طراحی سیستم‌های تحمل پذیر خطا

ترجمه:

مهندس سیامک وطنی

فن‌آوری نوین

سرشناسه	:	دوبروا، النا ولادیمیروونا، ۱۹۶۸ - م.Dubrova، Elena Vladimirovna
عنوان و نام پدیدآور	:	طراحی سیستم‌های تحمل‌پذیر خطا/[مولف النا ولادیمیروونا دوبروا] ؛ ترجمه سیامک وطنی.
مشخصات نشر	:	بابل: فن‌آوری نوین، ۱۳۹۷.
مشخصات ظاهری	:	۱۷۲ص.: مصور،جدول
شابک	:	۲۲۰۰۰۰ ریال: 978-600-7272-18-3
وضعیت فهرست‌نویسی	:	فیپا
یاداشت	:	عنوان اصلی[Fault-tolerant design,[2013 :
موضوع		عیب‌تابی (مهندسی)
موضوع		Fault tolerance (Engineering)
شناسه افزوده		وطنی، سیامک، ۱۳۴۹ -، مترجم
رده بندی کنگره	:	TA۱۳۹۷۱۶۹ ۴ط۹د/
رده بندی دیویی	:	۰۰۴۵۲/۶۲۰
شماره کتابشناسی ملی	:	۵۱۸۲۸۱۸

فن‌آوری نوین **بابل، صندوق پستی ۴۷۱۶۷-۷۳۴۴۸** **تلفن: ۳۲۲۵۶۶۸۷-۰۱۱**

www.fanavarienovin.net

طراحی سیستم‌های تحمل پذیر خطا

ترجمه: مهندس سیامک وطنی

ناشر: فن‌آوری نوین

چاپ اول: بهار ۱۳۹۷

جلد : ۱۰۰۰

شابک: ۳ – ۱۸ – ۷۲۷۲ – ۶۰۰ – ۹۷۸

حروفچینی و صفحه‌آرایی: فن‌آوری نوین **قیمت:** ۲۲۰۰۰تومان

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲ تلفکس: ۶۶۴۰۰۱۴۴-۶۶۴۰۰۲۲۰

مقدمه

این کتاب درسی به‌عنوان مقدمه‌ای برای تحمل خطا[1] به کار می‌رود، و برای دانشجویان کارشناسی، دانشجویان تحصیلات تکمیلی، و مهندسینی که نیاز به یک مرور کلی در این زمینه دارند، مناسب می‌باشد. خوانندگان با کمک واژگان قابلیت اطمینان[2]، در دسترس بودن[3]، و ایمنی[4] مهارت‌هایی در مدل‌سازی و ارزیابی معماری سیستم‌های تحمل‌پذیر خطا کسب خواهند کرد. آن‌ها درک کاملی از محاسبات قابل‌تحمل خطا ازنظر تئوری و عملی کسب خواهند کرد. درزمینه تئوری می‌توان به نحوه دستیابی چنین سیستمی از طریق افزونگی[5] سخت‌افزاری، نرم‌افزاری اطلاعات و زمان و درزمینه عملی به دانش طراحی سیستم‌های نرم‌افزاری و سخت‌افزاری باقابلیت تحمل خطا اشاره کرد.

این کتاب شامل ۸ فصل بوده و مباحث ذیل را در برمی‌گیرد. فصل ۱ مقدمه بوده و اهمیت تحمل‌پذیر بودن خطا در توسعه یک سیستم قابل‌اطمینان را شرح می‌دهد. فصل ۲ سه ویژگی اساسی قابلیت اطمینان نظیر: خصلت‌ها[6]، عوامل اختلال[7] و روش‌ها را شرح می‌دهد. فصل ۳ روش‌های ارزیابی قابلیت اطمینان، در دسترس بودن و مدل‌های قابلیت اطمینان نظیر بلوک دیاگرام و یا زنجیرهای مارکف[8] را شرح می‌دهد. فصل ۴ روش‌های مورداستفاده برای طراحی سیستم‌های سخت‌افزاری تحمل‌پذیر نظیر افزونگی ماژولار سه‌تایی[9]، افزونگی آماده‌به‌کار[10] و افزونگی خود پالاینده[11] را شرح داده و تأثیر آن‌ها را بر قابلیت اطمینان سیستم ارزیابی می‌کند. فصل ۵ نحوه استفاده از کدینگ برای حصول تحمل خطا را نشان می‌دهد. در این فصل خانواده مهمی از کدها نظیر: توازن، خطی، چرخشی، بدون ترتیب و کدهای ریاضی مورد بررسی قرار خواهند گرفت. فصل ۶ افزونگی زمانی برای آشکار کردن خطاهای گذرا و دائم را شرح می‌دهد. فصل ۷ روش‌های اصلی برای طراحی سیستم‌های نرم‌افزاری تحمل‌پذیر خطا نظیر: نقاط چک[12]، شروع مجدد[13]، بلوک‌های بازیابی، برنامه‌نویسی N نسخه‌ای و برنامه‌نویسی N خودآزمون را شرح می‌دهد. فصل ۸ نتیجه‌گیری کتاب محسوب می‌شود.

محتویات کتاب به‌گونه‌ای طراحی‌شده است تا به‌راحتی قابل‌دسترس باشند، برای مثال می‌توان به مثال‌های متعدد و مسائلی که برای تقویت مطالب آموخته‌شده به کار می‌روند اشاره کرد. جواب مسائل و اسلایدهای پاورپوینت را می‌توان از نویسنده درخواست و تهیه نمود.

النا دوبروا استکهلم، سوئد، دسامبر ۲۰۱۲

---

[1] fault tolerance [2] reliability [3] availability [4] safety [5] redundancy [6] attributes
[7] impairment [8] Markov chains [9] triple modular [10] standby [11] self-purging
[12] checkpoint [13] restart

قدردانی‌ها

این کتاب بر اساس یادداشت‌های من برای درس "طراحی سیستم‌های امن" که در موسسه فنی KTH از سال ۲۰۰۰ تدریس می‌کرده‌ام شکل گرفته است. در طول سال‌ها، بسیاری از دانش آموزان و همکارانم کمک کرده‌اند تا متن را بهبود بخشم. از همه‌کسانی که به من بازخورد دادند سپاسگزارم. مراتب قدردانی خود را از نان لی، شهره شریف منصوری، نسیم فراهینی، بائورون نواس، جانیان گرشتانی، خاویر، واجی، پیتر نویتز، هنریک کیرکیبی، چن فو، کاریم رفات، سرگ کوزینر، جولیا کوزنتسوا و روم موروا‌ک برای پیدا کردن اشتباهات متعدد و پیشنهادهای ارزشمند در دست‌نوشته اولیه اعلام می‌دارم. از هانو تنهونن که الهام‌بخش من برای تدریس این درس بود و همواره در طول فعالیت دانشگاهی از من حمایت کرده تشکر می‌کنم. من به شارل گلاسر از انتشارات اسپرینگر برای تشویق و مساعدت وی در نشر کتاب و ساندرا برونزبرگ برای ویرایش پیش‌نویس نهایی مدیونم. از دوست عزیزی که پیشنهاد استفاده از نرم‌افزار MetaPost را داده متشکرم و نتایج خوب حاصله مدیون این پیشنهاد است. در خاتمه از بنیاد سوئد برای همکاری بین‌المللی، تحقیق و آموزش عالی (STINT) برای اعطای بورس تحصیلی ۲۰۰۲KU-۴۰۴۴ که امکان مسافرت من به دانشگاه نیو سات ولز، سیدنی، استرالیا را فراهم آورد، تشکر می‌کنم. اولین پیش نویس این کتاب در اکتبر-دسامبر ۲۰۰۲ در این دانشگاه به رشته تحریر درآمد.

فهرست مطالب

فصل ۱ مقدمه

هر خطای ممکن رخ خواهد داد. قانون مورفی

در این فصل، به‌طور رسمی تحمل خطا را تعریف می‌کنیم و در مورد اهمیت آن برای طراحی یک سیستم قابل‌اعتماد بحث می‌کنیم. رابطه بین تحمل خطا و افزونگی را نشان داده و انواع افزونگی را بررسی خواهیم کرد. تاریخچه محاسبات قابل‌تحمل خطا را به‌طور مختصر بررسی کرده و زمینه‌های کاربرد اصلی آن را توضیح می‌دهیم.

