فصل ۲۰

وارسی نرم افزارهای شناخت گرا

مقدمه

سال های بسیاری است که سیستم های اتوماسیون شده کامپیوتری تقریبا در تمام زمینه های نرم افزار اهمیت زیادی به دست آوردند. در میان عامل های دیگر، دلیل این اهمیت، پیشرفت سریع در میکروالکترونیک و امکان پیاده سازی ارزان قیمت و قابل انعطاف آنها می باشد. ملاحظات اقتصادی شرایط مرزی دقیقی را در توسعه و بهره برداری از سیستم های فنی تحمیل میکند که برای سیستم های ایمنی مرتبط نیز برقرار هست. از آنجایی که نیروی انسانی به شدت در حال گران شدن است سیستم های ایمنی مرتبط نیاز به انعطاف پذیری بالایی دارند تا اینکه قادر باشند آنها را با تغییر نیازمندی ها با کمترین هزینه تنظیم کنند. به عبارت دیگر ، سیستم های ایمنی مرتبط باید به صورت کنترل شده، برنامه ریزی شوند. بنابراین، انتظار می رود که استفاده از سیستم های ایمنی سخت افزاری در مقایسه با نوع نرم افزاری کاهش خواهد یافت.

در جامعه، نگرانی فزاینده برای خطرات ایمنی و زیست محیطی یک افزایش تقاضا برای سیستمهای فنی قابل اعتمادی را که برای کمک به جلوگیری از فجایع زیست محیطی و از دست رفتن زندگی بشر است ، ایجاد میکند. برای فعال کردن سیستمهای تابعی سازگار و انعطاف پذیر برای نیازهای جدید و به منظور ارتقاء بهره وری فرایندهای توسعه سیستم ، سیستم های مبتنی بر کامپیوتر به طور فزاینده ای برای هر دو نوع توابع کنترلی و اتوماسیون تحت محدودیت زمان واقعی (Real Time) بکار برده میشوند . این سیستم ها ویژگی های خاصی دارند که سخت افزار و نرم افزار را بطور دقیق به سیستم های که از ترکیب تکنولوژی های مختلف است ، متصل می کند بعنوان مثل سیستم های تولید و پردازش یا کنترل ترافیک.

در هنگام ارزیابی قابلیت اطمینان آنها ، سخت افزار و نرم افزار به طور کامل خصوصیات متفاوتی دارند. سخت افزار موضوعی برای فرسوده شدن است. خطاها تصادفی رخ می دهند و ممکن است موقتی باشند. تا اندازه زیادی، این منابع غیر قابل اعتماد می توانند با استفاده از طیف وسیعی از روشهای افزونگی و تحمل خطا بطورموفقیت آمیز از عهده کار محوله بر آیند. شکست نرم افزار، از سوی دیگر، نه با فرسوده شدن ایجاد می شود و نه با حوادث محیطی مانند تشعشات و یا پالسهای الکتریکی. در عوض، تمام خطاها نیازمند آنالیز ، طراحی ، برنامه ریزی خطا هستند.یعنی آنها ماهیتی سیستماتیک دارند و علل آنها همیشه موجود است . قابلیت اطمینان نرم افزارها نمیتواند با کم کردن و به صفر رساندن تعداد خطاهای موجود در آزمایش، بازبینی یا با روشهای اکتشافی دیگر بدست آید، اما بسختی می توان ثابت کرد که نرم افزار عاری از خطاست . با توجه به وضعیت

کنونی هنر ، اگرچه، طرفداری گسترده روشهای رسمی تنها زمانی می تواند با این نیازمندی برخورد کند که برای تصدیق بخشهای کد ساخت یافته ساده و نسبتا کوتاه به کار برده شود.

در حال حاضر روش های متعددی از متد ها و دستورالعمل ها وجود دارد مثل IEC 880(1986) ، که مفید بودن آنرا برای توسعه نرم افزارهای با یکپارچگی بالا بمنظور کنترل ایمنی فرآیندهای فنی بحرانی، ثابت کرده است . پیش از بکارگیری آن ، چنین نرم افزار هایی ، بیشتر به اندازه گیری دقیق برای تایید و اعتبار سنجی اشاره دارد. با این حال، با توجه به وضعیت فعلی هنر، این اقدامات نمی تواند درستی برنامه های بزرگ ریاضی پیچیده را ضمانت کند. مشکلات پیش آمده بواسطه نیاز برای بررسی رفتار زمان واقعی تشدید میشوند. بنابراین تاییدیه نرم افزارهای سخت به طور کلی به خاطر نقش پیچیدگی این نرم افزارها هنوز یک مشکل حل نشده است. علاوه بر این ، به منظور ایمنی، مجوز کد شی (به عنوان مثال، فقط نسخه از برنامه در واقع قابل مشاهده و قربانی شدن با یک دستگاه است) باید بررسی شود، زیرا تغییر شکل نمایش برنامه از منبع به کد شی توسط یک کامپایلر یا اسمبلر ممکن است خطاهای را برای کد شیء بوجود آورد. در نتیجه، بسته به قانون ملی و عملکرد آن ، مسئولان مجوز هنوز هم بسیار بی میل هستند و یا حتی سرباز میزنند از اینکه سیستم های ایمنی که رفتار منحصرا کنترلی دارند را تایید کنند. به طور کلی ، مجوز ایمنی برای سیستم های بحرانی با ایمنی بالا با تکیه بر نرم افزار با پیچیدگی غیر بدیهی رد شده است .

حقیقتا، راه حل های مبتنی بر نرم افزار کم اعتماد تر از پیاده سازیهای مرسوم سخت افزار تلقی میشوند، که به عنوان مثال ، بواسطه عقیده های رایج قدیمی مهندسی سخت افزار و در نتیجه چندین دهه تجربه در توسعه استراتژی های مقابله با خطاهای مربوطه توجیه می شود. ارزیابی نرم افزار با مراجعه به جنبه های قابلیت اطمینان و ایمنی موضوع مهمی است که برای مدت طولانی نادیده گرفته شده بود. تنها پیشرفتهای قابل توجهی که هدف آن حمایت از روند تایید برنامه ها است نوید گام در مسیر درست است، زیرا قابل تایید بودن مهمترین پیشنیاز فراهم کردن تایید سیستم های مبتنی بر نرم افزار بزرگتر است. اهمیت این عبارت زمانی آشکار می شود که این واقعیت در نظر گرفته شود که،به دلایل عملی، حجم کاری مورد نیاز برای تایید ایمنی نرم افزار وابسته، بایستی از لحاظ اقتصادی به یک مقدار امکان پذیر محدود شود.

تا به حال از رویکردهایی برای کسب پیشرفتهایی در زمینه دستیابی به امنیت نرم افزار نتایجی بدست نیامده که کمکی به منظور افزایش اطمینان در راه حل های مبتنی بر نرم افزار، کرده باشد. از اینرو مسئولان صدور تاییدیه ، منحصرا نسبت به مجوز ایمنی مبتنی بر نرم افزار سیستمهای مرتبط، دودل هستند. در نتیجه، امروزه با وجود

معایب آشکار رویکردهای مرسوم سخت افزار با توجه به جنبه های اقتصادی ذکر شده در بالا، اغلب طراحان سیستم هنوز بکارگیری آنها را ترجیح می دهند.

برای بدست آوردن یک چاره برای این وضعیت ناخوشایند ، مثلا به منظور ارتقا قابلیت اعتماد و در نتیجه، قابلیت اعتماد ایمنی مبتنی بر نرم- افزار سیستم های مرتبط ، این مقاله برخی از مفاهیم تعیین شده برای تایید نرم افزاری مناسب را معرفی میکند. در پایان نیز، برخی مفاهیم اساسی و ویژگی های ذاتی نرم افزار ، با توجه ویژه به ماهیت متفاوت خطاهای نرم افزاری و سخت افزاری بحث می شود. سپس، سادگی را به عنوان اصل اساسی که برای مفاهیم تصدیق قطعی فرض میشود تا در امنیت مهندسی نرم افزار مرتبط، استفاده شود. یک ارزیابی از روشهای تایید نرم افزار موجود نشان می دهد که همه آنها یا به علت عدم دقت و یا اینکه بعلت دشواری بیش از حد درک و بکارگیری آنها، نامناسب هستند.

از انجایی که در اکثر موارد امکان مواجه با یک مشکل پیچیده وجود دارد ، در این مقاله ما سعی در ارائه راه حلی کارامد در زمینه امنیت با بهره گیری از موارد خاص که از دیدگاه مهندسی امنیت ارزشمند و مهم است ، داریم . در حال حاضر این مشکلات به دلیل محدودیت متغییرهای موجود در دامنه کاربرد نرم افزار قابل مدیریت است .علاوه بر این ،در حال حاضر نرم افزارها به بهترین شکل ممکن طراحی شده اند، ساده با طراحی پیچیده اتصالات گرافیکی و بلاکهای دقیق اطلاعات .چنین نرم افزارهایی به سادگی قابل تمایز و ویرایشند .

یک معماری کامپیوتر با قابلیت ترجمه بازگشتی ارائه شده است . ترجمه بازگشتی یک تکنیک متمایز نرم افزاری است که در متنهای زمان بر و کسل کننده به کار میرود . اگرچه زمانی که چنین برنامه ای به صورت گرافیکی و در بلاکهای مدون نرم افزاری نوشته شود ، برنامه ترجمه بازگشتی برنامه ای آسان ، باصرفه و کارامد از نظر زمانی خواهد بود . رویکرد حاضر در زمینه ارائه پشتیبانی برای تصدیق نرم افزار در معماری، بی همتاست .نظریه پیشگام در این طراحی بر پایه ترکیب مهندسی نرم افزار و روشهای مختلف معماری پشتیبان، استوار است . با این روش گپ ها یا فواصل معنایی موجود میان نیازمندی های نرم افزار و قابلیت های محیط اجرا می تواند از بین برود ، که موجب حذف نیاز به کمپایلرها و سیستمهای عامل و در نتیجه پائین اوردن سطح امنیت میشود .

مفاهیم اساسی امنیت

در استاندارد صنعتی آلمان (۱۹۸۷) DIN 31000 امنیت به معنای شرایطی است که ریسک در آن از یک محدوده ریسک فراتر نرود و یا شرایطی با حداکثر میزان مقبولیت و سطح قابل توجهی از ریسک در یک فرآیند یا شرایط فنی. در یک پروسه کاری معمولا میزان پتانسیل ریسک یا خطربالقوه موجود ، قابل اندازه گیری و

مقایسه با محدوده ریسک انتخاب شده برای تعیین اینکه ایا این پروسه ایمن است یا خیر، می باشد. محدوده ریسک خود به صورت یک کمیت مطلق نیست. در محیط های صنعتی ، محدوده ریسک عمدتا وابسته به استانداردهای موجود مهندسی است . هر چند مثالهای زیر نشان میدهند در بسیاری از شرایط ، محدوده ریسک، بر طبق معیارهای فرهنگی و اجتماعی تعیین میشوند. در ابتدا ، ما دو کشور آلمان و آمریکا را در زمینه استفاده از اتومبیل و اسلحه در نظر میگیریم . در جاده های آلمان در مقایسه با آمریکا هیچ حد و محدوده سرعتی وجود ندارد. از طرف دیگر حمل اسلحه جزو حقوق شهروندان آمریکاست، در حالی که صدور مجوز مربوطه در آلمان بسیار محدود است. بنابراین جای تعجب نیست که بیشتر مردم آلمان در جاده های آن و اکثر مردم آمریکا به ضرب گلوله کشته می- شوند. دوم، بیائید اتفاقاتی را در نظر بگیریم که معمولا رخ نمی دهند و در صورت رخ دادن، پیامدهای سنگینی به همراه دارند . به عنوان مثال اگر همه پروازهای B-747 در آلمان سقوط کنند ، این حادثه منجر به قطع خطوط هوائی و کشته شدن تنها یک صد هزار نفر در سال خواهد شد ، که برابر با میزان کشته شدگان بر اثر مصرف سیگار است.

کنترل زمان واقعی

سیستمهای زمان واقعی در کنترل فرآیند و اتوماسیون به کار میروند، که به طور فزاینده با برنامه های امنیتی سرو کار دارند. حالت اجرائی زمان واقعی در استاندارد صنعتی آلمان (۱۹۸۵) DIN 44300 به عنوان حالت اجرائی یک سیستم کامپیوتری تعریف می شود ، که در آن برنامه ها دائما برای پردازش اطلاعات ورودی آماده هستند، بطوری که نتایج این برنامه ها در دوره های زمانی از پیش تعیین شده در دسترس خواهد بود ؛ زمانهای دسترسی به اطلاعات میتواند به صورت تصادفی و یا بر مبنای برنامه های مختلف تعیین و مشخص شود . از این رو سیستمهای کامپیوتری زمان واقعی، همواره در محیط های بزرگی که نیاز به همگام سازی پویا دارند جاسازی شده و به کار گرفته میشوند . بنابر این این سیستم ها اغلب مترادف با سیستم های توکار به شمار می آیند.

