزمون) نرم افزار روی برد چه مواردی بودند؟

توسعه الگوریتم زیرسیستمها، توسعه درایورهای سخت افزاری، توسعه مدیریت مودها، توسعه سرویسهای پایه (لایه میانی و مستقل از سخت افزار و ماموریت)

سوالهای بعدی تنها برای افرادی که در طراحی و پیاده سازی نرم افزار نقشی داشته و یا اطلاعات دقیق از طراحی و پیاده سازی داشته اند، طراحی شده است:

۵. مهمترین چالشها در رابطه با توسعه (طراحی و پیاده سازی) نرم افزار روی برد چه مواردی بودند؟

توسعه الگوریتم زیرسیستمها، توسعه درایورهای سخت افزاری، توسعه مدیریت مودها، توسعه سرویسهای پایه (لایه میانی و مستقل از سخت افزار و ماموریت)

۶. بیشترین و مهمترین باگهای کشف شده در نرم افزار از کدام یک از انواع زیر بودند؟

uninitialized memory, null pointer dereferencing, division by zero, memory and resource leak,
buffer overflows, dead code, duplicated code

۲-۲- نتایج نظرسنجی

۶ نفر از ذینفعان در نظرسنجی مشارکت داشتند که نتایج آنها برای یکی از سه پروژه در جدولهای ۱ تا ۶ آورده شده است. جدول ۶ مهمترین آمار این نظرسنجی است که نمودارش در شکل ۱ آورده شده است.

جدول ۱- آمار پاسخ گویی به سوال ۱ (نقش مخاطب)

	Frequency	Percent
پیاده کننده	3	50.0
آزمونگر	2	33.3
ناظر	1	16.7
Total	6	100.0

جدول ۲- آمار پاسخ گویی به سوال ۲ (بیشترین مشکل)

	Frequency	Percent
الزامات نرم افزار و تحلیل سیستمی	2	33.3
طراحی سطح بالا یا معماری	2	33.3
پیاده سازی	1	16.7
آزمون تجمیع سیستمی	1	16.7
Total	6	100.0

جدول ۳- آمار پاسخ گویی به سوال ۳ (نیاز به تکمیل)

	Frequency	Percent
فعالیتهای مهندسی سیستم	5	83.3
تجهیزات و ابزارهای کارآمد	1	16.7
Total	6	100.0

جدول ۴- آمار پاسخ گویی به سوال ۴ (چالش صحه‌گذاری)

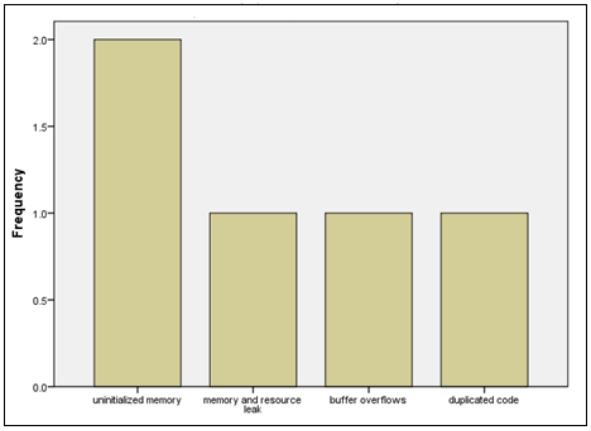
	Frequency	Percent
توسعه درایورهای سخت	1	16.7
افزاری	2	33.3
توسعه مدیریت مودها	2	33.3
توسعه سرویسهای پایه	1	16.7
Missing	6	100.0
Total		

جدول ۵- آمار پاسخ گویی به سوال ۵ (مهمترین چالش توسعه)

	Frequency	Percent
توسعه الگوریتم زیرسیستمها	1	16.7
توسعه مدیریت مودها	2	33.3
توسعه سرویسهای پایه	2	33.3
Missing	1	16.7
Total	6	100.0

جدول ۶- آمار پاسخ‌گویی به سوال ۶ (بیشترین باگ کشف شده)

	Frequency	Percent
uninitialized memory	2	33.3
memory and resource leak	1	16.7
buffer overflows	1	16.7
duplicated code	1	16.7
Missing	1	16.7
Total	6	100.0