۱–۱) تعریف تحمل خطا

تحمل خطا، توانایی یک سیستم برای انجام وظایف موردنظر باوجود خرابی است [۷]. به معنای وسیع‌تر، تحمل خطا باقابلیت اطمینان، با عملیات موفقیت‌آمیز و با عدم وجود اختلالات ارتباط نزدیکی دارد. یک سیستم تحمل‌کننده خطا باید قادر به مدیریت خطاها در قطعات سخت‌افزاری، مؤلفه‌های نرم‌افزاری، خرابی‌های منبع تغذیه و انواع دیگر مشکلات غیرمنتظره باشد و مطابق با مشخصات تعیین شده رفتار کند.

تحمل خطا ضروری است، چراکه امکان ساخت سیستم بدون عیب در عمل میسر نیست. مشکل اساسی آن است که با افزایش پیچیدگی یک سیستم، قابلیت اطمینان آن به‌طور قابل‌توجهی کاهش می‌یابد مگر آن‌که اقدامات جبرانی انجام گیرد. برای مثال، اگر قابلیت اطمینان یک قطعه ۹۹٫۹۹٪ باشد، قابلیت اطمینان یک سیستم متشکل از ۱۰۰ قطعه غیر افزونه ۹۹٫۰۱٪ است، درحالی‌که قابلیت اطمینان سیستمی متشکل از ۱۰،۰۰۰ قطعه غیر افزونه فقط ۳۶٫۷۹٪ است و چنین قابلیت اطمینان پایینی در اکثر کاربردها قابل قبول نیست. اگر قابلیت اطمینان ۹۹٪ برای یک سیستم متشکل از ۱۰،۰۰۰ قطعه مدنظر باشد، باید از قطعه‌هایی باقابلیت اطمینان حداقل ۹۹٫۹۹۹٪ استفاده شود که این موضوع موجب افزایش شدید هزینه می‌گردد.

یکی دیگر از مشکلات آن است که اگرچه طراحان بهترین تلاش را می‌کنند تا تمام ایراد های سخت‌افزاری و یا اشکالات نرم‌افزاری را از سیستم حذف کنند، اما تاریخ نشان می‌دهد که چنین هدف تقریباً قابل‌دستیابی نیست [۳]. برخی عوامل محیطی غیرمنتظره در نظر گرفته نمی‌شود و یا برخی از اشتباهات بالقوه کاربر پیش‌بینی نمی شود، بنابراین، حتی در مواردی نادر که یک سیستم کامل و بی نقص طراحی و اجرا گردد، خطاهای احتمالی از شرایطی خارج از کنترل طراحان ناشی می‌شوند.

زمانی که یک سیستم قادر به انجام وظایف تعیین شده نباشد، دچار خرابی شده است. سیستم در این کتاب به مفهوم عام یک گروه از عناصر مستقل اما مرتبط است که تشکیل یک جز واحد را می‌دهد؛

بنابراین، تکنیک‌های ارائه شده برای انواع محصولات، دستگاه‌ها و زیرسیستم‌ها قابل‌اجرا هستند. خرابی می‌تواند توقف کامل وظایف محوله و یا کاهش کارایی بعضی از وظایف از منظر کیفی و یا کمی نظیر نادرست بودن و یا بی‌ثباتی عملیات باشد. هدف از طراحی سیستم امن کاهش احتمال خرابی است، این خرابی‌ها می‌توانند به‌سادگی موجب آزار کاربران، زیان مالی، صدمات جانی و یا فاجعه زیست‌محیطی شوند.

۱–۲) تحمل خطا و افزونگی

راه‌های مختلفی برای دست‌یابی به سیستم‌های امن وجود دارد. فصل اشتراک تمامی این روش‌ها، وجود قدری افزونگی است. برای هدف ما، افزونگی فراهم کردن قابلیت عملکردی است که در یک محیط بدون خطا غیرضروری است [۸]. با افزودن یک قطعه سخت‌افزاری تکراری، چند خط کد برنامه که صحت نتایج برنامه را تأیید می‌کند و یا یک بیت اضافی متصل به یک‌رشته از داده‌های دیجیتال می‌توان به این هدف نائل شد. ایده تلفیق افزونگی به‌منظور بهبود قابلیت اطمینان سیستم، به یکی از کارهای جان فون نویمان در دهه ۱۹۵۰ در اثری به نام "منطق احتمالاتی و سنتز موجودات قابل‌اعتماد از اجزای غیرقابل‌اعتماد" برمی‌گردد [۱۲].

دو نوع افزونگی وجود دارد [۱]: افزونگی فضایی و افزونگی زمانی. افزونگی فضایی قطعات اضافی، توابع و یا اقلام داده‌ای را فراهم می‌کند، این افزوده‌ها برای عملیات بدون خطا ضروری نیستند. افزونگی فضایی بسته به منابع اضافه‌شده به سیستم، به افزونگی سخت‌افزاری، نرم‌افزاری و اطلاعاتی تقسیم می‌شود. در افزونگی زمانی، محاسبات و یا انتقال اطلاعات تکرار می‌شوند و نتیجه حاصله با نتایج از قبل ذخیره‌شده مقایسه می‌شوند.

۱–۳) کاربرد سیستم‌های امن

در ابتدا، تکنیک‌های تحمل خطا برای مقابله با ایرادهای فیزیکی در قطعات سخت‌افزاری موردا ستفاده قرار گرفت. طراحان سیستم‌های محاسباتی اولیه عناصر اساسی را سه برابر کرده و با استفاده از رأی اکثریت خطاها را می‌پوشاندند [۱۱]. پس از جنگ جهانی دوم، علاقه به سیستم‌های امن به‌طور قابل‌توجهی افزایش یافت. یکی از دلایل این موضوع بروز مشکلات اطمینانی در تجهیزات الکترونیکی در طول جنگ بود [۲]. روشن شد که فناوری نظامی آینده به‌شدت به الکترونیک پیچیده متکی است. یکی دیگر از عوامل فزاینده مسابقه فضایی بین ایالات‌متحده و اتحاد جماهیر شوروی بود که پس از راه‌اندازی اولین ماهواره به نام

اسپوتنیک[1] در سال ۱۹۵۷ آغاز شد [۲]. درنتیجه، بخش‌های دفاع و سازمان‌های فضایی در بسیاری از کشورها پروژه‌های مربوط به تحمل خطا را آغاز کرده‌اند.

همان‌طور که فنّاوری نیمه‌هادی پیشرفت کرد، اجزای سخت‌افزاری به‌طور ذاتی قابل‌اعتمادتر شدند و نیاز به تحمل قطعاتی باتحمل خطا در برنامه‌های کاربردی عمومی کاهش یافت. بااین‌وجود، تحمل خطا همچنان یک ویژگی ضروری برای سیستم‌های مورداستفاده در برنامه‌های بحرانی از منظر امنیتی، مأموریتی و تجاری محسوب می‌شوند. برنامه‌های امنیتی بحرانی، برنامه‌هایی هستند که عملکرد نادرست آن‌ها می‌تواند موجب خطرات جانی و یا فاجعه زیست‌محیطی گردد و باید از بروز آن جلوگیری کرد. سیستم‌های کنترل نیروگاه هسته‌ای که توسط کامپیوتر کنترل می‌شوند، دستگاه‌های پرتودرمانی، دستگاه‌های ضربان قلب و سیستم‌های کنترل هوایی مثال‌هایی ازاین‌دست محسوب می‌شوند. برنامه‌های مأموریت بحرانی، به تکمیل وظیفه محوله در یک فضاپیما یا یک ماهواره تأکید می‌کنند. برنامه‌های بحرانی تجاری، برنامه‌های هستند که برای نگهداری یک کسب‌وکار به‌طور مداوم، حائز اهمیت هستند. سیستم‌های تجاری بانک، بورس اوراق بهادار، سرورهای وب و تجارت الکترونیک مثال‌هایی ازاین‌دست محسوب می‌شوند.