سیستم های زمان واقعی بوسیله دخالت صریح بعد زمان،که یکی از نیازهای اساسی به هنگام بودن است، مشخص می شوند. سیستم های زمان واقعی بایستی حتی تحت شرایط بارگذاری حداکثری، برآورده شوند. بمحض درخواست فرآیندها باید استخراج داده ها ، ارزیابی و واکنش های مناسب بموقع اجرا شوند. در این سیستم ها، سرعت پردازش قطعی و مشخص نیست ، گرچه حدود زمان پاسخ گویی، قابل پیش بینی است اما این زمان هرگز قطعی نخواهد بود. از این رو مشخصه سیستمهای بلادرنگ این است که صحت عملکردی آنها علاوه بر نتایج پردازش ، به زمان پردازش نیز بستگی دارد. به خصوص در زمان بروز خطا، کاربر انتظار کاهش

کارایی سیستم را دارد. در نهایت، رفتار سیستم کاملا قطعی ، یک سیستم صدور مجوز ایمنی موثر برای ایمنی برنامه های حیاتی را فعال خواهد کرد. بنابراین پییش بینی رفتار سیستم های بلادرنگ، مهم ترین بخش در این سیستمهاست. این مکمل نیاز به همگام سازی دارد تا بتوان رفتار سیستم را بر اساس زمان و عوامل خارجی و در نتیجه بازخورد عملکرد سیستم را پیش بینی کرد .

مشکلات ذاتی یک نرم افزار

زمانی که به بررسی مشکلات و خرابی های یک سیستم کامپیوتری می پردازیم ، متوجه می شویم که در اغلب موارد مشکل سخت افزاری نیست . نوعی از خطاها و مشکلاتی که در زمانهای تصادفی رخ می دهد و از عوامل فیزیکی و مکانیزم هایی که تاثیر منفی بر سخت افزار دارند، ناشی می شود. برخی دیگر از خطاها آنهایی هستند که از حالات قطعی، که فرض سیستم بر این است چنین مشکلاتی در طول عملیات و در فازهای تعریف مشخصات و پیاده سازی رخ نخواهد داد، ناشی می شود.

براساس ماهیت غیر مادی آن ، نرم افزار دارای کیفیت کاملا متفاوتی نسبت به سخت افزار است. در نتیجه نرم افزار ها بر خلاف سخت افزار ها در طول دوران حیاتشان، تحت تاثیر عوامل فیزیکی قرار نگرفته و از کاراییشان کاسته نمی شود. از طرف دیگر نرم افزار ها ذاتا مستعد اشتباه و دارای خطاهای نوشتاری هستند. اینها خطاهای طراحی هستند که در فازهای مختلف توسعه نرم افزار به دلیل اشتباهات و یا حذفیات رخ می دهند. همچنین نرم افزار (از نظر آنالیز ریاضی) پیوسته نیست، بدین معنی که تاثیر خطاهای کوچک لزوما کوچک نیست و ممکن است حتی نامحدود باشد.

نرم افزار بایستی صحیح و معتبر باشد. که این مستلزم دو مرحله تایید است که باعث ایجاد مشکلات اساسی می شود :

۱. با نشان دادن اینکه نرم افزار معیارهای مسئله را بر آورده می کند،(یعنی اینکه، آن شامل خطاهای سیستماتیکی که در مراحل مختلف طراحی،کد نویسی و یا در اثر استفاده از ابزارهای مختلف بوجود می آید، نمی شود.) صحت نرم افزار فراهم می شود. این قبیل اثبات ها برای دقیق بودن، نیاز به روش تستی دارد که بتواند وجود خطا را کشف کند.
۲. نشان دادن اینکه واقعا برخی از مشخصات با در خواست های شخصی که آنها را مشخص کرده (که می تواند به صورت مبهم فرموله شود) یا حتی افکار او مطابقت دارد، وظیفه اثبات اعتبار است.

در واقع ممکن است یک نرم افزار مشخصات مورد نظر را داشته باشد اما آن گونه که باید رفتار نکند. در چنین شرایطی خیلی اوقات یک مشخصه بخودی خود نادرست ، ناقص یا ناسازگارند . از اینرو ، مشخصه و در نتیجه نرم افزار حاصل معتبر نیست. متاسفانه اگرچه این امکان وجود دارد که صحت یک برنامه در برابر مشخصه ها و معیارهای تعریف شده را بررسی کنیم ، اما راه حلی که صحت خود معیارهای تعیین شده را تایید کند، وجود ندارد. در نتیجه، برای تولید یک نرم افزار مناسب ، باید در ابتدا زمان و تلاش زیادی را صرف تعریف مشخصه ها و معیارهای مناسب برای آن نرم افزار کرد.

اگر چه هنوز راه حل مناسبی برای اجرای دو مرحله فوق وجود ندارد، اما این مهم از طریق روش های علمی و تکنیکی قابل حصول است . بهرحال، مسئله دوم ممکن است به دلیل عوامل انسانی دخیل در آن، هرگز به طور قطعی قابل حل نباشد.

سادگی به عنوان اصل هدایتگر

بمنظور فراهم کردن ملاک صحت نرم افزار با بیشترین قابلیت اعتماد و کمترین تلاش، بکارگیری مفاهیم برنامه نویسی و ویژگی های معماری که فرآیند تایید را پشتیبانی می کند، ضروری بنظر می رسد. همانگونه که قبلا تاکید شد ، تایید و در نتیجه ادعای اینکه یک برنامه ظاهرا عاری از خطاهای نرم افزاری است، حقیقتا پیش نیاز اصلی برای اعطای یک لایسنس امنیتی توسط موسسه های تصدیق مجاز می باشد.

همانطور که در بالا بحث شد ، دستیابی به ایمنی مطلوب در سیستم های کامپیوتری تنها از طریق تعیین نیازها و اهداف در نهایت سادگی و به عنوان اصول اساسی طراحی مناسب، قابل دستیابی خواهد بود . نظر Dijkstra ، لزوم مقابله با این پیچیدگی ها را بیان می کند؛

"وقت آن رسیده است که جامعه ی کاربران کامپیوتر را به عنوان یک جامعه پنهان، برای ایجاد و حفاظت از پیچیدگی مصنوعی نمایان کنیم" (دیکسترا ۱۹۸۹)

آن به هیچ وجه برای به دست آوردن طرح های ساده، راحت نیست،درمقابل، Biedeukopf در سال ۱۹۹۴ بیان کرده است که:

"راه حل های ساده ،مشکل ترین راه حل ها هستند: این راه حل ها نیاز به نو آوری زیاد و نفوذ فکری به مسایل دارد" (biedeukopf 1994) اگر تعریف کنونی از هنر را در تایید چگونگی برنامه ها درنظربگیریم، خواهیم دید سادگی، مشخصه اصلی و پیش شرط برای افزایش اطمینان درسیستم های مبتنی بر کامپیوتر و همچنین برای

فعال ساختن مجوز احراز هویت برای تایید رسمی استفاده از کامپیوترها با هدف ایمنی کنترل بحران است. موارد ذکر شده، در بررسی وضعیت شان در رابطه های مبتنی بر علت زیر، مشخص می شوند:

- ✓ **سادگی** ⟵ از لحاظ فکری آسان واز لحاظ اقتصادی امکان پذیر
- ✓ **قابل پیش بینی بودن** ⟵ قابل اعتماد بودن
- ✓ **سادگی** ⟵ به سادگی قابل فهم بودن ⟵ قابل تایید بودن

در ابتدا، دررابطه با کامپیو ترها این مورد باعث تعجب است که با مفهوم قابل پیش بینی بودن مواجه شویم،در اصل ،تمام کامپیوتر های دیجیتالی به صورت کاملا قطعی کار میکنند و بنابراین در رفتار خود قابل پیش بینی هستند. برای اینکه بخواهیم به طور دقیق معنی مشخصی از قابل پیش بینی بودن ارایه کنیم ،که به عنوان یک مفهوم اساسی از کنترل کامپیوتر مناسب باشد، صفت «آسانی» را به عنوان اولین مفهوم مورد استفاده قرار می دهیم. این مورد، مفهوم قابل پیش بینی بودن را به وسیله ی پرداختن به تلاش عقلانی و مقرون بصرفه، که نیازمند سرمایه گذاری برای ایجاد این خصوصیت برای سیستم مورد نظر است، توصیف میکند. اگر این موضوع ساده باشد، می تواند به راحتی درک شود ،که گام اصلی در جهت رفتار درست آن در تشخیص دکارت است " verum est quod valde dare distinct " (۱۶۴۱)

تایید نرم افزار به عنوان فرایند شناخت

تصور دکارت از فهم واضح و مجزا، کلید درک ماهیت تایید است،که نه علمی است و نه فنی،اما فرآیندی شناختی است. این برای اثبات های ریاضی به کار میرود ،که صحت آن بر پایه اجماع نظر در میان اعضای جامعه ی ریاضی دانان بنا شده است که در نتیجه ی افزودن یک مولفه اجتماعی به ماهیت شناختی فرآیند باعث می شود زنجیره ی خاصی از استدلال، به نتایج حاصله منتهی شود. که نه تنها برای امنیت سیستم های کامپیوتری مرتبط و بررسی اهمیت آنها در زندگی و سلامت بشر کاربرد دارد، بلکه برای کسانی که از لحاظ محیط زیستی واقتصادی اضافه شده اند، این مورد اهمیت پیدامیکند که این اجماع تا جایی که ممکن است، گسترده باشد. بنابراین سیستم ها باید ساده باشند ومتدهای تایید نرم افزار باید به راحتی برای افراد غیر متخصص قابل فهم باشد، بی آنکه دقت زیاد تلاش آنها را به خطر بیندازد.

ارزیابی متد های تایید نرم افزار:

تایید نرم افزار یک فرآیند بررسی در هر فاز از چرخه ی توسعه و نگهداری نرم افزار، برای تشخیص اینکه آیا تمام نیازهای یک فاز به درستی درآینده پیاده سازی میشوند، را در بر می گیرد. یک فرآیند تایید کامل، نشان میدهد، آیا رفتارهای یک برنامه با جزییات دقیق آن مطابقت دارد. تایید، فرایند ارزیابی یک مولفه نرم افزاری است که مشخص می کند آیا مولفه نرم افزاری شرایط تحمیل شده در ابتدای مرحله پیاده سازی را برآورده می کند یا نه. این بدین معناست که صحت نرم افزار بر اساس توصیف های مراحل قبل چک می شود. بعبارت دیگر، تایید تلاش می کند تا به این سئوال پاسخ دهد که آیا مولفه های نرم افزاری بدرستی ساخته شده اند یا نه. تایید بایستی بطور مشخص از مفهوم اعتبار، که یک فرآیند ارزیابی است برای تشخیص اینکه آیا آیتم نرم افزاری نیازمندی های نرم افزار را برآورده می کند یا نه، متمایز شود. این بدین معناست که صحت نرم افزار از منظر یک مجموعه از خصوصیات نیازمندی، که بعنوان بخشی از قراردادهای حقوقی بین مشتری و تیم توسعه است، بررسی می شود. اعتبار تلاش می کند به این پرسش پاسخ دهد که آیا آیتم های نرم افزاری درست، ایجاد شده اند.

کمال نرم افزار ، یعنی ، مشخصات بدون ابهام و برنامه های صحیح، یک مسئله پیچیده را نشان می دهد. از لحاظ تئوری، خطا های نرم افزار ممکن است از طریق تست های کامل، حذف شوند. به علت پیچیدگی های نرم افزار ،تست های کامل اغلب در تعداد معدودی از برنامه های اجرایی ، قابل انجام نیست. بنابراین مقیاس های تضمین کیفیت از همان اهمیت ویژه ای که در فاز توسعه ی نرم افزار برخوردارند، در فرآیند- های تایید و اعتبار سنجی نیز برخوردارند. این به معنی ، خصوصیات دقیق نیازمندی ها ،ماژولاریتی خوب برنامه ها، استفاده از متدهای برنامه نویسی ساخت یافته، استفاده از ماژول های نرم افزاری پیش ساخته و از قبل تایید شده، مستندات جامع و مهم تر از همه پایین نگهداشتن پیچیدگی نرم افزار بطوریکه این موضوع علت اصلی خطاهای طراحی سیستم است، می باشد. تمام مقیاس ها برای جلوگیری از ایمنی خطا های بحرانی، دامنه و اثربخشی محدودی دارند. بنابراین هیچ مقیاسی آنقدر قدرتمند نیست که بتوان به تنهایی برروی آن تکیه کرد .اثر مقیاس ها به وسیله ی استفاده ی ترکیبی آنها قابل افزایش است.