شکل ۱- آمار نموداری از پاسخ "بیشترین باگ کشف شده"

۳- آزمون استاتیک نرم‌افزار

آزمون استاتیک یکی از تکنیکهای آزمون نرم افزاری جعبه سفید است. این آزمون مبتنی بر دانش و منطق داخلی کد برنامه بوده که برای بررسی جریان ورودی-خروجی و بهبود طراحی، قابلیت استفاده و امنیت، انجام می شوند. برای انجام این نوع تست، سورس کد نرم افزار برای بررسی نقصهای نرم افزاری (باگها) بدون اجرای کد تحلیل می شود. آزمون استاتیک در استانداردهای ذیربط نرم افزار مانند ISO26262 و DO-178B/C هم الزام یا توصیه شده است. هدف آزمونهای استاتیک، تشخیص نقصها، اشکالات امنیتی و مطابقت دادن با استانداردهای کدنویسی مثل MISRA هست. نقصهایی که مورد پایش قرار می گیرند بیشتر از نوع سرریز حافظه، اشاره‌گر بدون تخصیص حافظه، نشتی حافظه و تهدیدهای امنیتی مانند تزریق فرمانهای SQL است [2].

نرم افزارهای مختلفی وجود دارد که سورس کد نرم افزار مورد آزمون را گرفته و پس از تحلیل، گزارشهای جامعی از باگهای احتمالی را تولید می کنند. از جمله این نرم افزارها می‌توان به Raxis، SonarQube، IBM Rational Software Analyzer ، Parasoft و PVS-Studio اشاره کرد[2,3]. بعضی از ابزارها این امکان را می دهند که به محیط کد نویسی برنامه نویس متصل شده و همزمان به نوشتن کد، هشدار و توصیه های لازم را برای نوشتن کد تمیز (clean code) و جلوگیری از ایجاد نقص به کاربر ارائه می دهند [4].

از مزیتهای آزمون تحلیل استاتیک این است که از مراحل اولیه پیاده سازی کد نرم افزار می توان آن را به کار بست. در حالی که آزمونهای کارکردی (آزمونهای جعبه سیاه) در مراحل پایانی پیاده سازی و تجمیع نرم‌افزار قابل استفاده موثر است. اما باید توجه داشت که هر نوع آزمونی کامل نیست و نمی تواند به تنهایی برای کشف همه نقصهای نرم افزار استفاده شود. حتی ابزارهای مختلف تحلیل استاتیک هم به همین گونه است و هر کدام نقاط قوت و ضعف دارند. لذا آزمون استاتیک به عنوان یک ابزار مکمل در کنار آزمونهای دیگری مانند آزمونهای دینامیکی و کارکردی استفاده می شود [5].

پروژه هایی که نرم افزار آنها مورد نظرسنجی از توسعه دهندگان آنها قرار گرفت، با استفاده از نرم افزار پاراسافت (Parasoft) به طور مستقل تحت آزمون تحلیل استاتیک هم قرار گرفتند [6]. نرم‌افزار آزمون تحلیل استاتیک گزارشهایی به صورت طبقه بندی شده با شدت نقصهای مختلف تولید کرد. برای هر نقص (به ویژه نقصهای با شدت بالا) فرم گزارش نقص طراحی و از نتایج آزمون در آن فرم درج گردید. یک نمونه از این گزارشات کشف نقص در شکل ۲ آورده شده است.

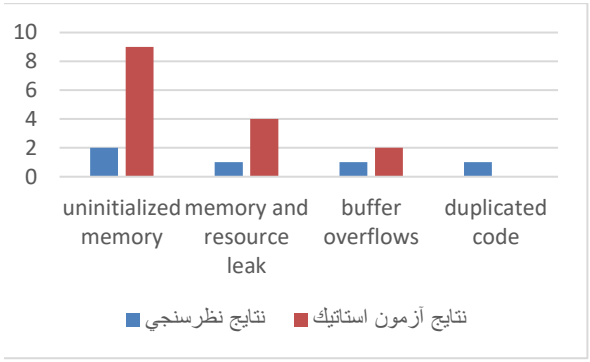
در جدول ۷ نتایج جمع بندی شده از نقصهایی که توسط آزمون خودکار استاتیکی در دو پروژه به دست آمده، نشان داده شده است. در شکل ۳ نتایج نظرسنجی و آزمون استاتیک برای دو پروژه مذکور در یک نمودار آورده شده است.