در اواسط دهه ۱۹۹۰ علاقه‌مندی به سیستم‌های تحمل خطا به‌طور قابل‌توجهی کاهش یافت. از سویی دیگر با کاهش مداوم ابعاد ساخت مدار، کاهش ولتاژ منبع تغذیه و افزایش سرعت عملیات، میزان حاشیه نویز[2] به یک سطح بحرانی رسید. این عوامل موجب شد مدارهای مجتمع نسبت به تداخل و آسیب‌های بیرونی نظیر نوترون‌های اتمسفری و ذرات آلفا حساس شوند [۱۰، ۱۴]. برای حفظ سطح قابل قبولی از قابلیت اطمینان، طراحی مدارهای مجتمع باقابلیت تحمل خطا به یک اجبار بدل شد [۶].

از سویی دیگر، توسعه سریع برنامه‌های محاسباتی بلادرنگ[3] که در اواسط ۱۹۹۰ آغاز شدند و تقاضا برای ابزارهای هوشمند توکار[4] به همراه نرم‌افزارهای مربوطه، مبحث تحمل خطا در نرم‌افزار را به یک موضوع مهم بدل کردند [۹]. سیستم‌های نرم‌افزاری یک طراحی جمع‌وجور همراه با عملکردی عالی و قیمتی رقابتی ارائه می‌کنند. به‌جای پیاده‌سازی عملکرد موردنظر به‌صورت سخت‌افزاری، کدهای مربوطه نوشته‌شده و سپس در پردازنده بارگذاری می‌شود. درصورتی‌که نیاز به تغییر عملکرد سیستم باشد، به‌جای ساخت یک ابزار فیزیکی جدید فقط می‌بایست دستورالعمل‌ها تغییر کنند.

---

[1] Sputnik
[2] noise margin
[3] real-time
[4] embedded intelligent device

نرم‌افزار بسیاری از محدودیت‌های فیزیکی سخت‌افزار را حذف می‌کند. برای مثال، نرم‌افزار از ایراد ساخت تصادفی رنج نمی‌برد و دچار فرسودگی نمی‌شود. بااین‌حال، یک مشکل مرتبط اجتناب‌ناپذیر آن است که طراحی یک سیستم توسط فردی که متخصص در آن سیستم نیست، انجام می‌گیرد. به‌عنوان‌مثال، کارشناس ترمز خودرو تصمیم می‌گیرد که ترمزها چگونه باید کار کنند و سپس این اطلاعات را در اختیار یک مهندس نرم‌افزار که وظیفه نوشتن برنامه موردنظر را بر عهده دارد، قرار می‌دهد. امروزه این ارتباط مضاعف بین مهندس و توسعه‌دهنده نرم‌افزار منبع بسیاری از خطاها در نرم‌افزار محسوب می‌گردد [۳].

در سال‌های اخیر با تغییر الگوی محاسبات رومیزی که یک کاربر از یک سیستم برای منظوری خاصی استفاده می‌کند به الگوی محاسباتی کوچک و متنوع تحت شبکه که در تمامی قسمت‌های زندگی رسوخ کرده‌اند، موجب افزایش علاقه‌مندی و توجه به سیستم‌های امن را فراهم کرده است [۱۳]. صنعت پیش‌بینی می‌کند تا سال ۲۰۲۰ بیش از ۵۰ میلیارد نفر و دستگاه‌ها به پهنای باند تلفن همراه متصل شوند [۵]. همان‌طوری که جامعه ما تبدیل به جامعه شبکه‌ای می شود، ضمانت قابلیت اطمینان سرویس‌های شبکه و اجزای آن به‌طور فزاینده‌ای اهمیت می‌یابد. این احتمال وجود دارد که رویکردی جدید و غیر سنتی برای دستیابی به تحمل خطا موردنیاز خواهد شد [۴].

فصل ۲ مبانی قابلیت اطمینان

استفاده‌های عجیبی از کلمه ایمن وجود دارد که قبلاً اطلاعی نداشتم.
داگلاس آدامز، راهنمای مسافران مجانی کهکشان[1]

هدف نهایی تحمل خطا، توسعه یک سیستم قابل‌اعتماد است. به‌طورکلی، قابلیت اطمینان[2] توانایی یک سیستم برای ارائه سطح خدمات موردنظر به کاربران می‌باشد [۱۶]. همین‌طور که محاسبات فراگیر می‌شوند و در تمام زندگی روزمره ما نفوذ می‌کنند، قابلیت اطمینان نه‌تنها برای برنامه‌های کاربردی سنتی، ایمنی، مأموریتی و بحرانی مهم است، بلکه برای کل جامعه نیز اهمیت دارد. در این فصل سه ویژگی اساسی قابلیت اطمینان نظیر: خصلت‌ها، عوامل اختلال و روش‌ها را شرح می‌دهیم. خصلت‌ها ویژگی‌های قابلیت اطمینان موردنظر یک سیستم را توصیف می‌کنند. اختلالات دلایل عدم کارکرد صحیح سیستم و عوامل تهدیدکننده قابلیت اطمینان را شرح می‌دهند. روش‌ها، متدها و تکنیک‌هایی مانند پیشگیری از خطا، تحمل خطا، حذف خطا و پیش‌بینی خطا هستند که امکان توسعه یک سیستم قابل‌اعتماد را فراهم می‌کنند.

۲–۱) نمادها

در این کتاب از علائم ., $\oplus$ و $\bar{x}$ به ترتیب برای نمایش عملگرهای AND و XOR و مکمل استفاده می‌کنیم. نماد × معرف ضرب ریاضی می‌باشد. از نماد + برای عملیات جمع ریاضی و OR منطقی استفاده می‌کنیم. مفهوم این نماد با توجه به متن قابل استنتاج است.

۲–۲) خصلت‌های اطمینان‌پذیری

خصلت‌ها بیانگر خواصی است که از یک سیستم انتظار می‌رود. سه ویژگی اصلی، قابلیت اطمینان، قابلیت دسترسی و ایمنی هستند. سایر ویژگی‌های احتمالی عبارت‌اند از: قابلیت نگه‌داری، آزمون‌پذیری، قابلیت اجرا و امنیت [۲۱]. بسته به کاربرد، ممکن است یک یا چند ویژگی برای ارزیابی مناسب رفتار سیستم موردنیاز باشند. به‌عنوان‌مثال، در یک ماشین خودپرداز (ATM)، نسبت زمانی که سیستم قادر به ارائه سطح خدمات موردنظر خود (دسترسی به سیستم) می‌باشد، اهمیت دارد. برای یک بیمار قلبی با یک ضربان ساز، عملکرد مداوم دستگاه، موضوع زندگی و مرگ است؛ بنابراین توانایی سیستم برای ارائه خدمات بدون وقفه (قابلیت اطمینان سیستم) بسیار مهم است. در سیستم کنترل نیروگاه هسته‌ای، توانایی

[1] The Hitchhikers Guide to the Galaxy
[2] Dependability

سیستم برای انجام صحیح عملکرد آن و یا توقف عملکرد آن به‌طور ایمن (ایمنی سیستم) از اهمیت بالایی برخوردار است.

۲-۲-۱) اطمینان‌پذیری

با فرض آن‌که سیستم در زمان صفر درست عمل کند، قابلیت اطمینان R(t) در زمان t، احتمال عملکرد صحیح سیستم در بازه زمانی [0,t] می‌باشد. قابلیت اطمینان اندازه‌گیری ارائه مستمر خدمات صحیح تلقی می‌گردد. قابلیت اطمینان بسیار بالا در شرایطی است که انتظار می‌رود سیستم بدون وقفه کار کند و یا زمانی که تعمیر و نگه‌داری به علت عدم دسترسی به آن میسر نیست. برای مثال یک ضربان ساز قلب و اکتشافات فضایی نمونه‌هایی ازاین‌دست تلقی می‌شوند. به‌عنوان مثالی دیگر، از یک سیستم کنترل مأموریت فضاپیما انتظار می‌رود که خدمات را بدون وقفه ارائه دهد. ایراد در این سیستم احتمالاً موجب تخریب فضاپیما می‌شود، همان‌طور که در مورد فضاپیمای لوئیس[1] ناسا که در ۲۳ آگوست ۱۹۹۷ راه‌اندازی شد، اتفاق افتاد [۲۰]. این فضاپیما وارد مداری شد که به انرژی خورشید دسترسی نداشت و درنهایت منجر به خالی شدن باتری شد. ارتباط با سفینه قطع شد و این فضاپیما در تاریخ ۲۸ سپتامبر ۱۹۹۷ نابود شد. بر اساس گزارش تحقیق در مورد خرابی فضاپیمای لوئیس، این مشکل ناشی از طراحی نامناسب سیستم کنترلی بود و فضاپیما در فاز عملیات اولیه خود به حد کافی کنترل نشده بود.