در مقالات، تعداد زیادی از روش های تایید واعتبارسنجی نرم افزار گزارش شده است. مقاله ی تحقیقی توسط hausen (1987) ، یک بررسی اجمالی از روشهای تست نرم افزاری شناخته شده، ارائه می دهد. منبع اطلاعاتی دیگر،کار اساسی انجام شده توسط EWICS TC7 (1985) است. بهر حال، در اصل، هیچ تکنیکی که بتواند تمام

خطاهای نرم افزار را خودش تشخیص دهد، وجود ندارد. بنابراین تکنیک ها بایستی بصورت ترکیبی مورد استفاده قرار گیرند. پس باید بین روش های تحلیل استاتیک ودینامیک نرم افزار، تمایز قائل شد.

تست تمام گونه های آن ،تا حد بسیار زیادی از روش آنالیز دینامیک برای تایید واعتبار سنجی نرم افزار استفاده میشود ،از آن زمان تنها در موارد بسیار اندک ، روش های اثبات صحت ماژول های نرم افزاری، می توانند با دقت ریاضی انجام شوند. آزمون، شامل اجرای مولفه های نرم افزاری تحت شرایط خاص ،ثبت نتایج، و ارزیابی آنها در برابر انتظارات مشابه می باشد. برای تدوین سیستماتیک روش های عمده ی تست نرم افزار ما به trauboth (1993) مراجعه می کنیم. هر چند که کاملاً موثر بودن به معنی یافتن خطا ها در ماژول ها واجزای نرم افزار است، صحت عموماً نمیتواند بر پایه ی تست استوار باشد. با وجود اینکه موارد بسیار زیادی را تحت پوشش قرار داده است، اغلب به اندازه ی کافی فراگیر نمی باشد. بنایراین آزمون نمیتواند عدم وجود تمام خطا ها را تایید کند. دو رویکرد متفاوت از تست نرم افزار، آزمون جعبه سیاه و آزمون جعبه سفید، تشخیص می دهد که آیا آزمون داده تنها از مشخصات برنامه به دست آمده است یا علاوه بر آن از بررسی ساختار برنامه های داخلی نیز به دست آمده است. به عنوان یک قاعده ،آزمون داده باید تمام موارد را پوشش دهد و تمام اقداماتی که برنامه انجام میدهد را شامل شود. از اینرو برای انتخاب آنها، خصوصیات برنامه کشف، شرایط اجرای طبیعی تشریح ، موارد خاص و ورودی های نامعتبر متناسب- از قبیل مقادیر خارج ازدامنه، یا شرایطی که منجربه پایان پردازش میشود توصیف می شوند- توصیف می شوند. در اینجا سه نوع کلی طراحی سیستماتیک برای آزمون جعبه سفید وجود دارد : ۱. آزمون دستور،که در آن هر دستور از مولفه نرم افزاری تحت آزمون حداقل یک بار اجرا می شود ۲. آزمون انشعاب که در آن هر نتیجه از هر نقطه تقسیم در مولفه های نرم افزاری، حداقل یک بار اجرا می شود ۳. آزمون مسیر،که در آن هر مسیر در میان مولفه نرم افزاری حداقل یک بار اجرا می شود. لازم به ذکر است که آزمون دستور به کمترین تلاش نیاز دارد، در حالی که آزمون مسیر نیاز به بیشترین تلاش دارد.

- گروه روش های تجزیه و تحلیل نرم افزار استاتیک شامل:
- بررسی و بازبینی
- بازرسی
- walkthrough (جلسه بررسی که در ان یک مرحله از توسعه سیستم یا برنامه برای معرفی کردن خطاها مرور می شود)
- روشهای اثبات صحت رسمی

• اجرای نمادین و

• ترجمه معکوس متمایز

بررسی یک فرآیند ارائه ی محصول به سایر افراد ذینفع است، برای نظارت و یا تایید. بازبینی یک آزمون مستقل و رسمی از یک محصول، برای ارزیابی انطباق آن با مشخصات یا توافق نامه های قراردادیست .اهداف انجام بررسی ها و بازبینی ها در این دوره از پروژه ی توسعه ی نرم افزار شامل دو بخش است. هدف تکنیکی این است که شایستگی بخشی از کارها را و درجه ای که آنرا با شرایط پیش بینی شده و انتظارات مطابقت می دهد، ارزیابی کنیم. هدف مدیریتی این است که میزان پیشرفت را تعیین و وضعیت فعلی پروژه را ارزیابی کنیم. یک بررسی، اغلب به عنوان یک جلسه ی رسمی سازمان یافته میشود که در ان یک محصول ، در این مورد خصوصیات مورد نیاز، یک توصیف طراحی یا یک برنامه است، بررسی میشود . هدف از این نشست و جلسه شناسایی و ثبت هر گونه مشکل با محصول تحت بررسی است. اگر هیچ خطای بحرانی پیدا نشود، محصول میتواند به طور رسمی پذیرفته شود. بایستی تاکید شود، که به کشف مشکلات و تلاش برای حل انان در میان جلسه بررسی، پرداخته نمیشود. مشکلات بالقوه باید به وسیله ی داوران از مدتها قبل، زمانیکه برای جلسه آماده میشوند، مشخص شود. سپس، تحلیل گر یا طراح بعد از جلسه، خطا ها را بر طرف می کند. یک بازبینی عملیاتی یک آزمون نهایی است، که قبل از تحویل نرم افزار برگزار میشود، تا تصمیم بگیرند که آیا محصول نرم افزاری تمام نیازمندی ها را بر آورده می سازد یا خیر. یک بازبینی فیزیکی یک بررسی سازگاری رسمی بین کد نرم افزار ، طراح مربوطه ومسندهای کد می باشد.

بررسی و بازبینی توصیف شده در بالا، شامل جلسات رسمی انجام شده توسط مدیریت پروژه برای ارزیابی وضعیت پروژهاست. دو تکنیک دیگر که شباهت های ظاهری زیادی را از لحاظ بررسی باهم دارند به نام های walkthroughs (جلسه بررسی که در آن یک مرحله از توسعه سیستم یا برنامه برای معرفی کردن خطاها مرور می شود.) و inspections (طراحی و بررسی کدها) نامیده می شوند، که می توانند برای تایید اسناد و کدها، مورد استفاده قرار گیرند. حقیقت امر این است که دو روش inspections و walkthroughs جزو گسترده ترین روش ها برای بازبینی کدها هستند. این تکنیک ها برای مقیاس وسیعی از سیستم های نرم افزاری، روش های عملی تضمین کیفیت توسعه نرم افزاری را ارائه می دهند.

روش طراحی و بررسی کدها (inspections) در مکانی بین بررسی رسمی و (walkthroughs) غیر رسمی قرار می گیرد. اینها جلسات رسمی هستند که در دو نقطه از چرخه توسعه نرم افزار یعنی بعد از طراحی جزئیات و

بعد از پیاده سازی، قرار می گیرند. هدف از روش inspections تشخیص بسیاری از خطاهای ممکن در بررسی محصولات و ارزیابی میزان کارایی طراحان و برنامه نویسان می باشد. عضویت در جلسه بررسی معمولا به چهار نفر محدود می شود:

۱- **رئیس :** که وظیفه کنترل فعالیت ها را بر عهده دارد.

۲- **طراح نرم افزار:** که مولفه های اولیه نرم افزار را توسعه داده و شروع به طراحی و بررسی کدها می کند.

۳- **برنامه نویس:** کسی که وظیفه برنامه نویسی مولفه ها را برعهده دارد.

۴- **تست کننده :** کسی که مسئولیت تست مولفه ها را بر عهده دارد.

از جمله نتایج حاصل از روش inspection ، می توان به کشف خطاها و طبقه بندی و ثبت آنها اشاره کرد. نمایندگان مدیریت معمولا توجهی به جلسات inspection ندارند، اگرچه از نتایج حاصل از آن آگاهی دارند.

ساختار walkthroughs (جلسه بررسی که در آن یک مرحله از توسعه سیستم یا برنامه برای معرفی کردن خطاها مرور می شود.) نیز شباهت زیادی به بررسی گروهی از افراد که همدیگر را برای بررسی بخشی از کار ملاقات می کنند، دارد. اگرچه تکنیک walkthroughs به صورت غیر رسمی می باشد که هدف اصلی آن ارزیابی کار نمی باشد، بلکه ترجیحا برای کمک به طراح و یا برنامه نویس در پیداکردن بسیاری از خطاهای ممکن در پروژه آن افراد می باشد. نمایندگان مدیریت معمولا در چنین جلساتی شرکت نمی کنند ولی ممکن است افراد برای جلوگیری از هرگونه خطا درکار، به عنوان بحث آزاد، پروژه خود را برای آنها ارائه کنند. جلسات walkthroughs می توانند برای بررسی خصوصیات نرم افزار، طراحی برنامه، کد، برنامه تست و یا نتایج تست، یا مستندات نرم افزار، سازماندهی شوند. اگرچه جلسات walkthrough ، برخلاف جلسات رسمی، به کامل شدن محصول نهایی کمکی نمی کنند، اما با بخش کوچکی از آن سر و کار دارند. تعداد جلسات می تواند متناسب با هر مرحله از چرخه توسعه نرم افزار، سازماندهی شود. ماژول های مورد بررسی توسط افراد ناظر، تشابه زیادی به ماژول های مورد بررسی در جلسات رسمی بررسی نرم افزار دارند. کلمه "ساختار" استفاده شده برای نامگذاری این تکنیک معادل " به خوبی سازماندهی شده" می باشد و پیشنهاد می شود تا پیش بینی های لازم قبل از یک جلسه بمنظور اطمینان از موفقیت آن، صورت پذیرد. شرکت کنندگان بایستی نسخه- هایی از موارد لازم برای بررسی را فراهم کرده باشند. تجربه نشان داده است که میزان موارد لازم برای بررسی نباید از یکصد خط کد برنامه تجاوز کند و یا بیش از پنج تا ده صفحه از مشخصات یا توضیحات باشد. برای جلوگیری از بحث های طولانی مدت ، اعضای جلسات walkthrough باید به حداکثر شش نفر محدود شوند و طول جلسات منتهی به

نیم ساعت یا حداکثر چهل و پنج دقیقه باشد. ضروریست که شرکت کنندگان در جلسات، بر روی خود محصول متمرکز شوند و نه طراح یا برنامه نویس. این کار باعث ایجاد فضایی دوستانه و صمیمی بین افراد تیم می شود.

مزایای استفاده از ساختار walkthroughs متعدد هستند. شرکتهایی که از این روش در طی سالیان متمادی استفاده کرده اند گزارشهایی مبنی بر افزایش کیفیت نرم افزار، افزایش تخصص و بهره وری طراحان و برنامه نویسان و در محدود زمانی جلسات و بودجه، ارائه کرده اند. بهبود کیفیت نرم افزار از کاهش چشمگیر تعداد خطاها در مشخصات نیازمندی ها ، طراحی نرم افزار و کدهای نهایی حاصل می شود. نکته بسیار مهم این است که خطاها خود را نسبتا زود نشان می دهند. بنابراین روند اصلاح ساختار و بهینه سازی ساختار نرم افزار، زیاد طولانی نخواهد شد و باعث تسهیل سیستماتیک تست برنامه و افزایش قابلیت اطمینان برنامه خواهد شد. علاوه بر این ساختمان عمومی طرح تست ها و داده ها، می تواند روند تست را بسیار موثر تر نماید.

مشخصات موجود در ابتدای فرایند توسعه نرم افزار، برای مثال مشخصات نیازمندی ها و مشخصات نیازمندی های تفصیلی، می توانند با توجه به وضعیت کنونی پروژه، و تنها به روش های غیر ریاضی و غیر اتوماتیک مانند inspections و walkthroughs انجام شده توسط تیم های بازرسان انسانی و البته تحت نظر مدیران، مورد تایید قرار گیرند. تجربه حاکی از آن است که چیزی حدود ۸۰٪ از خطاها در هر دوره از روش inspection قابل شناسایی هستند، به خصوص تایید مشخصات نیاز به بزرگترین تلاش ؛ از اینرو غالبا خطاها به پیچیدگی و بکارگیری زبانهای طبیعی منجر می شوند و به اسانی می توانند روی کل پروژه انتشار یابند. مشخصات فنی باید در صورت امکان به صورت نیمه رسمی فرموله شوند، اینکار به گسترش درجه صحت آنها کمک می کند. جداول تصمیم گیری ابزارهای مناسبی برای این منظور هستند، زیرا تمامی خصوصیات نماد های رسمی را بدون به خطر انداختن درک عمومی، ارائه می دهند. این جداول اجازه توصیف جریان کنترل منطقی زیر فرآیندها با استفاده از وضعیت های درونی خود، را می دهند. بنابراین امکان بررسی کامل را تسهیل می کنند.