فرم گزارش نقص (باگ) نرم افزاری در آزمون ساختاری نرم افزار (آزمون جعبه سفید)	
نوع نقص (باگ) نرم افزاری: uninitialized memory	**شناسه نقص (باگ):** OBS-FLT-ST-0003
گزارش دهنده: شاهرخ جلیلیان	**دریافت کننده(گان) گزارش:** ***
نام پروژه: ***	**تاریخ گزارش:** ۱۴۰۰/۹/۱
نام مدل: مدل کیفی - QM	**نام زیرسیستم/ماجول:** ترم افزار روی برد
تاریخ یا نسخه سورس کد مربوطه: OBSW_QM__000719_03-New	**نام فایل مربوطه:** EPS_Handler.cpp
نام تابع دارای نقص: EPS_Handler::EPS_ALG_CHRmanagment	**شماره سطر نقص:** 883
ابزار مورد استفاده برای کشف نقص: Parasoft	**نام آنالیز:** Flow Analysis Standard
شدت نقص: خیلی بالا	**پیوست:** تدارد
شرح نقص: solar.HK_CHR_R_IN_SP_VOL در تابع get_data_solar_monitoring مقدار دهی نمی شود اما در کد استفاده می شود	
وضعیت نقص (در صورت تغییر وضعیت، انتخاب شود): گزارش نقص	
رفع کننده نقص:	
تایید کننده (گان) رفع نقص:	

شکل ۲- گزارش یک نقص نمونه از نتیجه آزمون تحلیل استاتیک

جدول ۷- آمار نتایج تحلیل استاتیک از دو پروژه

	Frequency	Percent
uninitialized memory	9	60
memory and resource leak	4	26.7
buffer overflow	2	13.3
Total	15	100.0



شکل ۳- مقایسه نتایج نظرسنجی و آزمون تحلیل استاتیک

٤- بحث، بررسی و درس آموخته

با توجه به شکل ۳ مشخص می شود که نتایج نظرسنجی با نتایج آزمون تحلیل استاتیک که به صورت مستقل از یکدیگر انجام شده اند، مطابقت دارد. اهمیت این موضوع از آن جهت است که توسعه دهندگان بر اساس نظرسنجی انجام شده، انتظار داشتند که (برای مثال) بیشترین نقص نرم افزاری در مقداردهی نشدن حافظه (متغیرها) وجود داشته باشد اما بدون انجام آزمون و کشف نقصها، نمی توانست معتبر باشد. در واقع نتایج آزمون خودکار تحلیل استاتیکی مهر تاییدی بر نظرات توسعه دهندگان بود. در عین حال انجام این پژوهش، بعضی نقایص مهم (هر چند ساده) در فرآیندها و محصول نهایی را برجسته تر و شفاف تر کرد که الزام می کند، متولیان ذیربط برای محدود کردن آنها، روشها و فرآیندهای تولید محصول را بازبینی نمایند. در اینجا درس آموخته هایی که از انجام این پژوهش بدست آمده به اختصار آورده میشود.