قابلیت اطمینان تابعی از زمان است. شیوه‌ای که در آن زمان مشخص می‌شود بستگی به ماهیت سیستم موردنظر دارد. به‌عنوان‌مثال، اگر یک سیستم مانند یک فضاپیمای انتظار می‌رود که مأموریت خود را در یک دوره زمانی مشخص تکمیل کند، احتمالاً زمان موردنظر به‌صورت زمان تقویمی و یا چند ساعت تعریف می‌شود. برای نرم‌افزار، فاصله زمانی اغلب در واحدهای زمانی و یا طبیعی مشخص می‌شود. یک واحد طبیعی یک واحد مربوط به مقدار پردازش انجام‌شده توسط یک محصول مبتنی بر نرم‌افزار نظیر صفحات خروجی، تراکنش‌ها، job ها و یا query ها تلقی می‌گردد. قابلیت اطمینان احتمال موفقیت را بیان می‌کند. با فرض آن‌که سیستم در زمان صفر درست عمل کند، عدم اطمینان Q(t) در زمان t، احتمال خرابی سیستم در بازه زمانی [0, t] می‌باشد. عدم اطمینان احتمال خرابی را بیان می‌کند. رابطه عدم اطمینان و قابلیت اطمینان با فرمول $Q(t) = 1 - R(t)$ بیان می‌شود.

۲-۲-۲) دسترسی

سیستم‌های نسبتاً کمی طراحی‌شده‌اند که به‌طور مداوم، بدون وقفه و بدون نگه‌داری کار کنند. در بسیاری از موارد، نه‌تنها به‌احتمال خرابی علاقه‌مندیم، بلکه به تعداد خرابی‌ها و مدت‌زمان رفع خرابی، علاقه‌مند هستیم. برای چنین کاربردهایی، تمایل داریم آن بخش از زمان‌که سیستم در حالت عملیاتی است بیش‌ترین مقدار ممکن شود. به این ویژگی قابلیت دسترسی گویند. قابلیت دسترسی A(t) یک سیستم در زمان t، احتمال آن است که سیستم در زمان t درست کار کند.

---

[1] Lewis

A(t) قابلیت دسترسی لحظه‌ای و یا نقطه‌ای نام دارد. بیش‌تر اوقات نیاز به محاسبه قابلیت دسترسی بازه‌ای داریم و با رابطه ذیل بیان می‌شود:

$$A(T) = \frac{1}{T}\int_0^T A(t)dt \qquad ۱-۲$$

A(T) مقدار متوسط قابلیت دسترسی در بازه زمانی T هست. این بازه زمانی طول عمر سیستم و یا زمان موردنیاز برای تکمیل یک فرآیند می‌باشد. درنهایت، در بیش‌تر مواقع بعد از چند وضعیت گذرا، قابلیت دسترسی مقداری مستقل از زمان تلقی می‌شود. قابلیت دسترسی در حالت پایدار با رابطه ذیل بیان می‌شود:

$$A(\infty) = \lim_{T\to\infty}\frac{1}{T}\int_0^T A(t)dt \qquad ۲-۱$$

اگر نتوان سیستمی را تعمیر کرد، A(t) با مقدار قابلیت اطمینان برابر می شود، یعنی A(t) احتمال خراب نشدن سیستم در فاصله زمانی ۰ تا t می‌باشد؛ بنابراین اگر مقدار T به سمت بی‌نهایت میل کند، قابلیت دسترسی در حالت پایدار به صفر میل می کند:

$$A(\infty) = 0$$

اغلب اوقات قابلیت دسترسی در حالت پایدار با مدت‌زمان ازکارافتادگی سیستم در یک سال بیان می‌شود. جدول ۲-۱ مقادیر قابلیت دسترسی و مدت‌زمان خرابی مربوطه را نشان می‌دهد.

جدول ۲-۱) قابلیت دسترسی و مدت‌زمان خرابی سیستم

قابلیت دسترسی (٪)	زمان عدم دسترسی در یک سال
۹۰	۳۶,۵ روز
۹۹	۳,۶۵ روز
۹۹,۹	۸,۷۶ ساعت
۹۹,۹۹	۵۲ دقیقه
۹۹,۹۹۹	۵ دقیقه
۹۹,۹۹۹۹	۳۱ثانیه

در دسترس بودن به‌طورمعمول به‌عنوان یک سنجش قابلیت اطمینان برای سیستم‌هایی که تحمل وقفه‌های کوتاه را دارند، محسوب می شود. سیستم‌های شبکه‌ای مانند سوئیچینگ تلفن و سرورهای وب ازاین‌د ست تلقی می شوند. یک مشترک تلفنی انتظار دارد یک تماس را بدون وقفه برقرار و تکمیل کند. بااین‌حال، خرابی چنددقیقه‌ای در طول یک سال قابل‌قبول در نظر گرفته می‌شود. نظرسنجی‌ها نشان می‌دهند که یک مشتری آنلاین توقع دارد یک صفحه وب در عرض ۲ ثانیه بارگیری شود [۲]. این به بدان معنی است که وب‌سایت‌های تجارت الکترونیک همیشه باید در دسترس باشند و اگر تعداد زیادی از خریداران به‌طور هم‌زمان به آن‌ها د سترسی پیدا کنند، باید سریعاً پاسخ دهند. مثال دیگر، سیستم کنترل برق است. مشترکان انتظار دارند که برق در تمام طول شبانه‌روز و در هر شرایط آب و هوایی موجود باشد. قطع برق طولانی‌مدت ممکن است منجر به صدمات جانی و زیان مالی شود. به‌عنوان‌مثال، در ۱۱ مارس ۲۰۰۱، برق منطقه صنعتی کیستا که در حومه شهر استکهلم قرار دارد، به علت آتش‌سوزی در تونل مجاور نیروگاه قطع شد. از ۷:۰۰ صبح تا ۸:۳۵ بعدازظهر ۵۰۰۰۰ کارمند، ۷۰۰ واحد تجاری و ۳۰۰۰۰ نفر با قطعی برق روبرو

شـدند [۹]. گرمایش، تهویه، پمپ آب شـیرین، تلفن و چراغ‌های راهنما ازکارافتادند. تمام کامپیوترها، دستگاه‌های قفل و سیستم‌های امنیتی مختل شدند. کل خسارت قطع برق کیستا در حدود ۱۲٫۸ میلیون یورو تخمین زده‌شده است [۱۵].

۲-۲-۳) ایمنی

ایمنی می‌تواند توسعه قابلیت اطمینان باشد، به‌عبارت‌دیگر این ممکن است این قابلیت اطمینان به‌جای خرابی‌ها حوادث غیر خطرناک ایجاد کند. ازنقطه‌نظر قابلیت اطمینان، تمامی خرابی‌ها یکسان هستند. برای ملاحظات ایمنی، خرابی‌ها به دو رده ایمن و غیر ایمن تقسیم می‌شوند.

به‌عنوان‌مثال، یک سیستم هشدار را در نظر بگیرید. ممکن است سیستم در هنگام وجود خطر به‌درستی عمل نکند و یا ممکن است بدون وجود خطر هشدار نادرستی صادر کند. حالت اول به‌عنوان یک خرابی غیر ایمن طبقه‌بندی می‌شود. حالت دوم یک خرابی ایمن در نظر گرفته می‌شود. به‌طور رسمی، ایمنی به شرح زیر تعریف می‌گردد:

با فرض آن‌که سیستمی در زمان ۰ به‌درستی عمل کند، ایمنی S(t) در لحظه t احتمال آن است که سیستم در زمان t درست عمل کند و یا آن‌که عملیات خود را به روشی ایمن متوقف کند.