تایید رسمی می تواند نشان دهد که مشخصات پیاده سازی نرم افزار با استفاده از اثبات دقیق ریاضی، توصیف شده است. در اینجا برای متقاعد کردن خواننده در مورد حقیقت برخی از فرمول ها، اثبات دنباله ای از مراحل استدلال، آورده شده است. بمنظور دست یافتن به توانایی تایید رسمی مشخصات نیاز به رسمی کردن داریم. برای این منظور طیف وسیعی از روش های رسمی ابداع شده است. این روش ها حامی رسیدن به نرم افزاری قابل اعتماد بخصوص در زمینه ایمنی و امنیت حوزه برنامه های بحرانی، می باشند. حتی در برخی از محافل رضایت خاطر در مورد این روش های رسمی وجود دارد و به آنها به عنوان دارویی برای تمام مشکلات امنیتی سیستم، نگاه می کنند.

البته بطور کلی و به صورت خاص در زمینه امنیت مرتبط با سیستم های اتوماسیون، فاصله بسیار زیادی بین پژوهش های علمی و نیاز های کاربر و عمل بصورت صنعتی وجود دارد. با توجه به روش های رسمی، در سال ۱۹۹۲ توسط موسسه مطالعات پیشرفته ناتو این نتیجه بر روی محاسبات بلادرنگ بدست آمد که :

"... با اینکه توضیحات رسمی با ارزش هستند ، ولی وضعیت این هنر هنوز به آنجایی نرسیده که بتوان از آنها بصورت گسترده استفاده کرد. بطور خاص تعداد افرادی که می توانند به ایجاد توضیحات رسمی سیستم های غیر بدیهی اعتماد کنند، بسیار محدود است" (چه برسد به تعداد افرادی که به دستکاری این توضیحات برای افزایش صحت آنها دست بزنند) (Halang and Stoyenko, 1994)

رویکرد رسمی دارای ضعف ها و اشکالاتیست و این دلیلی برعدم استفاده عملی از آن، است:

• روش های رسمی فقط برای خطاهایی که به صورت طبیعی و به صورت مدل ریاضی(جبری یا حسابی) باشند، موثر هستند. برای برنامه هایی که دارای چنین مدل هایی نیستند، نیاز به تلاش های قابل ملاحظه ای برای ساختن تئوری ریاضی، قبل از شروع فرایند توسعه وجود دارد.

• مشخصات رسمی برای کاربران وجود عینی ندارد، بنابراین اعتبار بخشیدن به این مشخصات نیازمندیها، دشوار است.

• استفاده از تکنیک های تایید رسمی، نیاز ویژه به تخصص ریاضی دارند.

• زمانی که با دست انجام شوند ، اثبات صحت عملکرد برنامه های نسبتا طولانی، ناچارا به خطاهای انسانی منجر می شود که ممکن است برای مدت طولانی به ازای هر بررسی قابل تشخیص نباشند.

• در اینجا عدم وجود ابزار کافی برای پشتیبانی دیده می شود. ابزار هایی که وجود دارند، حتی برای استفاده متخصصان نیز مشکل می باشد، چراکه آنها بر مبنای تمرکز بیشتر بر روی اثبات تجربی متد و در نتیجه تمرکز کمتر بر روی پشتیبانی فرایند انجام کار و ساختن متد های شفاف و آسان برای استفاده، طراحی شده اند.

با توجه به نتایج بالا، روش های اثبات صحت رسمی، تنها برای ماژول های کوچک نرم افزاری قابل اجرا هستند ، اگرچه این مسئله در کل کافی نیست ولی خوشبختانه در حال حاضر به ما اجازه مقابله با ایمنی زیاد توابع مرتبط را می دهد. خوشبختانه در این حوزه اغلب امکان بکارگیری اجرای برنامه های نمادین وجود دارد، یک روش رسمی تنها به تبدیلات جبری استاندارد نیاز دارد.

توسعه این فنون برای صدور مجوز ایمنی برنامه های کامپیوتری، هنوز در مرحله بسیار ابتدایی قرار دارد. ملاحظات ما تاکنون نشان داده اند که روش های غیر رسمی (بر پایه مفهوم ریاضی) تجزیه و تحلیل استاتیک (بررسی reviews ، بازبینی audits ، بازرسی inspections و walkthroughs) ذاتا فاقد دقت لازم برای تشخیص تمامی خطاهای موجود در بعضی از نرم افزار های مشخص، می باشند. از سوی دیگر، روش های رسمی به طور باید و شاید مناسب نیستند زیرا آنها ارزشی برای مولفه انسانی قائل نمی باشند.

امروزه، برای اثبات اینکه یک برنامه بزرگ و جامع نیازمندیهای مربوط به ایمنی سیستم اتوماسیون را بر آورده می کند، مثلا صحت کارکرد و دقت نتایج به همراه در نظر گرفتن خواسته های مقطعی و زمانی محیط ,تنها فن ترجمه معکوس میتواند تاییدی بر این نرم افزار باشد. (Krebs and Haspel,1984) که بطور مشترک توسط Rheinland, Cologne, Gesellschaft ftir Technischer Uberwachungsverein (TUV) Reaktorsicherheit, Garching (همه در آلمان) و موسسه Energitechnikk, Halden در نروژ، در پروژه آزمایشی نیروگاه هسته ای هالدن توسعه داده شده است. این روش تا حدودی، معیارهای ارگونومیک را برآورده می کند و فرض می شود که نه تنها موثرترین بلکه مناسبترین و عموما شناخته شده ترین روش (با مجوز مقامات) برای تایید برنامه ها می باشد.

صدور مجوز ایمنی برنامه های نوشته شده در زبان های سطح بالا، زمانی که هنوز کامپایلر وجود نداشت، برای عملکرد های صحیحی که ثبت شده، ممکن نیست. به همین دلیل باید استفاده ی اسمبلرها، لینکرها و سیستم عامل ها در گسترش و کاربردهای امنیت که به نرم افزار مربوط می شود را معرفی کرد. از این طریق برای پیدا کردن مشکل کامپایلرهای که که مجوز غیر امن داشتند، می بایست صحت تحقیقات بر پایه ی کد شی گرا بود. برای مثال، تنها فرم برنامه ها در دسترس ماشین ها بود تا آن ها را اجرا کنند. مواجه با این محدودیت ها ترجمه ی برعکس شامل خواندن برنامه های ماشین خارج از حافظه ی کامپیوتر و تحویل آنها به تعداد مختلفی از تیم های بازنگری که بدون هیچ نقطه اشتراکی کار می کنند، می شود. این تیم ها به طور دستی کد را تجزیه و دوباره کامپایل می کنند و خصیصه های اصلی را دوباره بازسازی می کنند. یک مجوز امنیتی به یک سیستم نرم افزاری اعطا می شود اگر خصیصه های اصلی اش با معکوس تمام خصوصیات بدست آمده، سازگار باشد.

اگر چه، چیزی که توسط Dahll و همکاران (1998) و Pofahl (1994) گزارش شده، برای اجرای برنامه های غیر بدیهی بر روی ماشین های ون نیومن، این روش به طور کلی دست و پا گیر، زمانبر و گران است. این منجر به ایجاد یک شکاف معنایی بین خصیصه ها، فرموله شده بر اساس توابع کاربر و نیازمندی های امنیتی، و پیاده سازی کدهای ماشین سنتی بر حسب حافظه ،دستیابی حافظه و عملیات های سطح پایین دیگر، می شود. از

سوی دیگر، ترجمه معکوس، احتیاجی به دانش تخصصی ندارد. مجوز امنیتی به لحاظ قانونی، مهم است و باید به آن توجه داشت.در طول قضاوت، داوران و هیئت منصفه لازم نیست تنها به توصیه افراد خبره تکیه کنند بیکه می توانند نتایج خود را اعلام کنند.

تجربیات ذکر شده نشان می دهد، که تلاش برای بررسی کلاسیک "کد اسپاگتی" روش نامناسبی است. روش بهتر مبتنی بر درخواستی که محاسبات باید برای مردم ارائه دهند، استفاده از الگوهای برنامه نویسی که تصدیق را ساده تر می کند. دو مثال از این الگوها در قسمت های بعد معرفی خواهند شد. استفاده ازآنها(تلاشهای مورد بحث از لحاظ اقتصادی توجیه پذیر است) از اینرو یک مرحله ی ساده منجر به بازگشت مستقیم از کد ماشین به سطح خصوصیات مسئله، می شود. اصول سادگی و شفاف سازی به عنوان دستورالعمل های اساسی کافی است زمانی که انتخاب یک روش تایید برای پیاده سازی یک نرم افزار،امنیت سیتم های مرتبط را پشتیبانی کند. این به این خاطر است که سادگی یک فرآیند تایید نرم افزار، به طور مستقیم با قابلیت اطمینان کامپیوترهای مبتنی بر سیستم های کنترلی مهندسی شده، مرتبط است. این واضح است که ترجمه ی معکوس، سادگی مورد نیاز را برآورده می کند. می توان نشان داد که آن بسیار مفید می شود، زمانی که با الگوهای برنامه نویسی و معماری تضمین شده،ترکیب شود.

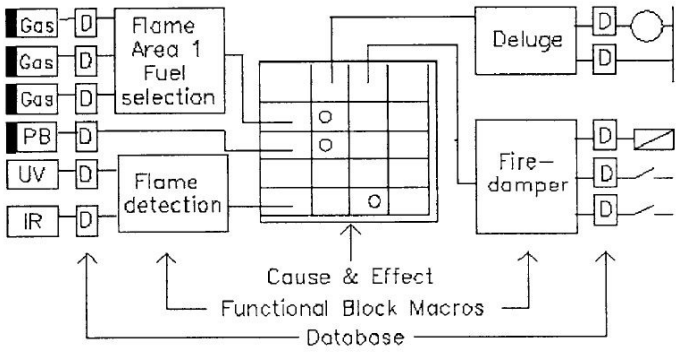
بیشترین پروسه خلاق در یک پروژه ی نرم افزاری،تحلیل نیازمندی ها و خصیصه ها است. در ابتدا یک خصیصه ی بدون ابهام و کلی ساخته می شود، که رفتار سیستم را با جزییات توصیف میکند تا توسعه یابد. معانی این چنین توصیف هایی نباید در مرحله طراحی و پیاده سازی تغییر پیدا کند. این مشاهدات به جنبه ی اجرای مستقیم نیازمندی های خصیصه بر میگردد(یا پیاده سازی که میتواند بوسیله ی انجام تبدیلاتی روی نیازهای خصیصه ها بدست آید)که به آسانی میتواند درستی آن، ثابت شود. این رویکرد در بخش های بعدی پرداخته خواهد شد. مزایای آن آشکار و واضح است. مراحل فشرده کار طراحی، پیاده سازی و تایید غیر قابل اعتماد ناپدید می شوند. در عوض، پروسه ی توسعه شامل تحلیل نیازمندی ها ،خصیصه ها و دسترس پذیری می شود. مستندات سیستم اولیه شامل ویژگی نیازمندی ها می شود. نگهداری دیگر روی کدها انجام نمی شود. این ویژگی بود که انتقال و دوباره پیاده سازی را تغییر شکل داد،فقط در طی فاز اصلی.

جداول علت و معلول

یک الگوی برنامه نویسی که بنظر می رسد تا برای حضور نیازمندی هایی با بیشترین امنیت مناسب باشد، بوسیله جداول علت و معلول تشکیل شده است. این مفهوم به خوبی در مهندسی عمدتا به عنوان روشی برای

ساخت سیستم های حفاظتی، ایجاد می شود. (برای مثال، سیستم های خاموش کردن فوری،که بایستی در برابر واکنش که وقوع موقعیت های خطرناک مستفل کاربردی، پاسخگو باشد.) بمنظور حفظ سیستم و محیط آن از خرابی های جدی. نرم افزار مبتنی بر جداول علت و معلول، بواسطه حداکثر سادگی و شفافیت مشخص می شوند; تا حدی که، تایید آن با نظارت مستقیم و اجماع گسترده برای هر دو نوع اعتبار و صحت پیاده سازی ممکن باشد.

طبق تحقیقاتی که در شکل ۱به عنوان مثال مطرح شده است، نرم افزار برای سیستم های حفاظتی،به صورت جداول تصمیم گیری معرفی شده،که جداول علت و معلول نامیده می شود. ردیف های آنها مربوط به رویدادها می شودکه رویدادها منجر به پیش شرط های بولین می شود. با استفاده از مارک کردن فیلدها،که به ستون های اصلی تعلق دارند و با اعمال مخصوص مرتبط اند،کاربر می تواند پیش شرط ها و خصیصه هایی که بر طبق اعمالی که انجام شده انتخاب کند که همه ی پیش شرط ها درست باشند.