- در حالی که حافظه مقداردهی نشده اولیه از باگهای بسیار ساده‌ای است که به راحتی قابل پیشگیری هست، اما در این پژوهش به عنوان بیشترین باگ در دو پروژه شناخته شد. لذا باگهای ساده نیز نباید به هیچ عنوان نادیده گرفته شود بلکه باید با اتخاذ رویه مناسب برنامه نویسی و فرآیند صحیح صحه گذاری از تولید آنها تا حد امکان جلوگیری کرد. نقصهای دیگر کشف شده هم هر چند آمار کمتری داشتند، اما نباید به هیچ وجه آنها را کم اهمیت جلوه داد. برخی از نقصها در شرایط عادی منجر به بروز خطا و خرابی نمی شوند بلکه در حالتهای خاص باعث اختلال در کارکرد یا کارآیی نرم افزار می گردند.
- نقصهای نرم افزار همیشه در فرآیند توسعه نرم افزار تولید می‌شوند. نقصهایی وجود دارد که حتی ممکن است در آزمونهای متداول هم کشف نشوند. در پژوهش حاضر، با اینکه در هر دو پروژه، آزمونهای کارکردی انجام شده بود اما با اجرای آزمون خودکار تحلیل استاتیک توسط نرم افزار، نقصهای کشف نشده ای شناسایی شد. لذا برای پروژه هایی که بحرانی هستند (چه به لحاظ ماموریتی و چه ایمنی یا امنیتی)، توصیه اکید می شود که از آزمونهای خودکار تحلیل استاتیک که بخشی از آزمون جعبه سفید نرم افزاری هستند، استفاده گردد.
- استفاده از آزمونهای خودکار تحلیل استاتیک به جهت آنکه نیاز به اجرای نرم افزار در محیط نهایی را ندارد و توسط نرم افزار آزمونگر در روی کامپیوتر شخصی مورد آزمون قرار می گیرد، در وقت آزمون صرفه جویی قابل توجهی دارند. باید توجه داشت که هر نوع آزمون نمی تواند به تنهایی تمام نقصها را تشخیص دهد بلکه آزمونها مکمل یکدیگر هستند. لذا در حالی که تمایل مدیران پروژه ها بیشتر در جهت استفاده از آزمونهای کارکردی یا جعبه سیاه هست، فعال کردن آزمونهای جعبه سفید از جمله تحلیل استاتیکی به صورت خودکار می تواند ضمن ارتقا کیفیت نرم افزار، در هزینه و زمان پروژه صرفه جویی نماید.
- آزمون تحلیل استاتیکی در کنار آزمون دینامیکی بر خلاف آزمون کارکردی، می تواند از فازهای ابتدایی توسعه نرم افزار مورد استفاده قرار بگیرد. این کار به افزایش کیفیت نرم افزار و کاهش هزینه های توسعه در کل چرخه حیات توسعه نرم افزار کمک شایانی می کند.

٥- نتیجه‌گیری

پژوهش حاضر به منظور بررسی و شناسایی نقص در نرم افزار روی برد دو پروژه فضایی و فرآیند توسعه آنها انجام گرفت. بررسی در دو فرآیند مجزا و مستقل صورت گرفت. در مرحله اول از توسعه دهندگان نرم افزار روی برد (مشتمل بر طراح، پیاده ساز و آزمونگر و مدیر نرم افزار/سیستم) نظرسنجی در خصوص نقصهای مورد مواجهه و سایر موارد ذیربط انجام گرفت. در مرحله دوم، آن دو نرم افزار روی برد تحت آزمون خودکار تحلیل استاتیک قرار گرفت. نتایج آزمون خودکار با نتایج نظرسنجی مطابقت داشت. این آزمون در شرایطی انجام شد که محصول مورد آزمون، آزمونهای کارکردی را گذرانده بود و نقصهای کشف شده در آن آزمونها برطرف گردیده بود. با این حال نقصهایی کشف شد که در نظر اول دارای پیچیدگی هم نبود اما در صورت عدم برطرف شدن، می تواند منجر به بروز خطا و خرابی در شرایط خاصی در زمان عملیاتی شدن محصول نرم‌افزاری گردد. البته باید توجه داشت که این نظرسنجی و آزمون استاتیک برای دو پروژه انجام شده است و برای دقیق تر شدن نتیجه پژوهش، بهتر است در آینده بر روی پروژه های بیشتری انجام گیرد.

٦- مراجع

[1]Andrea Di Mauro, The importance of software in space missions, https://bmsis.org/the-importance-of-software-in-space-missions/

[2]https://www.parasoft.com/solutions/static-code-analysis/

[3]https://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_sm/1/897/ENUS5724-U01/index.html

[4] https://docs.sonarqube.org/latest/

[5] Peng L., Cui B. A Comparative Study on Software Vulnerability Static Analysis Techniques and Tools. Proceedings of the 2010 IEEE International Conference on Information Theory and Information Security; December 2010; Beijing, China. IEEE

[6]https://www.parasoft.com/solutions/static-code-analysis/