ایمنی برای کاربردهایی از نوع ایمنی بحرانی که خرابی می‌تواند منجر به صدمات جسمی، مرگ و یا فاجعه زیست‌محیطی گردد، ضروری است [۸]. کنترل‌کننده‌های صنعتی، قطار، خودرو، سیستم‌های هوائی، سیستم‌های پزشکی و سیستم‌های نظامی مثال‌هایی ازاین‌دست محسوب می‌شوند.

تاریخچه نشان می‌دهد که بسیاری از شکست‌های ناامن ناشی از اشتباهات انسانی است. به‌عنوان‌مثال، در ۲۶ آوریل ۱۹۸۶، طراحی بد برای تولید برق از انرژی باقی‌مانده در توربو ژنراتورها منجر به حادثه چرنوبیل گردید [۱۲]. برای انجام آزمایش، تمام سیستم‌های خاموش‌کننده خودکار و سیستم خنک‌کننده هسته اورژانسی راکتور، به‌طور دستی خاموش شده بود. آزمایش توسط یک مهندس که با تسهیلات رآکتور آشنا نبود، رهبری می شد. با علت این دو عامل، زمانی که همه‌چیز نادرست پیش می‌رفت، امکان ملغی کردن آزمایش میسر نبود.

۲-۳) اختلالات

عوامل اختلال با سه واژه ایراد[1]، خطا[2] و خرابی[3] تعریف می شوند. این سه واژه به‌طور مشترک وجود یک اشکال را بیان می‌کنند. تفاوت آن‌ها در آن است که ایراد در لایه فیزیکی، خطا در سطح محاسباتی و خرابی در سطح سیستم روی می‌دهد [۴].

---

[1] Fault
[2] Error
[3] Failure

۲-۳-۱) خطا، اشتباه و خرابی

خطا یک نقص و یا عیبی است که در قطعات سخت‌افزاری و یا نرم‌افزاری وجود دارد. اتصال کوتاه میان دو اتصال مجاور، شکستگی یک پین و اشکال نرم‌افزاری مثال‌هایی ازاین‌دست محسوب می شوند. اشتباه یک انحراف از صحت یا دقت در محاسبه است که براثر خطا روی می‌دهد. اشتباه‌های معمولاً با مقادیر نادرست در حالت سیستم مرتبط هستند. یک مدار یا یک برنامه یک مقدار نادرست محاسبه می‌کند و یا اطلاعات نادرست دریافتی در هنگام انتقال داده‌ها مثال‌هایی ازاین‌دست محسوب می‌شوند.

یک خرابی کارایی ضعیف سیستم در قیاس با کارایی مورد انتظار محسوب می‌گردد. زمانی که خدمات ارائه شده به کاربران توسط سیستم در یک بازه زمانی مشخص منطبق با مشخصات تعریف شده نباشد، سیستم دچار خرابی شده است [۱۶]. سیستم زمانی دچار خرابی می‌گردد که عملکرد آن با مشخصات تعریف شده منطبق نباشد و یا مشخصات سیستم به‌طور کامل و دقیق عملکرد سیستم را تعریف نکرده باشد.

خطاها دلایل اشتباه‌ها و اشتباه‌ها به‌نوبه خود دلایل خرابی هستند. برای مثال، یک نیروگاه تولید برق را در نظر بگیرید، در این مکان سیستم‌های که با کامپیوتر کنترل می شوند، وظیفه نظارت بر دما، فشار و سایر ویژگی‌های فیزیکی تجهیزات را بر عهده‌دارند. سنسور سرعتی که موجب توقف توربین می‌شود را گزارش می‌کند. خرابی‌هایی ازاین‌دست موجب ارسال بخاری بیش از نیاز توربین شده (اشتباه) و موجب توقف و خاموش شدن توربین می‌گردد و از صدمه دیدن آن جلوگیری می‌کند. اکنون برقی تولید نمی‌شود (سیستم دچار خرابی از نوع ایمن شده است).

تعاریف سطوح فیزیکی، محاسباتی و سیستم در هنگام استفاده از نرم‌افزار کمی مبهم به نظر می‌رسد. در این کتاب، کد برنامه را به‌عنوان سطح فیزیکی، مقادیر حالت برنامه را به‌عنوان سطح محاسبات و سیستم نرم‌افزاری که برنامه را اجرا می‌کند، سطح سیستم تفسیر می‌کنیم. به‌عنوان‌مثال، یک اشکال در برنامه یک خطا است، مقدار نادرست ناشی از این خطا یک اشتباه بوده و ازکارافتادن سیستم‌عامل، یک خرابی سیستم محسوب می‌گردد.

هر خطای منجر به اشتباه نشده و هر اشتباهی منجر به خرابی نمی‌گردد. این موضوع در مورد نرم‌افزار نمود بیش‌تری دارد. بعضی از برنامه‌ها دچار مشکل هستند ولی پیدا کردن آن‌ها بسیار دشوار می‌باشد، به دلیل آن‌که خرابی‌ها فقط در حالت بسیار خاصی روی می‌دهند. برای مثال، در نوامبر ۱۹۸۵ مبلغی در حدود ۳۲ میلیارد دلار در بانک نیویورک انتقال یافت و منجر به ضرر ۵ میلیون دلاری شد. این مشکل به دلیل بررسی نکردن اشتباه سرریز[1] در یک شمارنده ۱۶ بیتی روی داد [۷]. در سال ۱۹۹۴ مشخص شد پردازنده پنتیوم ۱ شرکت اینتل بعضی از محاسبات تقسیم ممیز شناور را به‌درستی محاسبه نمی‌کند. برای مثال، با تقسیم ۵۵۰۵۰۰۱ بر ۲۹۴۹۱۱ به‌جای ۱۸,۶۶۶۵۱۹۷ جواب ۱۸,۶۶۶۰۰۰۹۳ تولید می‌شد. این اشتباه به دلیل حذف ۵ داده در یک جدول ۱۰۶۶ مقداری که مورداستفاده الگوریتم تقسیم می‌گرفت، حادث شد. این

[1] overflow

فیلدها می‌بایست با مقدار ثابت ۲ مقداردهی شوند ولی به دلیل خالی بودن توسط پردازنده با مقدار صفر تعبیر شدند. شیوه‌ای که یک سیستم دچار خرابی می‌گردد **حالت خرابی** نامیده می‌شود. برای مثال، ممکن یک قفل هنگام خرابی بازشده و یا قفل گردد.

حالت‌های خرابی برحسب حوزه کاری (زمان‌بندی و ارزشی)، استنباط کاربران از خرابی (خرابی‌های متناقض و یا نامتناقض) و پیامدهای آن بر محیط اطرافشان (خرابی‌های جزئی، اصلی و مصیبت‌بار) طبقه‌بندی می‌شوند [۳]. عواقب واقعی یا پیش‌بینی شده خرابی، **اثرات خرابی**[1] نام دارد. اثرات خرابی برای شناسایی اقدامات اصلاحی و جبران خرابی سیستم تجزیه‌وتحلیل می‌شوند. اثرات خرابی در برنامه‌ریزی، نگه‌داری و آزمون سیستم استفاده می‌شود. در برنامه‌های ایمنی بحرانی، ارزیابی اثرات شکست یک فرایند ضروری در طراحی سیستم از اوایل مرحله توسعه تا مراحل طراحی و آزمون محسوب می‌شود [۲۵].

۲-۳-۲) منشأ خطاها

همان‌طور که در بخش قبلی اشاره شد، خرابی‌ها ناشی از اشتباه‌ها و اشتباه‌ها ناشی از خطاها هستند. خطاها به‌نوبه خود از مشکلات متعددی در مرحله مشخصات، پیاده‌سازی، ساخت و فرایند طراحی نشأت می‌گیرند. همچنین می‌توانند از عوامل خارجی مانند اختلالات زیست‌محیطی یا اقدامات انسان، تصادفی و یا عمدی ناشی شوند. به‌طور گسترده، می‌توان منابع خطاها را به چهار گروه مشخصات نادرست، پیاده‌سازی نادرست، اشکال در ساخت و عوامل بیرونی تقسیم کرد [۱۴].