شکل ۱ – یک نوع جدول علت و معلول

محتوای جداول علت و معلول تنها نرم افزار کاربردی پیاده سازی شده در سیستم های حفاظتی می باشد. در اینجا نرم افزار معنای برنامه ی که واکشی شود با استفاده از حافظه ی قابل نوشتن به عنوان معماری کامپیوتر وون نیومن نیست. در عوض، آن به عنوان یک پارامتر که توسط یک کنترلر چند منظوره که برای اجرای توابع خاص تنظیم شده است، مشخص می شود. از آنجا که جداول علت و معلول بایستی در حافظه های فقط خواندنی به دلیل نیازهای امنیتی، نگه داشته شوند. این نوع نرم افزار بطور کلی در قالب Firmware فرض می شود.

به دلایل زیر، ما جداول علت ومعلول بررسی می کنیم تا برای پیاده سازی کامپیوتر مبتنی بر سیستم های کنترلی با نیازهای امنیتی، ایده آل باشد.

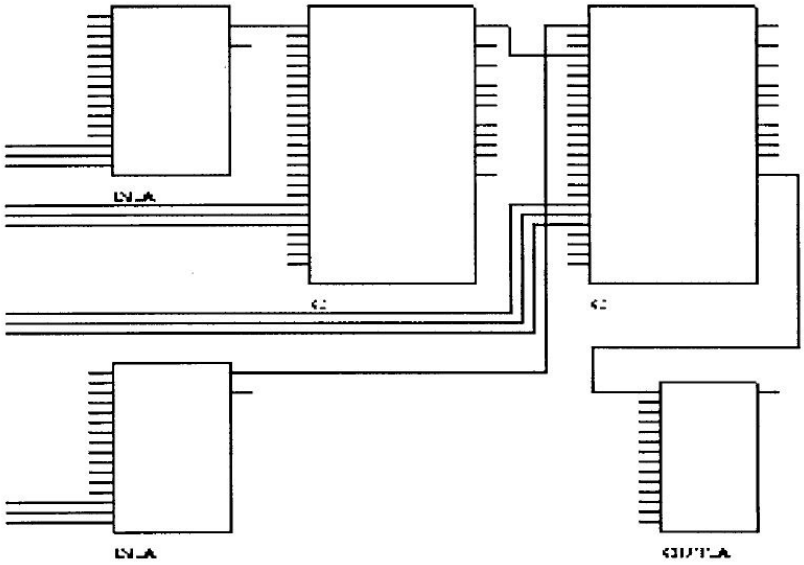
- خصوصیات به صورت قابل فهم فرموله شدند اما با وجود روش رسمی ،یعنی به عنوان جداول تصمیم.
- ازین رو، تایید خصوصیات بوسیله توافق اجتماعی، ساده است.
- عملیات مشخص شده، می توانند بطور مستقیم بوسیله ی ماشین و بدون نیاز به تبدیلات پیچیده و تایید صحت پیاده سازی، تفسیر و اجرا شوند.

از آنجا که این الگو کمک می کند تا نیاز به تایید نرم افزار عموما پیچیده به سادگی ناپدید شود، پس تکنولوژی اتوماسیون بایستی از جدول علت و معلول تا حد ممکن بطور گسترده استفاده کند.

دیاگرام های بلوک تابعی

دومین الگوی برنامه نویسی که به نرم افزار ها اجازه می دهد تا به راحتی و با توجه به کد منبع و شی در قالب بلوک دیاگرام تابعی زبان (FBD) که در دسترس است، قابل فهم باشند. این در استاندارد بین المللی IEC (1992) 3-1131 به عنوان یکی از دو زبان گرافیکی برای کد نویسی کنترلرهای منطقی قابل برنامه سازی، تعریف شده است. براساس یک سنت طولانی مدت در مهندسی کنترل، این نوع از برنامه نویسی گرافیکی قبلا در تکنولوژی اتوماسیون بخوبی ایجاد شده است. دیاگرام های بلوک تابعی از نقشه ی مدارات دیجیتالی، جایی که هر چیپ بصورت یک ماژول خاص از یک عملکرد کل، نشان داده می شود، نتیجه می گردد. بنابراین رویه برنامه نویسی ساده، بسیار شبیه به گردآوری سنتی سیستم های کنترل از مولفه های سخت افزاری وابزارها (مثل سیم کشی تنظیم کننده آنالوگ ، تقویت کننده های عملیاتی، رله های الکترومغناطیسی، مولفه های TTL و غیره) یا هر متن دیگری روی تجهیزات کنترلی کلاسیک می باشد. اینجا اگرچه، بلوکهای تابعی معادل های نرم افزاری مولفه ها و ابزارهای کنترلی متداول، هستند. مولفه های اساسی زبان در FBD ها، نمونه ای از توابع و بلوک های تابعی هستند که با هدف ترکیب برنامه، توسط خطوط ارتباطی بین ورودی و خروجی شان، به هم متصل می شوند. توابع و بلوک های تابعی، بعنوان مولفه های زبانی کاربرد گرای سطح بالا و قابل استفاده مجدد ارائه می شوند. توابع و بلوک های تابعی متفاوت از هم هستند تا جایی که، توابع حس ریاضی برای تفسیر شدن هستند، از این رو هیچ وضعیت داخلی ندارند، در حالی که بلوک های تابعی ممکن است در قالب نمونه های متمایز وجود داشته باشد. نمودارهای تابع گرا، حاصل از این شیوه توسعه برنامه، برداشت مستقیم از کاربردهای وابسته به جریان داده های اساسی را فراهم میکند. شکل۲، نمونه ای از سگمنت برنامه ترکیبی را بر اساس رویه های توضیح داده شده در بالا، نمایش می دهد. برنامه های ارائه شده در FBD، ساختار پیمانه ای

دارند و قادر به تسهیل توسعه و تایید صحت نرم افزار بدلیل ارزش ذاتی مستنداتشان هستند(بعنوان مثل: ظرفیت آن ها برای منعکس کردن ساختار مسئله اصلی).



شکل ۲ – مثالی از دیاگرام های بلوکی تابعی

بلوک های تابعی ذاتا، زیر روال هستند و در طی اجرا یک یا مقدارهای بیشتری را بر می گردانند. نمونه هایی با چندین نام از بلوک های تابعی (برای مثال، Coping) می تواند ایجاد شود. تمام مقادیر متغیرهای خروجی ومتغیرهای ورودی، از یک اجرای یک نمونه از بلوک های تابعی به نمونه بعدی، انتقال می یابند. بنابراین، وجود بلوک های تابعی با آرگومان های یکسان لزوما مقادیر خروجی یکسانی را تولید نمیکند. ضروری است تا قادر به بیان بازخوردهای داخلی و رفتار ذخیره سازی باشیم. تنها متغیرهای ورودی و خروجی در خارج از یک نمونه بلوک های تابعی، قابل دسترسی هستند (مثال: متغیرهای داخلی بلوک های تابعی از بیرون مخفی هستند).

برای فرموله کردن برنامه های اتوماسیون،کاربر تنها اعمال درخواست ،جایگذاری و اتصال نمونه های بلوک تابعی بهم را از کتابخانه های داده شده، انجام می دهد که نتایج در دیاگرام هایی که شبیه به دیاگرام های جریان داده ای استفاده شده برای مدل سازی تابعی یا فعالیتی، خلاصه می شود. جعبه هایی که در نمودار بلوک تابعی شکل۲ نشان داده شده، فعالیت های برنامه را بیان می کند، درحالی که خط ها یک جریان یک طرفه(چپ به راست) از نوعی از اطلاعات ضروری برای بلوک های تابعی را برای انتقال این فعالیتها، مدل می کنند. بلوک های تابعی منحصر بفرد بر اساس سفارش جزئی ارائه شده توسط Wiring راه اندازی می شود و به همین دلیل،آن ها داده ها را از خط های متصل عبور می دهند. خطوط ممکن است برای نمایش fan-out شاخه شاخه شوند، جایی که چندین کپی یکسان از اطلاعات به طور تصادفی به بلوک های تابعی دیگر به عنوان مصرف

کننده آن اطلاعات، تحویل داده شود. اما خطوط ممکن نیست fan-in ها را بهم متصل کند و یا داده های ورودی را ادغام کند.

علاوه بر ارائه ثابت ها به عنوان پارامتر، (از قبیل "bar" متصل شده به ورودی خارجی XUNIT بلوک تابعی B1 از شکل ۷) نمونه های بلوک تابعی و جریان های داده ای بین آن ها، تنها مولفه های زبانی استفاده شده در این الگوی برنامه نویسی هستند. توسعه نرم افزار در فرم گرافیکی با استفاده از ابزارهای CAD مناسب، انجام میشود. زمانی یک نمودار رضایت بخش است که تبدیلات کامپایلر به طورگرافیکی منطق برنامه را به صورت کد شیئی نمایش دهد. چون این منطق تنها قادر به فرض کردن ساختار ساده است، برنامه های به روزشده شامل هیچ خصیصه ای ازترتیب فراخوانی رویه و حرکت داده ی داخلی، نیست.

بعنوان یک مشخصه ی ضروری الگوی برنامه نویسی، FBD مبتنی بر قابلیتهای خود جهت تسهیل مدیریت سیستم های بزرگ نشان داده شده در شکل ۳،که بلوک های تابعی در واحدهای جدیدی متصل شده اند، هست. دوما بلوکهای تابعی در انتزاع بالاتری تشکیل می دهند وممکن است در نمودار های دیگر با انتزاع بیشتر وجزئیات پیچیده بیشتری که مبتنی بر کاربرد درسطح بالاتر است، اتفاق بیفتد. بدین صورت قوانین سخت، ترکیبات سلسله مراتبی نمودارها را کنترل می کنند. در نتیجه، پیچیدگی درون رابطه ای نمودار های منفرد را همیشه می توان پایین نگه داشت. علاوه بر این، اصول بکار رفته در مهندسی نرم افزار مبتنی بر قابلیت استفاده مجدد، تعداد راه حل های ممکن برای حل مشکل داده شده از راه های مختلف را کاهش می دهد.

یک کتابخانه بلوک تابعی شامل مولفه های استاندارد دارای ویژگی های زیر است:

- عموما قابل اجرا برای اهداف اتوماسیون
- معمولا ارائه شده توسط فروشندگان سیستم
- مولفه های خاص پروژه تعریف شده توسط کاربر

دو نقطه بی نهایت در دامنه رویکردهای ارائه شده جهت تعریف برخی کتابخانه ها ، در واقع، یکی از آنها تعداد اندکی از بلوک های تابعی بسیار پیچیده با امکان انتخاب الگوریتم های خاص با پارامترهای متغیر، که بعضی اوقات این تعداد به بیش از ۵۰ می رسد، و دیگری تعداد زیادی از بلوک های تابعی کاملا ساده با حداکثر چند ورودی بعنوان مثال ۱۰ پارامتر، وجود دارد. همانطور که میبینیم در نتیجه، قابلیت استفاده مجدد کتابخانه ها، برای مسئله ارائه مجوز نرم افزاری، حیاتی است. کتابخانه ها توسط تولید کنندگان پیشرو که "کاملا محلی[1]"

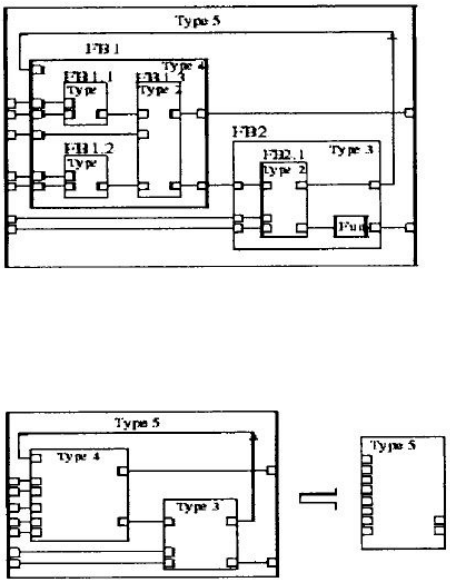
[1] Locally complete

هستند بدین معنی که عملا تمام مشکلات فردی که در یک محیط کاربردی خاص اتفاق می افتد، می تواند بوسیله ی آنها حل شود. وضعیت Partically All یک امر تجربی است که طی سالها از روی برنامه های متعدد توسعه داده شده است. اساسا برای هر محیط کاربردی یک سری بلوک های تابعی خاص وجود دارد، اگرچه برخی ماژولها شبیه به ورودی و خروجی های آنالوگ و دیجیتال است که ارتباط عمومی بیشتری دارند. بعضی از بلوک ها ممکن است با منابع ماشین گره خورده باشند، مانند سنسورها، محرک ها، کانال های ارتباطی یا واسط های بین ماشین و انسان. آنها توسط بلوک های IO در نمودارهای بلوک تابعی نماش داده می شوند. معمولا، تعداد نسبتا کمی از مولفه های کتابخانه برای فرموله کردن تمام برنامه ها در درون محیط معینی از اتوماسیون فرآیند، کافی است. برای مثال همانطور که راهنمای VDI/VDE 3696 (۱۹۹۵) نشان میدهد، برای مهندسی شیمی این تعداد ۶۷است. لیست زیر بلوک های تابعی تعریف شده در راهنما می باشد که یک تصوری از کارکردهای عمومی را ارائه می دهد:

