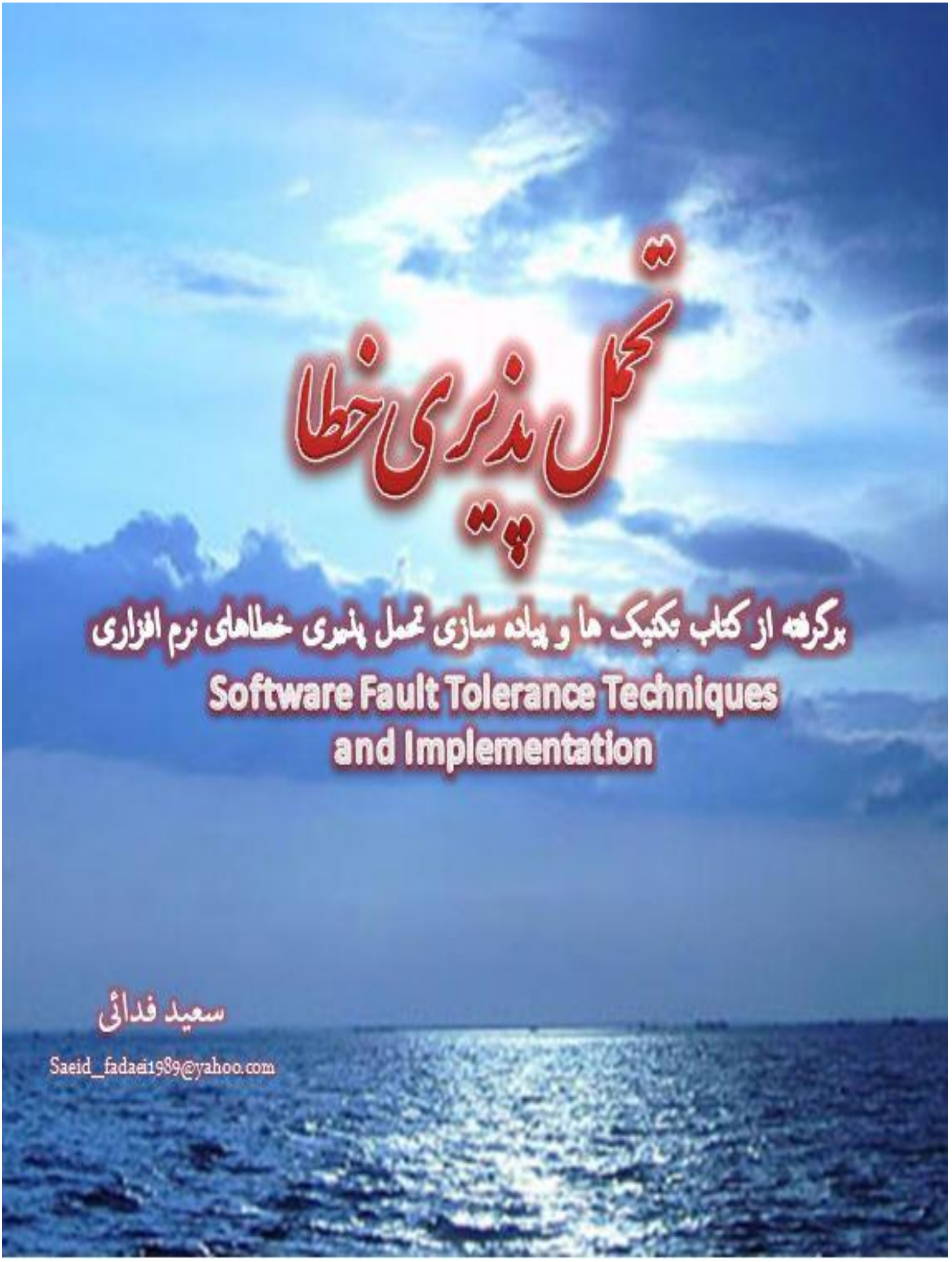

تحمل پذیری خطا
برگرفته از کتاب تکنیک ها و پیاده سازی تحمل پذیری خطاهای نرم افزاری
Software Fault Tolerance Techniques
and Implementation
سعید فدائی
Saeid_fadaei1989@yahoo.com