مشخصات نادرست نتیجه الگوریتم‌های نادرست، معماری‌ها و یا نیازمندی‌های نادرست می‌باشد. یک مثال معمول، مواردی است ویژگی‌های محیطی که سیستم باید در آن کار کند، نادیده گرفته شود. سیستم ممکن است در بیش‌تر مواقع به‌درستی عمل کند، اما ممکن است مواردی از کارایی نادرست آن به چشم بخورد. خطاهای نشأت گرفته از مشخصات نادرست، **خطای مشخصات** نام دارد. خطای مشخصات بسیار رایج هستند، برای مثال، در سیستم‌های SOC مشخصات هسته‌ها با مشخصات موردنظر طراح مطابقت کامل ندارند. قسمتی از این مشکل به مسئله حفظ حقوق مؤلف و مخفی کردن جزئیات لیست گره‌ها[2] و طرح‌بندی[3] برمی‌گردد [۲۳]. خطاهای ناشی از پیاده‌سازی نادرست که اغلب به‌عنوان **خطاهای طراحی** نامیده می‌شوند، زمانی رخ می‌دهد که پیاده‌سازی سیستم به‌طور دقیق مشخصات را پیاده‌سازی نمی‌کند. در سخت‌افزار، خطای طراحی شامل انتخاب قطعه ضعیف، اشتباهات منطقی، زمان‌بندی نادرست و هم‌زمانی بد می‌باشد. در نرم‌افزار، نمونه‌هایی از اجرای نادرست، اشکالات در کد برنامه و استفاده مجدد از اجزای نرم‌افزاری هستند. نرم‌افزار به شدت به فرضیات مختلف در مورد محیط عملیاتی آن متکی است. اگر این فرض‌ها در محیط جدید نادرست باشند، احتمالاً اشتباه‌هایی رخ می‌دهند. در نرم‌افزار اشکالات در کدهای برنامه و یا استفاده مجدد از کد به‌گونه‌ای ضعیف مثال‌های از پیاده‌سازی نادرست محسوب می‌شوند.

---

[1] failure effects
[2] netlists
[3] layouts

نرم‌افزار به‌شدت به فرضیات مختلف در مورد محیط عملیاتی متکی است. ممکن است خطاها به دلیل نادرست بودن فرض‌ها در محیط جدید حادث شوند. حادثه موشک آریان ۵ مثالی از یک خطا است که به علت استفاده مجدد از قطعات نرم‌افزاری به وجود آمده است [۱۷]. در ۴ ژوئن ۱۹۹۶ این موشک ۳۷ ثانیه بعد از برخاستن به علت نتایج حاصل از تبدیل یک عدد اعشاری ۶۴ بیتی به یک عدد ۱۶ بیتی صحیح منفجر شد. عدد اعشاری قابل‌نمایش در ۱۶ بیت نبود. براثر اشتباه سرریز حافظه کامپیوتر پاک شد. موشک محتویات حافظه را به‌عنوان دستوری برای نازل تلقی کرده و درنهایت موشک منفجر شد.

بسیاری از خطاهای سخت‌افزاری ناشی از مشکل قطعات می‌باشند. این مشکلات شامل: ساخت نادرست قطعه، اشتباه تصادفی ابزار و فرسودگی قطعات می‌باشند. قابلیت اطمینان پایین سخت‌افزارهای اولیه یکی از دلایل اصلی استفاده از سیستم‌های تحمل‌پذیر اشتباه در سیستم‌های محاسباتی بود. با پیشرفت فن‌آوری ساخت ادوات نیمه هادی قابلیت اطمینان قطعات افزایش یافت و خطاهای ناشی از عیوب سخت‌افزاری به‌طور قابل‌ملاحظه‌ای کاهش پیدا کرد.

دلیلی چهارم خطاها عوامل بیرونی هستند که از خارج سیستم نظیر محیط، کاربر و یا اپراتور ناشی می‌شوند. عوامل خارجی پدیده‌های هستند که بر عملکرد سیستم تأثیر می‌گذارند، پدیده‌های نظیر دما، ارتعاش، تخلیه الکترواستاتیک، تابش هسته‌ای و امواج الکترومغناطیس بر روی ورودی‌های سیستم تأثیر می‌گذارند. برای مثال، تابش یک عامل بیرونی است که موجب می‌شود مقدار یک سلول فلیپ‌فلاپ برعکس شود. مشکلات کاربری و یا اپراتوری می‌توانند تصادفی و یا تعمدی باشد. برای مثال، ممکن است کاربر با دستوری نادرست موجب خرابی سیستم شده و یا مقادیر اولیه نرم‌افزار نادرست باشد. ویروس‌های کامپیوتری و یا دستورات هکر مثالی از مشکلات عامدانه تلقی می‌گردند.

فرض کنید در حال رانندگی در مکانی دورافتاده در استرالیا هستید و اتومبیل شما از کار می‌افتد. ازنظر شما این اتومبیل دارای یک خطا فنی است، اما چه خطای منجر به این خرابی شده است؟ در اینجا تعدادی از احتمالات ممکن را فهرست کرده‌ایم:

۱) ممکن است طراح خودرو به خاطر پاره‌ای از دلایل محدوده دمای مناسب را لحاظ نکرده باشد:

- ازنظر نادرستی مشخصات، افرادی که مشخصات فنی را آماده می‌کردند، دمای بالای ۵۰ درجه را پیش‌بینی نکرده‌اند.
- ازنظر خطا طراحی محدوده ۵۰ درجه پیش‌بینی‌شده بود ولی طراح از آن صرف‌نظر کرده است.
- ازنظر مشکل پیاده‌سازی کارخانه سازنده دقیقاً طرح را اجرا نکرده است.

۲) شما تابلوی کانگورو تا فاصله ۱۰۰ مایلی را می‌بینید، اما به آن توجهی نمی‌کنید و با کانگورو تصادف می‌کنید. به این مسئله اشتباه کاربر گویند.

۳) کارگر بزرگراه اشتباهاً تابلوی کانگورو تا فاصله ۱۰۰ مایلی را با تابلوی حداکثر سرعت ۱۰۰ مایل بر ساعت جابجا می‌کند، به این مشکل اشتباه اپراتور گویند.

۴) دریچه تخلیه رادیاتور دچار نشتی شده، رادیاتور از کار می‌افتد و موتور بیش‌ازحد گرم شده است، این مشکل عملکرد نادرست تجهیزات به علت فرسودگی است.

۵) یک مترو با ماشین شما برخورد می‌کند، این مشکل یک اشتباه محیطی است.

۲-۳-۳) خطاهای حالت مشترک

خطا حالت مشترک خطای است که در دو و یا چند قطعه افزونه به‌طور هم‌زمان روی می‌دهد. خطا حالت مشترک براثر پدیده‌هایی حادث می‌شوند که وابستگی را میان واحدهای افزونه ایجاد می‌کنند و همین خطاها این واحدها را از کار می‌اندازند، برای مثال، باس های ارتباطی مشترک و عوامل محیطی مشترک نمونه‌هایی ازاین‌دست محسوب می‌شوند. اگر سیستمی فقط به یک منبع تغذیه، یک سیستم خنک‌کننده و یا یک باس I/O تکیه کند، نسبت به خطا حالت مشترک آسیب‌پذیر است. یکی دیگر از دلایل خطای حالت مشترک اشتباه طراحی است، یک طراحی نامناسب موجب می شود کپی‌های افزونه سخت‌افزاری و یا نرم‌افزاری در شرایط یکسان از کار بیفتند. تنها راه غلبه بر طراحی حالت مشترک، طراحی متنوع است. تنوع طراحی، پیاده‌سازی چند واحد عملیاتی است. ثابت‌شده است که برای کاربردهای مبتنی بر کامپیوتر تنوع در سطوح بالایی و انتزاعی مؤثرتر است [۵]. برای مثال، تنوع در الگوریتم نسبت به جزئیات پیاده سازی نظیر انتخاب زبان برنامه‌نویسی کارایی بیش‌تری دارد. از آنجائی که طرح‌های متنوع باید یک مشخصات مشترک را پیاده‌سازی کنند، همیشه امکان وابستگی در فرآیند پالایش مشخصات وجود دارد. طرح‌های کاملاً متنوع با استفاده از تیم‌های طراحی جداگانه، قوانین طراحی مختلف و ابزارهای نرم‌افزاری وابستگی را از بین می‌برند. این موضوع در بخش ۷-۳-۴ موردبررسی قرار می‌گیرد.