۱. توابع ریاضی Monadic : مقدار صحیح، کوسینوس، سینوس، تابع نمایی، اگوریتم مبنای ۱۰، ریشه جذر، حد، تابع غیر خطی ثابت، مقیاس خطی
۲. توابع ریاضی Polyadic : جمع، تفریق، ضرب، تقسیم، عاد کردن، توان
۳. مقایسه ها : تساوی، بزرگتر مساوی، بزرگتر مساوی،کمتر مساوی، کمتر، نامساوی
۴. تابع بولین Monadic : خنثی سازی
۵. توابع بولین Polyadic : ترکیب عطفی، ترکیب فصلی، Antivalence
۶. تشخیص دهنده لبه : مشخص کردن لبه ی بالا رونده و پایین رونده
۷. توابع Selection : انتخاب حداکثر و حداقل، انتخاب بوسیله ۱ از N بیت، دی مالتی پلکسر برای بولین ها و اعداد، مالتی پلکسر برای بولین ها و اعداد، انتخاب باینری برای بولین و یا اعداد
۸. شمارنده ها، مونو استیبل ها، بی ایستیبل ها، تایمرها : شمارنده بالا رونده و پایین رونده، شمارنده جریان، مولفه های بی استیبل با ست و ری ست، ماجولاتور مدت پالس، تاخیر های روشن و خاموش، مولفه های Non-re-triggerable monostable
۹. فرآیند ورودی و خروجی : ورودی/خروجی آنالوگ، ورودی/خروجی باینری، ورودی/خروجی دیجیتال، ورودی ضربه
۱۰. ارتباطات شبکه ای ورودی خروجی : ورودی/خروجی ارتباط برای مقادیر بولین و عددی

۱۱. تنظیم کننده ها و مولفه های پویا : کنترلرهای استاندارد و جهانی (PID-T1)، میانگین زمان اجرا، مهلت زمانی، تمایز با تاخیر (D-T1)، انتگرال (I)، Lead Lag (PD-T1)،دومین سفارش

۱۲. شرایط برای نمایش و عمل : محدودیت با آلارم یا ذخیره پیغام، روند ثبت، مقدار ورودی دستی با سوییچ و محدودیت



شکل ۳ – بلوک های تابعی با ۲ یا ۳ سطح از جزییات

پیچیدگی مولفه های کتابخانه معمولا قابل مدیریت است، چون نرم افزار معمول در محیط های کاربردی خاص، محدود به متغییر هاست. در برنامه نوشته شده در زبان سطح بالای پاسکال، ماژولهای نرم افزار کاملا کوتاه هستند: سورس کد آنها از دو صفحه بیشتر نخواهد شد. تکرارهای زیاد و توابع بازگشتی در این ماژول ها رخ نمی دهد. بنابراین، صحت آنها می تواند با یک تلاش قابل تحمل ثابت شود(مثال: با استفاده از حساب دیفرانسیل و انتگرال گزاره، و همچنین اجرای نمادین، و یا در بعضی مواقع با توابع بولین حتی تست کامل).

در مثال دیگر، برنامه نویسی سیستم های شات دان اورژانسی، که معمولا بصورت گرافیکی در قالب نمودار های توابع منطقی نمایش داده می شود،که نگاشت از ورودی بولین به خروجی بولین به عنوان توابع زمان را توصیف می کند. از قبیل :

```
If    a pressure is too high
Then  a valve should be opened
and   an indicator should light up
after 5 seconds
```

آن به ۴ بلوک تابعی احتیاج دارد، بعبارتی، ۳ عملگر بولین و یک تایمر. برای درک عمیق تر مفهوم بلوک تابعی ما به منبع (Zoller، ۱۹۹۱) مراجعه می کنیم.

استفاده از برنامه نویسی FBD برای تایید نرم افزار، با توجه به تایید نرم افزار ضرروریست که در زبان برنامه نویسی FBD دو گام از فرآیند توسعه بیان شود :

۱. ساختن یک کتابخانه از بلوک های تابعی- تنها یکبار برای محیط کاربردی معین
۲. کاربرد روابط داخلی خاص از نمونه های بلوک تابعی

و همچنین در فاز تایید منتشر می شود. بر این اساس در ۲ مرحله ی بالا:

۱. تایید رسمی همه ی مولفه ها در کتابخانه بلوک های تابعی داده شده، با بکارگیری متدهای رسمی مناسب قبل از انتشار آن.
۲. برای هر برنامه کاربردی ارائه شده، تایید بوسیله ی ترجمه بازگشتی معکوس یک پیاده سازی مناسب از الگوی بین ارتباطی خاص از نمونه های بلوک های تابعی ایجاد شده(برای مثال یک جریان داده معین).

استفاده از ترجمه بازگشتی معکوس برای تایید نرم افزار توسط معماری مبتنی بر مسئله معرفی شده در قسمت بعدی معرفی می شود، آسان شده است.

معماری امن گرا

به عنوان یک معماری برای سیستم کنترلی کامپیوتر امن گرا، ما استاندارد میکرو کامپیوتر با یک مفسر سفت افزار که یک مجموعه از بلوک های تابعی پایه را پیاده سازی می کند، انتخاب می کنیم. برای اهداف رویایی اجرای نرم افزار نشان داده شده در فرم نمودارهای بلوک تابعی، مفسر تنها نیاز به اجرای ۲ دستورالعمل در سطح برنامه نویسی کاربر دارد :

GET<operand-address>

PUT<operand-address>

یک بلوک تابعی ذکر شده به یک بخش برنامه شی که شامل یک تعداد از پارامترهایی که از محل RAM یا ROM با استفاده از دستورالعمل GET واکشی می شود، توسعه می یابد و یک تعداد از ذخیره سازی نتایج عملیات با کمک دستورالعمل های PUT انجام می شود. ابتدا، مفسر شماره شناسایی بلوک تابعی درخواستی را واکشی می کند، از این طریق تعداد مناسب پارامترهای ورودی و در صورت نیاز،متغیرهای وضعیت داخلی بلوک مشتق شده و سپس واکشی می شوند. سپس پیاده سازی برنامه هدف برای بلوک تابعی اجرا می شود. مفسر

تمام دستکاریهای لازم داده و ارتباطات با محیط را انجام می دهد. تشریح بلوک تابعی با ذخیره سازی نتایج و وضعیت جدید داخلی در RAM تمام می شود.

در نتیجه برای جلوگیری از هرگونه تغییرات بوسیله ی توابعی که درست عمل نمی کنند، در این معماری همه ی برنامه ها باید در حافظه های فقط خواندنی جمع آوری شوند. به دلایل کاربردی، در کل ۲ نوع از این حافظه ها وجود دارد. اگرچه در کل برنامه Ram وجود ندارد. به جای آن، کد بلوک های تابعی پایه در پوشش ROM سفت افزار برنامه ریزی شده مستقر شده اند که تحت نظارت مراجع مجوز دهی تولید و منتشر شده است. از طرف دیگر، ترتیب اجرای بلوک ها همراه با پارامتر پسینگ که کاربر را معرفی می کند. این قسمت از نرم افزار موضوعی است برای تایید دوباره که بوسیله مراجع مجوز دهی انجام می شود که سرانجام نیاز به نصب و مهروموم PROM ها در کنترل پروسه ی کامپیوترها دارد. این تنظیمات حافظه برنامه به طرز فیزیکی جدای از ۲ قسمت نرم افزار انتخاب می شود : یکی با دامنه ی عمومی که فقط احتیاج به تایید یک باره دارد و دیگری خصیصه ی کاربردی است. معماری بالا می تواند در سخت افزار نیز پیاده سازی شود. این روش بوسیله ی کنترل کننده منطقی قابل برنامه ریزی برای کاربرد های امنیتی بحرانی دنبال می شود.

گواهی امنیت نرم افزار

همه ی مولفه های یک مجموعه از بلوک های تابعی مشمول در یک کتابخانه توسط روش های رسمی تایید می شود. این مجوز امنیتی پرهزینه باید یک بار انجام شود، بعد یک مجموعه بلوک تابعی معین برای یک محیط کاربردی معرفی و استاندارد می شود. هزینه مجوز دهی می تواند، از این طریق، بر روی بسیاری از پیاده سازی هایی که مربوط به هر پروژه اتوماسیون مستقل کم هزینه است، گسترش پیدا کند. در واقع بلوک های تابعی هم که در تمرین اتفاق می افتند، پیچیدگی کم تری دارند و به همین دلیل، صحت درستی شان در واقع قراردادی است. به عنوان جزییات پیاده سازی برای بلوک های تابعی یک قسمتی از سخت افزار به کار رفته است.آنها از دید برنامه های کاربردی پنهان می مانند و بیشتر از این، نیاز به مجوز امنیتی ندارد.

از این رو، برای هر برنامه کاربردی جدید،فقط پیاده سازی مناسب یک الگوی بین ارتباطی نمونه های درخواستی بلوک های تابعی احتیاج دارد تا صحیح شود. خوشبختانه حذف نیاز برای یک-که هنوز غیر ممکن و بنابراین در حال حاضر غیر قابل دسترس است-کامپایلر لایسنس شده امنیتی که نرم افزار گرافیکی که به کد شی به عنوان یک پیش شرط لازم برای به کارگیری نمونه نمودارهای بلوک تابعی است،تبدیل می کند. اگرچه ممکن است که به عنوان یک روش بروت فورس غیر زیبا مطرح شود،ترجمه ی بازگشتی معکوس مخصوصا مناسب برای تایید

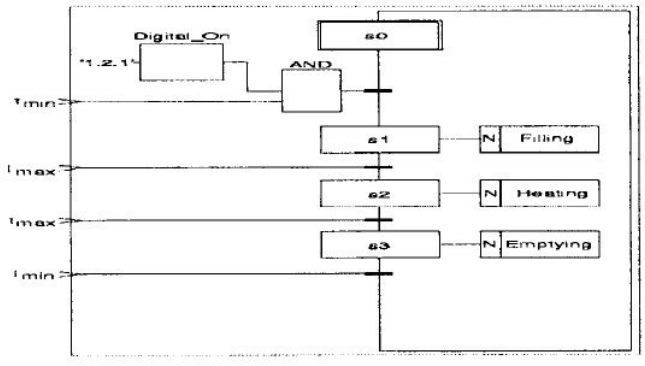
یک پیاده سازی درست برنامه های مشخص گرافیکی روی معماری معرفی شده ی بالا، می باشد. این به خاطر دلایل زیر است:

این روش غیر رسمی، به راحتی قابل درک و بلافاصله قابل اجرا بدون هیچ آموزشی می باشد، بنابراین بطور کامل مناسب برای استفاده در سطح برنامه نویسی کاربردی بوسیله افرادی با بیشترین پیش زمینه آموزشی ناهماهنگ می باشد. سهولت فهم و استفاده ذاتا متدهای کاربردی بدون خطا را گسترش می دهد.

بوسیله برنامه نویسی در فرم نمودارهای بلوک تابعی، ویژگی ها مستقیما به کد شی که شامل فراخوانی رویه هاست و پارامتر های ارسالی نگاشت شده در حالی که یک دستورالعمل ماشین متعارف خارج شده از چهارچوب برنامه ،هدفش را نشان نمی دهد، رخداد یک نمونه بلوک تابعی معمولا نشانه ای از وجود مشکل، راه حل آن و نقش بلوک ها در آن است، از زمانی که برنامه نویسی گرافیکی مبتنی بر بلوک های تابعی کاربرد گرا که کیفیت سطح مشکل ویژگی را داشت و به همین دلیل، بواسطه طراحی، هیچ شکاف معنایی در معماری ما بین سطح مبدل(فراهم شده توسط مفسر) ماشین و انسان وجود ندارد، ترجمه ی بازگشتی معکوس منجر به بازگشت به عقب در یک مرحله آسان از کد ماشین به تعریف دوباره مسئله در فرم گرافیکی می شود. در نتیجه تعادل با ویژگی اصلی می تواند به راحتی چک شود.