مقدمه:

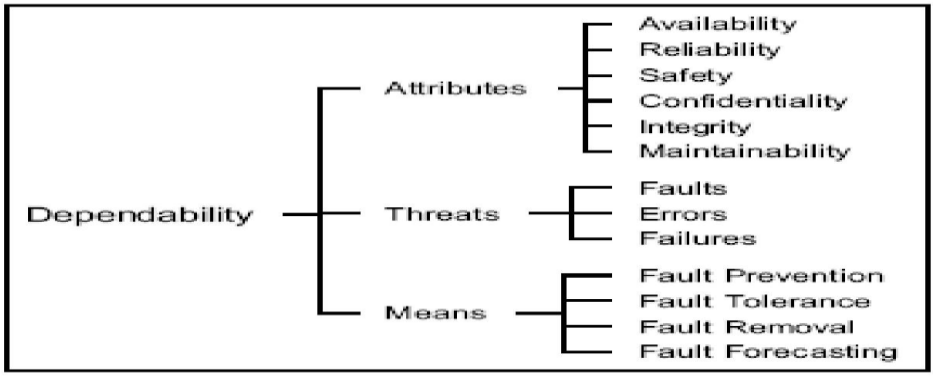
امروزه با هرچه پیچیده تر شدن سیستم های کامپیوتری ، وجود عدم قطعیت در اغلب بخش های یک سیستم و لزوم دخالت عوامل متعدد در حصول خروجی ، طراحی یک سیستم که با بروز مشکل در هر یک از قسمت های آن ، دچار نقص عمده نشود و قادر باشد عملکرد صحیح خود را حفظ نماید و صرفا با تغییری در کارایی کلی ، هدف نهایی را برآورده سازد ، بسیار ضروری است . به این ترتیب می توان تعریفی از یک سیستم تحمل پذیر خطا به شرح زیر ارائه نمود.

یک سیستم تحمل پذیر خطا ، سیستمی است که بتواند وظایفی که برایش مشخص شده است را حتی با وجود خطاهای نرم افزاری و خرابی های سخت افزاری به درستی به انجام برساند . تحمل پذیری خطا به دلیل مصرف رو به فزونی هر چه بیشتر کامپیوترها در کاربردهای مختلف در زندگی بشر ، بسیار مورد توجه قرار گرفته است و در واقع به علت اهمیت کاربردها ، تحمل پذیری خطا یک ویژگی مهم در سیستم ها محسوب می شود.

اهداف تحمل پذیری خطا

تحمل پذیری خطا یک ویژگی مهم است که به منظور دستیابی به برخی اهداف طراحی در سیستم منظور می شود. درست همانطور که یک طراحی باید بسیاری از اهداف و کارایی های مطلوب را برآورده سازد ، باید اهدافی از قبیل قابلیت اطمینان ، دسترس پذیری ، امنیت ، قابلیت اجرا ، قابلیت نگهداری و آزمون پذیری را فراهم نماید .واژه اتکاپذیری شامل مفاهیم قابلیت اطمینان ، دسترس پذیری ، قابلیت نگهداری ، امنیت ، قابلیت اجرا و آزمون پذیری می باشد .

اتکاپذیری (قابلیت اتکا) یک مفهوم کیفی و عام است . قابلیت اتکای یک سیستم برای ارائه و تحویل سرویس موردنظر به کاربران است به گونه ای که بتوان به ارائه شدن آن سرویس اطمینان داشت . منظور از سرویس تحویلی توسط سیستم ، رفتار سیستم است ، آن گونه که توسط کاربران دریافت و ادراک می شود .مفهوم قابلیت اتکا را می توان براساس درخت قابلیت اتکا مطابق شکل زیر ترسیم نمود .