۲-۳-۴) خطاهای سخت‌افزاری

در این بخش دو رده اصلی خطاهای سخت‌افزاری نظیر خطاهای دائمی[1] و گذرا[2] را موردبررسی قرار خواهیم داد. در مرحله بعد نحوه مدل کردن خطاهای سخت‌افزاری را نشان می‌دهیم.

۲-۳-۴-۱) خطاهای دائمی و گذرا

خطاهای سخت‌افزاری برحسب مدت‌زمان به سه گروه دائمی، گذرا و میانی تقسیم می شوند[۳]. یک خطا دائمی تا زمانی که عملیات برطرف کردن آن انجام نشود، باقی می ماند. این خطاها براثر عوامل فیزیکی نظیر اتصال کوتاه در مدار، قطع ارتباطات درونی و مشکلات حافظه روی می‌دهند. یک اشتباه گذرا و یا اشتباه نرم[3] برای مدت کوتاهی وجود دارد. یک خطا گذرا که به صورت دوره‌ای فعال می شود،

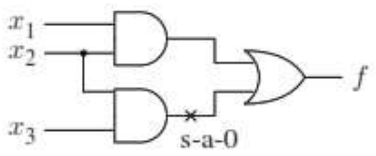
---

[1] permanent
[2] transient
3 soft-error

یک خطا متناوب نام دارد. به علت کوتاه بودن مدت خطاهای گذرا، آ شکار شدن به‌و سیله ا شتباه‌هایی که در نتایج انتشار می‌یابند انجام می‌گیرد. خطاهای گذرا معمول‌ترین خطا در آی‌سی‌های امروزی محسوب می‌شوند. برای مثال در حدود ۹۸٪ خطا های حافظه رم دینامیکی از نوع گذرا هستند[۲۴]. هر یک گیگابایت حافظه رم دینامیکی در هرروز به‌اندازه یک بیت دچار اشتباه می‌شود[۲۶]. این خطاها غالباً از محیط اطراف نظیر ذرات آلفا، تخلیه الکتریکی، افت ولتاژ و یا گرمای بیش‌ازحد ناشی می‌شوند. خطاها متناوب براثر خطاهای پیاده سازی، کهنگی[1]، فر سودگی[2] و شرایط غیرمنتظره حادث می شوند. برای مثال، یک لحیم‌کاری ضعیف به همراه ارتعاش می‌تواند منجر به یک خطا متناوب گردد.

۲-۳-۴-۲) مدل‌های خطا

این امکان وجود ندارد که بتوان تمامی خطاهای ممکن قابل اتفاق در یک سیستم را فهرست کرد. برای ارزیابی بیش‌تر خطاها، فرض بر این است که آن‌ها بر اساس برخی از مدل‌های خطا رفتار می‌کنند [۱۱]. مدل خطا تلاش می‌کند تا اثر یک خطا که امکان رخ دادن آن وجود دارد را توصیف کند. شایع‌ترین مدل خطا سطح گیت، گیرکردن[3] در یک سطح منطقی است. چنین خطای موجب می شود تا یک خط در یک مدار منطقی به‌صورت دائمی در یک یا صفر منطقی قرار گیرد [۱]. فرض بر آن است که عملکرد اصلی مدار براثر خطا تغییر نمی‌کند. برای مثال، یک مدار ترکیبی به یک مدار ترتیبی تبدیل نمی‌شود و یا یک گیت AND به OR تبدیل نمی‌شود. با توجه به‌سادگی و کارآمدی این مدل، گیرکردن در یک سطح منطقی معمول‌ترین مدل خطا محسوب می شود. در مدل گیرکردن چندگانه، چندین خط مدار در یک و یا چند مقدار منطقی گیر می‌کنند. یک مدار با k خط می‌تواند در 2k حالت منفرد و یا 3^k-1 حالت چندگانه گیر کند؛ بنابراین امکان تست کردن تمامی خطاهایی ازاین‌دست میسر نیست. با مقایسه جدول صحت تابع $f(x_1, x_2,\ldots, x_n)$ که با مدار درست پیاده‌سازی می‌شود با جدول صحت $f^{\alpha}(x_1, x_2,\ldots, x_n)$ که با مدار مشکوک پیاده سازی شده، می‌توان خطا گیرکردن را استخراج کرد. مقادیری از ورودی $(x_1, x_2,\ldots, x_n)$ که منجر به برقراری ذیل نامساوی ذیل شود، آزمونی برای خطا گیرکردن α محسوب می‌شود $f^{\alpha}(x_1, x_2,\ldots, x_n) \neq f(x_1, x_2,\ldots, x_n)$.



شکل ۲-۱) مدار منطقی برای مثال ۲-۱

[1] ageing
[2] wear-out
[3] stuck-at

جدول ۲-۲) جدول صحت برای توابع منطقی پیاده سازی شده با مدار درست (f) در شکل ۲-۱ و مدار معیوب (f^{α})

x1	x2	x3	f	f^a
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

مثال ۲-۱) به مدار منطقی شکل ۲-۱ توجه کنید. این مدار تابع $f(x_1,x_2,x_3) = x_1x_2 + x_2x_3$ را پیاده سازی می‌کند. وجود مشکل گیرکردن در صفر با علامت × مشخص شده است (با نماد s-a-0)، این تابع به تابع $f^{\alpha}(x_1,x_2) = x_1x_2$ بدل می شود. جداول صحت دو تابع f و تابع f^{α} بدل می شود. به ازای ورودی $(x_1,x_2,x_3) = (0,1,1)$ نامساوی $f(0,1,1) \neq f^{\alpha}(0,1,1)$ برقرار می‌باشد، بنابراین مقدار ورودی (0,1,1) یک حالت آزمون برای این مشکل مدار محسوب می‌شود. مثال دیگری از خطا در سطح گیت پل شدن و یا اتصال کوتاه دو مسیر مجاور در یک مدار منطقی می‌باشد، این خطا موجب AND و یا OR شدن سیگنال‌ها می‌گردد. یک خطا از نوع پل موجب برقراری فیدبک شده و مدار از حالت ترکیبی به مدار ترتیبی بدل می‌شود. تست کردن خطا پل نسبتاً دشوارتر می‌باشد، چراکه این خطا به چندین خط ارتباطی بستگی دارد.

۲-۳-۴) خطاهای نرم‌افزاری

نرم‌افزار دلیل نیمی از خرابی های سیستم می گردد. ویژگی های خاص نرم‌افزار، منابع خطا آن را مشخص می کند. نرم‌افزار از خرابی‌های ساخت متأثر نیست، دچار کهنگی و یا فرسودگی نمی‌شود. برخلاف قطعات الکترونیکی و یا مکانیکی سخت‌افزار، نرم‌افزار دچار تغییر شکل نمی شود، نمی شکند و یا از عوامل محیطی تأثیر نمی گیرد. با فرض قطعی بودن عملکرد نرم‌افزار تا زمانی که مشکلی برای سخت‌افزار ذخیره کننده نرم‌افزار روی ندهد، رفتار آن در محیط‌های یکسان همواره مشابه می‌باشد. به همین دلیل عمده دلایل مشکلات نرم‌افزاری به خطا زمان طراحی برمی گردد [۱۸]. خطاهای طراحی به عوامل انسانی بستگی دارد و به همین دلیل جلوگیری کردن از آن‌ها دشوارتر می‌باشد. در سخت‌افزار هم اشتباه طراحی وجود دارد ولی نوع آن با نرم‌افزار تفاوت دارد، برای مثال، خرابی های هنگام ساخت و یا خطاهای گذرا که از محیط اطراف نشأت می گیرند. با به‌روزرسانی نرم‌افزار در جهت افزایش امنیت آن اشکالات جدیدی بروز می کند. به‌روزرسانی نرم‌افزار با طراحی مجدد، پیاده‌سازی دوباره و استفاده از ماژول هایی که بهتر توسعه یافته‌اند، انجام می گیرد و موجب افزایش قابلیت اطمینان و امنیت آن می گردد[۱۹]. به‌روزرسانی عملکرد نرم‌افزار، عملکرد آن را بهبود می‌بخشد؛ اما نیت خوب می تواند عواقب بدی به دنبال داشته باشد. برای مثال، در سال ۱۹۹۱ تغییر سه خط کد در برنامه‌ای متشکل از چند میلیون کد موجب ازکارافتادن مراکز سوئیچ مخابراتی در کالیفرنیا و سواحل شرقی شد.