رفتار موقتی

از بسیاری از برنامه های اتوماسیون بهمراه ایمنی مرتبط با آن به عنوان کنترل های ترتیبی ساخته شده از مراحل و انتقالات بین ارتباطی بوسیله لینک های مستقیم یاد می شود. برای مثال، ما به موارد ساده بررسی می کنیم، اما همچنین وظایف اتوماسیون معمولی را رها می کنیم(VDI/VDE 3696, 1995). اگر یک دکمه استارت فشار داده شود و دما کمتر از ۴۰ درجه سانتیگراد باشد- در فاز یا مرحله ۱- دریچه پر کردن یک کانتینر باز خواهد شد تا اینکه وضعیت پر بودن آن به ۹۰ درصد برسد. بعد ازآن- در دومین مرحله-کانتینر گرم شده با بخار توسط دریچه گرمایی باز شده تا محتوای دمای کانتینر به ۹۰درجه سانتیگراد برسد- در سومین مرحله- شیر تخلیه کانتینر باز شده تا اینکه وضعیت پر بودن آن بیش از ۵درصد کاهش یابد. سپس کنترل به وضعیت آغازین-مرحله اول بر می گردد. شکل۴ یک برنامه کنترل گام مربوطه را در قالب یک نمودار تابعی ترتیبی، زبان برنامه نویسی گرافیکی منحصر به فرد، نشان می دهد که در استاندارد بین المللی-IEC 1131 3(1992) تعریف شده بود.

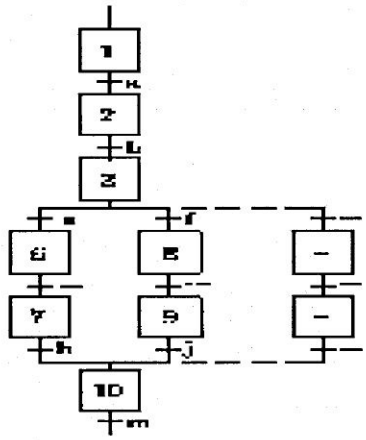


شکل ۴ - کنترل دنباله ای اتوماسیون کانتینر

پس از آغاز یک برنامه ساخت یافته با چارت تابع ترتیبی، اولین مرحله آن فعال می شود.زمانی که یک مرحله فعال می شود، یک بخش سازمانی برنامه مربوطه بیان شده در قالب نمودار بلوک تابعی اجرا می شود. به عنوان بخش پایانی این مطلب، وضعیت انتقال نسبت داده شده به انتقال بین یک مرحله و جانشین آن در نمودار تابعی ترتیبی، ارزیابی می شود. اگر وضعیت انتقال برآورده نشود، آن مرحله فعال باقی می ماند و نمودار بلوک تابعی دوباره اجرا می شود. هر زمان که وضعیت انتقال بعد از یکی از این تکرارها برآورده شود، مرحله غیرفعال می شود و یکی دیگر بلافاصله پس از انتقال در امتداد یک خط مستقیم، فعال می شود.

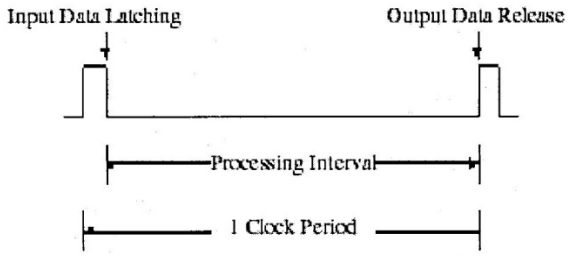
بسیاری از ویژگی های سیستم های محاسباتی مدرن مثل Piplining، حافظه های نهان[1]،دسترسی مستقیم حافظه، و انجام چند وظیفه ای غیر همزمان ، و از این رو گواهی امنیتی، رفتار موقتی سیستم های اتوماسیون کامپیوتری بطور نا امید کننده ای پیش بینی شان مشکل است. تنها راه برون رفت از این وضعیت قابل قبول برای سیستم های مرتبط با ایمنی، سادگی است: تمام ویژگی های مضر بایستی چشم پوشی شوند و یک روش سخت گیرانه که وضوح و سادگی رفتار موقتی را فراهم و تضمین می کند، باید رعایت شود که سازگار با هر شرایط محیطی باشد. ما پایه تعریف چنین روشی را بر روی مفهوم نمودارهای تابعی متوالی و مد اجرای استفاده شده توسط آنها، استوار می کنیم. فقط توالی های خطی از مراحل و شاخه های جایگزین از چنین دنباله ها یی مجاز می باشد.

[1] Caches



شکل ۵ – یک نمودار تابعی ترتیبی

همانطور که در (۱۹۹۲) IEC 113-3 نیز معرفی شد به دلایل امنیتی، شاخه های موازی باید با موازی سازی سخت افزار پیاده سازی شوند یا در زمان طراحی با سری‌سازی صریح حل شوند. روش اتخاذ شده در شکل ۶ به تصویر کشیده شده است. اگرچه نسبتاً محدود کننده است اما کاملاً قابل پیاده سازی است چون اکثر کنترل گرها و سیستمهای بلادرنگ (Real Time) به روش دوره ای کار می کنند. علاوه براین، موجب تنزل بازبینی رفتار زمانی سیستمهای کامپیوتری به یک تست معمولی می شود.



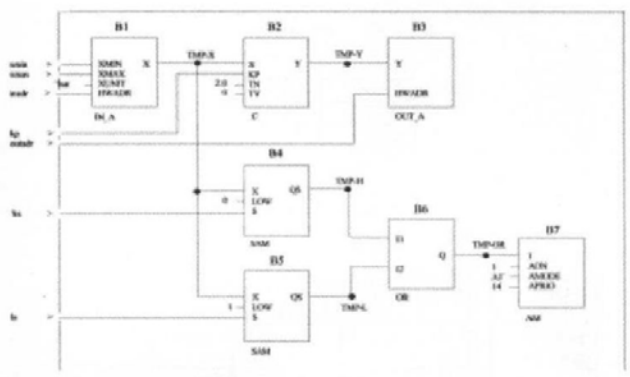
شکل ۶- عملیات کنترل تناوبی دقیق تر

شکل ۶ عملیات کنترل کاملاً تناوبی این روش مبتنی بر کلاک (clock) سخت افزاری است که در آن ،دوره های پردازش علامتگذاری می‌شود. طول سیکل به اندازه زمان اجرایی طولانی‌ترین مرحله اجرای نرم‌افزار انتخاب می‌شود. علاوه براین، مجموعه دستورالعمل های مفسری که معماری مبتنی بر ایمنی را پیاده سازی می‌کند و در بالا معرفی شده، به وسیله دستور STEP بدون هیچ عملوندی بسط یافته است. همانطور که در قالب نمودارهای عملیات متوالی (ترتیبی) شرح داده شد برنامه هایی که به وسیله مفسرها اجرا می شوند شامل مجموعه ای از گامها است. یک دستورالعمل STEP، قبل از تکه برنامه هر گام اضافه می گردد که کار این دستورالعمل بررسی

این مسأله است که آیا این تکه برنامه در محدوده زمانی یک سیکل اجرا شده است یا خیر. اگر زمان اجرای آن تکه بیش از یک سیکل گام شود، برنامه خاتمه پیدا کرده و پروسه تحت کنترل در حالت امن پایان داده می شود. عموماً اجرای تکه کد قبل از رسیدن سیگنال گام بعدی تمام شده و منتظر رخداد سیگنال کلاک که بیانگر شروع گام بعدی است می‌شود. وقتی نهایتاً سیگنال گام بعدی می‌رسد بسته به مقدار فعلی شرایط گذر،تصمیمی مبنی براینکه همان گام باید تکرار شود یا گام منطقی بعدی باید اجرا شود، اتخاذ می‌گردد. از آنجا که انشعاب برنامه فقط به همین شیوه محدود می شود از هرگونه دسترسی نامناسب به کد گامهای غیرفعال دیگر و مکانهایی غیر از شروع گام به طرز مؤثری جلوگیری می‌شود. اگرچه مشخص کردن سیکل گام باعث تعیین دقیق اجرای دوره‌ای گامهای تکی می‌گردد، در عین حال مدت پردازش عملیات مختلف در یک سیکل می تواند متغیر و در نتیجه غیرقابل تعیین باشد. از آنجا که رفتار زمان بندی قابل پیش بینی فقط برای خارج مجموعه مهم است (برای عملیات ورودی و خروجی) تعیین دقیق زمان بندی به شیوه زیر انجام می‌پذیرد. عملیات ورودی و خروجی توسط وسایل مستقلی انجام می‌پذیرد که با کلاک همزمان شده است. وسایل ورودی همیشه داده های ورودی فعلی را فراهم می‌کند و بدون توجه به اینکه در یک دوره خاص مورد استفاده قرار می گیرند یا نه، به ابتدای دوره‌های پردازش منصوب شده و تشکیل به اصطلاح تصاویر پردازش را می دهند. داده‌های خروجی محاسبه شده ابتدا بافر می شوند و سپس توسط کلاک پالس بعدی که اجرای برنامه را از سر می گیرد آزاد می شوند.

یک مثال عملی

برنامه کاربردی ترجمه به زبان سطح پایین با کمک یک مثال ساده و در عین حال واقعی تشریح می شود. سطوح نمایش متفاوت از یک برنامه، به عنوان مثال نمودار بلوک تابعی، لیست شبکه ای و کد هدف جهت مفسر در معماری ما با جزئیات نشان داده شده است. بدیهی است که کشیدن یک نمودار بلوک تابعی از روی یک برنامه هدف که کارش انجام ترجمه به عنوان یک روش بازبینی نرم افزار است، بسیار ساده و آسان خواهد بود.



شکل ۷ - برنامه نظارت و تنظیم فشار

شکل ۷ در قالب گرافیکی به نمایش یک برنامه اتوماسیون صنعتی نمونه می پردازد. این برنامه نظارت و تنظیم فشار را انجام می دهد و به صورت بلوک تابعی استاندارد که در راهنمای VDI/VDE ۳۶۹۶ تعریف شد، شرح داده می شود. یک مقدار سنجش آنالوگ و متغیر کنترل شده توسط یک بلوک تابعی از نوع IN-A از کانال ورودی به آدرس INADR تأمین می شود و در بازه XMIN تا XMAX به یک مقدار فیزیکی با واحد XUNIT مقیاس می شود. متغیر کنترل شده به یک بلوک تابعی از نوع C داده می‌شود که کارش تنظیم دیفرانسیلی – انتگرالی – نسبی متأثر از پارامترهای کنترل IN، KP و TV است. متغیر تنظیم کننده حاصل به وسیله یک بلوک تابعی خروجی از نوع OUT-A به یک مقدارآنالوگ تبدیل می شود و به کانالی که به وسیله OUTADR مشخص شده هدایت می شود. علاوه براین، متغیر کنترل شده، همیشه تحت نظارت دو نمونه از نوع بلوک تابعی استاندارد SAM است تا در محدوده مشخص شده با پارامترهای LS و HS باقی بماند. چنانچه متغیر کنترل شده خارج محدوده قرار گیرد یکی از خروجی های QS دو نمونه SAM مقدار True می گیرد و به همین ترتیب خروجی از نوع بلوک تابعی OR نیز True می شود. این به نوبه خود موجب می شود اخطار نوع AM و بلوک تابعی ذخیره پیام، یک رکورد اخطار زمان بندی شده ایجاد کنند. ورودیهای بلوک تابعی استاندارد با استفاده از برنامه ای که نه از طریق ورودیهای قابل مشاهده خارجی خود نرم افزار و نه به صورت داخلی به وسیله خروجی های بلوک های تابعی استاندارد دیگر تغذیه نشده اند، مقادیر ثابت می گیرند. فرم نمایش لیست شبکه ای برنامه مثال فوق که با برنامه OrCAD تولید شده است در شکل ۸ نشان داده شده است. نمودارهای شبکه، نمایش‌های متنی را تشکیل می دهند که کاملاً معادل طراحی های اصلی هستند (به جز برای جنبه‌های هندسی)

<<< Component List >>>

IN_A	B1
C	B2
OUT_A	B3
SAM	B4
SAM	B5
OR	B6
AM	B7

<<< Wire List >>>

NODE	REF	PIN #	PIN NAME	PIN TYPE	PART TYPE
[00001] N00001	B1	5	X	Output	IN_A
	B2	1	X	Input	C
	B4	1	X	Input	SAM
	B5	1	X	Input	SAM
[00002] N00002	B2	5	Y	Output	C
	B3	1	Y	Input	OUT_A
[00003] N00003	B4	4	QS	Output	SAM
	B6	1	I1	Input	OR
[00004] N00004	B6	3	Q	Output	OR
	B7	1	I	Input	AM
[00005] N00005	B5	4	QS	Output	SAM
	B6	2	I2	Input	OR
[00006] XMIN	B1	1	XMIN	Input	IN_A