شکل 1- درخت قابلیت اتکا

از آنجایی که قابلیت اتکا یک مفهوم کیفی است ، برای ارزیابی مهندسی و دقیق تر سیستم ها چند خاصیت برای سیستم های قابل اتکا تعریف شده است که عموما مفاهیم کمی هستند .

در ادامه تعریف مختصری از آنها بیان می گردد.

1) قابلیت اطمینان

قابلیت اطمینان که با R(t) نمایش می دهند ، احتمال شرطی این است که سیستم در بازه زمانی [t۰t ,] به درستی کار کند به شرط اینکه سیستم در ابتدای بازه زمانی t. درست بوده باشد . قابلیت اطمینان معیاری برای پیوستگی و تداوم سرویس درست می باشد . قابلیت اطمینان بالا از سیستمی مورد انتظار است که باید بدون هیچ گونه وقفه ای عمل نماید . به عنوان مثال از یک سیستم کنترل ترافیک خطوط هوایی انتظار می رود که سرویس بدون وقفه ای ارائه دهد . قابلیت اطمینان تابعی از زمان است . بازه زمانی در این پارامتر بستگی به ماهیت سیستم مورد نظر دارد . به طور مثال یک ماهواره باید در مدت زمان تعیین شده ای (طول عمرش) عملکرد درست داشته باشد .

2) قابلیت دسترسی

برخی سیستم ها باید به گونه ای طراحی شوند که بدون وقفه و بدون هر گونه نگهداشتی و به طور پیوسته به عملیات خود ادامه دهند . در بسیاری از موارد نه تنها احتمال خرابی ، بلکه تعداد خرابی ها و به ویژه زمان مورد نیاز برای تعمیر خرابی ها نیز دارای اهمیت بسیار است . برای چنین کاربردهایی قابلیت دسترسی بسیار پراهمیت است به طوریکه توسط آن می توان زمان مورد نیاز عملکرد سیستم را افزایش داد . قابلیت دسترسی A(t) یک سیستم در لحظه t احتمال این است که سیستم در لحظه t به درستی در حال سرویس دهی باشد . A(t) مقدار قابلیت دسترسی متوسط در بازه زمانی t می باشد . این بازه زمانی می تواند طول عمر سیستم یا زمان اتمام یک وظیفه خاص باشد . اگر سیستمی قابل تعمیر نباشد قابلیت دسترسی آن برابر با قابلیت اطمینان سیستم خواهد شد ، یعنی احتمال اینکه سیستم در بازه زمانی صفر تا t دچار خرابی نگردد .

3) ایمنی

از دیدگاه ایمنی ، خرابی ها به دو گونه تقسیم می شوند:

الف – ایمن به خرابی

ب – ناایمن به خرابی

ایمنی به خرابی یعنی اتفاق ناگواری با وقوع این نوع خرابی نمی افتد . ایمنی S(t) یک سیستم ، یعنی احتمال اینکه سیستم به درستی عملکرد خود را انجام دهد یا به دلیل خرابی از نوع ایمن به خرابی ، عملیات خود را قطع کند . ایمنی در سیستم های کنترل انرژی هسته ای و شیمیایی پرکاربرد است . بسیاری از خرابی های غیر ایمن به دلیل اشتباهات بشری است . مثلا در نیروگاه اتمی چرنوبیل انجام آزمایش ها در شرایطی که سیستم های ایمنی به طور دستی خاموش شده بودند ، باعث وقوع فاجعه شد . یک مهندس برق که آشنایی کافی با نیروگاه هسته ای نداشت ، می خواست ببیند که آیا می توان از انرژی رسوب شده برق تولید کرد . در نتیجه برخی سیستم های ایمنی را خاموش کرده بود.