۲-۴) تکنیک‌های قابلیت اطمینان

تکنیک‌های قابل‌اطمینان[1]، روش‌ها و تکنیک‌های هستند که امکان توسعه یک سیستم قابل‌اطمینان را فراهم می‌سازند. تحمل خطا که مبحث این کتاب می‌باشد، یکی از این روش‌ها محسوب می‌شود. روش‌های ترکیبی نظیر جلوگیری از خطا[2]، حذف خطا [3] و پیش‌بینی خطا [4] به‌طورمعمول مورداستفاده قرار می‌گیرند [۶]. روش های جلوگیری از خطا ، از بروز خطا جلوگیری می کنند. روش حذف خطا تعداد خطاهای موجود در سیستم را کاهش می‌دهند. روش پیش‌بینی خطا ، تعداد خطاهای موجود و یا خطاهای احتمال در آینده را تخمین می‌زنند.

۲-۴-۱) تحمل خطا

هدف تحمل خطا توسعه سیستم‌های هست که باوجود خطا به‌درستی کار کنند. همان‌طوری که در بخش ۲-۱ اشاره شد، تحمل خطا نوعی افزونگی است. افزونگی این امکان را فراهم می کند، یک خطا فیلتر شده و یا به مکان مربوطه آشکار گردد و درنهایت عملیات مهار کردن و بازیابی صورت گیرد. افزونگی برای یک سیستم باقابلیت تحمل خطا ضروری است ولی کافی نیست. برای مثال، یک سیستم دوبل که به‌طور موازی کار می کند، ضرورتاً یک سیستم تحمل خطا محسوب نمی‌شوند، مگر آن‌که عملیات نظارتی انجام شود و با تحلیل نتایج یکی از دو سیستم درست انتخاب گردد.

فیلتر کردن خطاها تضمین می کند که فقط مقادیر صحیح به خروجی سیستم منتقل می‌شوند. این متد با تکنیک‌هایی نظیر تصحیح خطا، جبران خطا و یا سایر روش‌ها مانع تأثیر گرفتن سیستم از خطا می گردد. حافظه‌ای که قبل استفاده سیستم از داده‌ها، اشتباه‌ها را تصحیح می کند، مثالی ازاین‌دست تلقی می‌شود. مثال دیگر، سیستم افزونه سه ماژولاری است که از روش رأی‌گیری مبتنی بر اکثریت استفاده می کند.

روش آشکار کردن خطا، فرآیند مشخص کردن وجود خطا در سیستم می‌باشد. تست‌های پذیرش و مقایسه مثال‌هایی از تکنیک آشکار کردن خطا محسوب می شوند. تست پذیرش خطا ، راهکاری است که می‌تواند برای سیستم‌هایی که دارای اجزا تکراری نیستند مورداستفاده قرار گیرد. تست‌های پذیرش در سیستم‌های نرم‌افزاری رایج هستند [۲۲]. نتیجه یک برنامه تحت آزمون قرار می گیرد. اگر نتیجه برنامه موردقبول باشد، استفاده از آن استمرار می‌یابد. یک آزمون شکست‌خورده معرف یک خطا می‌باشد. برای سیستم‌هایی که دارای اجزای دوگانه هستند، مقایسه روش دیگری برای آشکار کردن خطا محسوب می گردد. عدم تطابق خروجی‌ها معرف وجود یک اشتباه می باشد.

مکان خطا، فرآیند مشخص کردن موقعیت خطای اتفاق افتاده می‌باشد. در حالت کلی تست پذیرش نمی‌تواند مکان خطا را مشخص کند. این تست فقط مشخص می کند قسمتی از سیستم دچار مشکل شده

---

[1] Dependability
[2] Fault prevention
[3] fault removal
[4] fault forecasting

است. به روشی مشابه هنگام عدم تطابق خروجی‌های دو ماژول امکان مشخص کردن ماژول معیوب میسر نیست.

جبران کردن اشتباه، فرآیند جداسازی مشکل و جلوگیری از انتشار آن در سیستم می‌باشد. این روش با آشکار سازی مستمر خطا، استفاده از پروتکل‌های درخواست و تائید چندگانه و آزمون تطابق سازگاری میان ماژول‌ها انجام می‌شود.

زمانی که سیستم یک قطعه معیوب را شناسایی کند آن را از مابقی قسمت‌ها ایزوله کرده و با انجام عملیات بازیابی آن را به وضعیت عملیاتی برمی‌گرداند. در این روش ممکن است قطعه معیوب با قطعه‌ای پشتیبان جایگزین گردد. درروش دیگر قسمت معیوب خاموش شده و سیستم به وضعیت عملیاتی خود با قدری تنزل در میزان کارایی بازمی‌گردد. به این روش تنزل مطبوع[1] گویند.

۲-۴-۲) جلوگیری از خطا

جلوگیری از خطا مجموعه‌ای از تکنیک‌هایی هستند که از همان ابتدا سعی می‌کنند از شکل‌گیری خطا و یا رخ دادن آن جلوگیری کنند. روش جلوگیری از خطا با استفاده از تکنیک‌های کنترل کیفیت در حین مراحل تعیین مشخصات، پیاده‌سازی و ساخت قابل‌دستیابی است. برای سخت‌افزار این مراحل شامل بازبینی چندباره طراحی، غربال‌گری قطعه‌ها و تست می‌باشد[۲۷]. برای نرم‌افزار این مراحل عبارت‌اند از: برنامه‌نویسی ساخت‌یافته، استفاده از روش ماژولار[2] و تکنیک‌های صحت سنجی رسمی[3][۲۹]. یک بازبینی دقیق طرح می‌تواند تعداد زیادی از خطاهای تعیین مشخصات را حذف کند. اگر طراحی به‌طور دقیق مورد آزمون قرار گیرد تعداد مشکلات و قطعات معیوب به نحو چشم‌گیری کاهش می‌یابد. برای از بین بردن خطاهای ناشی از عوامل بیرونی نظیر رعدوبرق و یا تابش می‌توان از محافظت در برابر این عوامل و ... استفاده نمود. با آموزش مستمر و تهیه روال‌های نگه‌داری می‌توان از خطاهای اپراتوری و یا کاربری جلوگیری کرد. با استفاده از فایروال و ابزارهای امنیتی می‌توان از خراب‌کاری‌های مغرضانه نظیر ویروس‌ها و یا حملات هکرها جلوگیری کرد.

۲-۴-۳) حذف خطا

حذف خطا مجموعه‌ای از تکنیک‌ها برای کاهش تعداد خطاها در سیستم می‌باشد. روش حذف خطا در حین فرایند توسعه و یا در زمان بهره‌برداری از سیستم قابل‌اجرا است. این روش در حین توسعه شامل سه بخش صحت سنجی، تشخیص خرابی و رفع خرابی می‌باشد. حذف خطا در زمان بهره‌برداری سیستم شامل عملیات نگه‌داری، تصحیح خطا و جلوگیری از بروز آن می‌باشد. در صحت‌سنجی بررسی می‌شود که آیا سیستم با شرایط مدنظر ما مطابقت دارد و یا خیر؟ در صورت عدم تطابق دو فاز دیگر انجام‌شده و عملیات پیدا کردن خرابی و رفع آن اجرا می‌شوند. درروش نگه‌داری برای جلوگیری از خرابی، پیش از وقوع خطا

---

[1] graceful degradation
[2] modularization
[3] formal verification techniques