شکل ۸ - فرم نمایش شبکه ای یک برنامه نمونه

کد هدفی که در نهایت برای مفسر به وسیله ترجمه اتوماتیک از فرم نمایش شبکه ای یک برنامه نمونه بدست می آید در شکل ۹ لیست شده است و نسخه زبان اسمبلی قابل خواندن را نشان می دهد. انواع بلاک های عمل ذکر شده در مثال SAM و C و AM دارای متغیرهای حالت درونی هستند (به عنوان مثال C دارای سه متغیر و دو مورد دیگر دارای یک متغیر حالت درونی هستند)

GET ROM-loc-ID-IN_A	GET ROM-loc-ID-OUT_a	GET ROM-loc-ID-OR
GET RAM-loc-XMIN	GET RAM-loc-TMP-Y	GET RAM-loc-TMP-H
GET RAM-loc-XMAX	GET RAM-loc-OUTADR	GET RAM-loc-TMP-L
GET ROM-loc-BAR	PUT RAM-loc-TMP-OR	
GET RAM-loc-INADR	GET ROM-loc-ID-SAM	
PUT RAM-loc-TMP-X	GET RAM-loc-TMP-X	GET ROM-loc-ID-AM
GET ROM-loc-0	GET RAM-loc-TMP-OR	
GET ROM-loc-ID-C	GET RAM-loc-HS	GET ROM-loc-1
GET RAM-loc-TMP-X	GET RAM-loc-B4-isv	GET ROM-loc-A1
GET RAM-loc-KP	PUT RAM-loc-TMP-H	GET ROM-loc-14
GET ROM-loc-2.0	PUT RAM-loc-B4-isv	GET RAM-loc-B7-isv
GET ROM-loc-0.0	PUT RAM-loc-B7-isv	
GET RAM-loc-B2-isv1	GET ROM-loc-ID-SAM	
GET RAM-loc-B2-isv2	GET RAM-loc-TMP-X	STEP
GET RAM-loc-B2-isv3	GET ROM-loc-1	
PUT RAM-loc-TMP-Y	GET RAM-loc-LS	
PUT RAM-loc-B2-isv1	GET RAM-loc-B5-isv	
PUT RAM-loc-B2-isv2	PUT RAM-loc-TMP-L	
PUT RAM-loc-B2-isv3	PUT RAM-loc-B5-isv	

شکل ۹- نمایش کد هدف برنامه مثال کد هدف نمایش داده شده

شکل ۹ بیانگر این است که فراخوانی همه نمونه های بلوک تابعی که در برنامه رخ می دهند، به طور مستقیم به رویه هایی متناظر می گردند. هر یک از آنها با یک دستور GET آغاز می شوند که مشخصه بلاک مربوط را از خارج از محل ROM مناسب به مفسر انتقال می‌دهد. سپس پارامترهای ورودی با خواندن خانه‌های ROM (برای ثابت ها) یا RAM (برای پارامترهای برنامه و مقادیر بلافصل) مناسب فراهم می شوند. نهایتاً چنانچه ورودی موجود باشد مقادیر متغیرهای وضعیت داخلی رویه از روی RAM خوانده می شود. برای هر نمونه بلوک تابعی با وضعیت داخلی یک مجموعه مکانهای برچسب دار متناظری (مثلاً RAM-LOC-B2-isv) وجود دارد. وقتی مفسر همه داده ها را دریافت کند، رویه را اجرا کرده و در صورت وجود نتیجه، مقادیر پارامترهای خروجی و / یا متغیرهای وضعیت داخلی که در محل RAM مربوطه ذخیره شده اند را باز می گرداند. رابطـه بین خـروجی یک بلوک تابعی و ورودی بلوک دیـگر با دستور PUT و GET پیاده سازی می شود. دستورPUT مقدار خروجی را در مکانی از RAM به عنوان یک مقدار موقتی مثل TMP-X ذخیره می کند و دستور GET آن را از مکان مورد نظر می خواند. به بیان دیگر هر گره (Node) در نمودار شبکه ای باعث دقیقاً یک انتقال از مفسر به یک سلول RAM و یک انتقال یا بیشتر از آنجا به مفسر است. جزئیات پیاده سازی رویه های گوناگون بخشی از Firmware معماری است و به همین دلیل غیرقابل مشاهده می ماند. با توجه به ساختار توصیف شده فوق در خصوص برنامه های هدف مفسر، پروسه ترجمه و جداسازی کد هدف بسیار ساده است. برای انجام ترجمه، ابتدا دستورات STEP جستجو می شوند که به طور کاملاً واضح گامهای متوالی مختلف موجود در برنامه را از هم جدا می کند. کد بین دو دستور STEP به یک نمودار بلوک تابعی مربوط می شود. سپس اولین دستور GET تفسیر می گردد. بدین وسیله یک بلوک تابعی جهت ترسیم در دیاگرام بلوک تابعی مشخص می شود. با مقایسه GET های بعدی

با توصیف بلوک تابعی درون کتابخانه استفاده شده ، صحت ارسال پارامتر به سادگی قابل تایید است. علاوه بر این برای هر GET که با یک پارامتر مناسب متناظر است (ونه با یک متغیر وضعیت داخلی) یک خط انتقال در نمودار کشیده می شود. دونوع اتصال وجود دارد. در نوع اول ارتباطاتی از ثابت‌ها یا ورودی های برنامه به ورودیهای بلوک تابعی یا از خروجی های بلوک تابعی به خروجی های برنامه وجود دارد. در نوع دوم ارتباطات یکطرفه ای از خروجی های بلوک تابعی به نقاط اتصال نامگذاری شده در نمودار یعنی گره های لیست شبکه ای یا از چنین نقاطی به ورودیهای عمل وجود دارد. زمانیکه نمودار به طور کامل کشیده شود، نام این نقاط قابل حذف است. با توجه به متغیرهای وضعیت داخلی باید مشخص شود که مکانهای RAM متناظر به درستی مقدار دهی شده اند و مقادیر جدید حاصل از اجرای بلوک تابعی درست در همان مکانهایی نوشته شده اند که وضعیت های داخلی در همین فراخوانی بلاک خوانده شده بودند. پروسه تشخیص بلوک تابعی و تأیید پارامترهای پاس شده دقیقاً همراه رسم نماد برای بلوک و اتصالات مربوط به آن آنقدر تکرار می شود تا یک دستور STEP که این گام و نمودار بلوک تابعی معادل را خاتمه می دهد فرا برسد.

نتایج و تجربیات

براساس معماری فوق یک نمونه اولیه از کنترل گر کامپیوتری و یک مفسر بلوک تابعی که اعمال نقض بولی، ترکیب و تجزیه را به همراه تأخیرهای زمانی انجام می دهد ساخته ایم. در حالیکه سه عمل اول جزئی هستند، زمان سنج تأخیری نیازمند یک تأیید رسمی صحت است که به وسیله استخدام HOL انجام شده و مشخص گردید که این عمل خیلی پرکار و طولانی است. بهره برداری از کنترل گر در عمل نشان داد که پیاده سازی سیستم با قابلیت خاموش شدن در حالت اضطراری به شیوه سیم بندی شده به همراه سیستم الکترونیکی برنامه‌ریزی شده ممکن است و الگوی برنامه نویسی مبتنی بر ماژول های عمل تأیید شده موجود در برنامه های کاربردی تأیید شده توسط ترجمه های متعدد می تواند باعث ظهور نرم افزاری عاری از هر خطا شود. این مورد اخیر، به همراه سکوهای سخت افزاری با قابلیت چشم پوشی از خطا امکان سیستمهای محافظ قابل برنامه ریزی را فراهم می کند که این سیستمها خصلت بدون خطا بودن خودشان را در شبکه ای از سیستمهای سیم‌بندی شده، به اشتراک می گذارند. به دلایل ارگانیکی نرم افزار پیچیده بهتراست به شیوه سلسله مراتبی و به طریقی طراحی شود که فرمولاسیون ماژول های آن در قالب نمودار بلوک تابعی همیشه در یک صفحه خروجی ترمینال (مانیتور) جا شود. همانطور که قبلاً تشریح شد این ماژول ها به صورت بلوک تابعی در سطح بالاتر بعدی از ساختار سلسله مراتبی که اجزای داخلی آن به صورت مجرد بوده و توسط مفسر مذکور اجرا شده اند ظاهر می

گردد. از این رو پیچیدگی بلوک تابعی هرگز از حدود ده برابر مثال بخش ۸ بیشتر نخواهد بود که این خود باعث می‌شود پروسه تأیید به حداکثر چند ساعت کار ختم گردد.(برای هرگونه نمودار بلوک تابعی خاص)

« نتیجه گیری» توجه به امنیت در جامعه رشد فزاینده ای یافته است و به تبع آن شاهد افزایش آگاهی محیطی نیز هستیم. این رخداد تبعات مهمی در ارزیابی سیستمهای کنترل شده توسط کامپیوتر داشته است. ما به مرور متوجه مسائل امنیتی ذاتی نرم افزار شدیم. عدم استفاده از کامپیوترها برای اهداف اتوماسیون مرتبط با ایمنی نه تنها غیرمنطقی است بلکه برعکس بنابه دلایل ذکر شده تردیدی وجود ندارد که بهره برداری از چنین برنامه های کاربردی رو به افزایش است و این خود باعث سخت تر شدن قابلیت اعتماد به نرم افزار می شود. در چنین شرایطی این مقاله یک مسأله آزار دهنده را مورد بررسی قرار داد. در این مقاله راه حلهایی برای همه سئوال های موجود در مبحث مجوز استفاده(License) ایمن یک نرم افزار ارائه نشده است. اما مقدمه ای برآن ذکر شده که کاملاً شدنی بوده و به گستره وسیعی از مشکلات کنترلی مرتبط با ایمنی،قابل اعمال است. ازاین رو انتظار می رود که مفاهیم ارائه شده در اینجا، نهایتاً باعث شود منطق رله یا گسسته جایگزین سیستمهای الکترونیکی قابل برنامه‌ریزی که اجرا کننده نرم افزار تأیید شده ایمنی و مسئول اعمال بحرانی در اتوماسیون هستند گردد. همزمان با پاسخگویی به درخواست جامعه برای سیستمهای کامپیوتری قابل اعتماد بیشتر، تحت محدودیت های اقتصاد پایدار، مفاهیم ارائه شده می تواند بلافاصله به پیاده‌سازی صنعتی راه پیدا کند. روش ما توجه ویژه ای به شرایط اقتصادی دارد. این روش گامی به سمت الگوی مهندسی نرم افزار است که عامل به وجود آمدن برنامه نویسی ایمن وساده می باشد. مشخصه این روش، سیستم طراحی مجتمع است که نه تنها در آن نشانی از پیچیدگیهای ریاضی نیست بلکه مفاهیم آن قابل فهم برای افراد متخصص نیز هست. این روش می تواند جایگزین روش تجربی بررسی مجوز استفاده از نرم افزار شود. تایید نرم افزار وقتی به عنوان یک پروسه اجتماعی رسیدن به توافق در نظر گرفته شود می تواند توسط قابلیتهایی مانند برنامه نویسی گرافیکی، استفاده مجدد از اجزا تایید شده قبلی، برنامه نویسی در سطح مشخصات(Specification level programming) و به وسیله پر کردن جدول تصمیم گیری تسهیل شود و تقریبا در همه جنبه ها حداکثر سادگی را ارائه دهد. سیستمهای کنترل مبنی بر نرم افزار هنوز هم از نظر قابلیت اعتماد پایین تر از اجزای سیم بندی شده معمولی محسوب می گردند. قدمت مهندسی نرم افزار با سالها تجربه در امر توسعه استراتژی ها می تواند این برتری را جهت غلبه بر نقایص توجیه کند. پیشنهاد روشهای برنامه نویسی اختصاصی را برای مدیریت این وضعیت مطرح کرده ایم که از قواعد و اصولی پیروی می کند که هر چه سیستمی به لحاظ امنیتی بحرانی تر باشد نرم افزار کنترل مربوط به آن ساده تر می شود. بنابراین فرد قادر است در بالاترین سطح ایمنی تنها با نگاه کردن به جدول نشانه-دلیل،

امنیت نرم افزار را تایید کند(بدون نیاز به استفاده از اثباتهای رسمی) برای کاهش نیازمندیها می توان نرم افزار کنترل را بر مبنای بلاک عملهای اثبات شده قبلی به صورت گرافیکی بنا کرد که این امر باعث تسهیل در روند توسعه نرم افزار در مقایسه با زبانهای منتی شده و اثباتهای امنیتی را در سطح قابل قبولی محکم و پایدار و در عین حال ساده و غیر رسمی نگاه می دارد.