4) قابلیت نگهداشت

نگهداشت به دو دسته اصلی تعمیر (Repair) و اصلاح (Modification) تقسیم می شود . مقوله تعمیر و تحمل پذیری خطا مفاهیمی مرتبط با یکدیگر هستند . تفاوت میان تحمل پذیری خطا و نگهداشت بدین صورت است که در نگهداشت همواره یک عامل خارجی دخالت دارد . مانند یک تعمیرکار ، یک دستگاه آزمون و یا بارگذاری مجدد نرم افزاراز راه دور ، در حالی که تعمیر بخشی از فرآیند رفع نقص در طول فاز مصرف می باشد و پیش بینی نقص معمولاموقعیت های تعمیر را بررسی می کند.

5) امنیت

امنیت وجود همزمان قابلیت دسترسی برای کاربران مجاز ، محرمانگی 10 و یکپارچگی 11 می باشد . امنیت از جمله مواردی است که در حوزه های مختلف تعریف می شود . با این وجود ، امنیت در تمام حوزه های در سه ویژگی اولیه تحت عنوان محرمانگی ، صحت و درستی و قابلیت دسترس پذیری اشتراک دارند .

6) آزمون پذیری

آزمون یا تست ، راهی است که با کمک آن می توان وجود و کیفیت برخی ویژگی ها را در یک سیستم مورد بررسی قرارداد و آزمون پذیری ، توانایی آزمودن برخی خواص در یک سیستم است . آزمون پذیری و قابلیت نگهداری با هم در ارتباط هستند ، چرا که مینیمم کردن زمان لازم برای تشخیص و تعیین موقعیت خطا از مسایل بسیار مهم می باشد.

تهدیدات قابلیت اتکا

عیب(Fault) ، خطا (Error) ، خرابی (Failure) ، مفاهیمی هستند که رابطه علت و معلولی زیر برای آن ها صادق است.

هر کاستی و نقض (عموما فیزیکی) مثل قطع شدن یک سیم یا سوختن دیود عیب تلقی می شود. اگر عیب وضعیت یک مولفه از سیستم را تغییر دهد، اشتباه روی داده است و اگر اشتباه باعث شود که سیستم نتواند کارکرد صحیح خود را ارائه دهد، اشتباه منجر به خرابی شده است.

به عبارت دیگر عیب در حوزه و دنیای فیزیکی (Physical universe) روی می دهد، اشتباه (Error) در حوزه و دنیای اطلاعات (Information universe) و خرابی در حوزه و دنیای خارجی (External universe) روی می دهد.

بطور مثال ایجاد سوراخ در لاستیک چرخ اتومبیل عیب محسوب می شود و اگر سوراخ منجر به پنچر شدن لاستیک شود، خطا اتفاق افتاده و در صورتی که این لاستیک پنچر کارکرد درست اتومبیل را تحت تاثیر قرار دهد، خرابی رخ داده است . (ممکن است لاستیک پنچر، لاستیک زاپاس باشد.)

روش ها و مکانیزم های افزایش قابلیت اتکا

❖ پیشگیری از عیب

به روشهایی اطلاق می شود که سعی دارند از بروز خطا پیشگیری کنند و اکثر آنها مبتنی بر استفاده از روشهای کنترل کیفیت بر ساخت سیستم های سخت افزاری و نرم افزاری می باشند. برخی از این روشها عبارتند از: برنامه سازی ، ساخت یافته و شی گرا، استفاده از متدولوژی در توسعه نرم افزار ،Shielding استفاده از دیواره های آتش (Firewall) .....

❖ رفع عیب

روش های برداشت خطا هم در زمان توسعه و هم در زمان به کارگیری و بهره برداری سیستم انجام می شود. در زمان توسعه سیستم در این روش، سه مرحله بازبینی و ممیزی ، تشخیص و عیب شناسی و تصحیح انجام می شود. در مرحله بازبینی و ممیزی بررسی می شود که آیا سیستم خصوصیات و کارکرد لازم را دارد. مرحله تشخیص و عیب شناسی برای یافتن علت مشکل انجام می شود و سپس خطای کشف شده اصلاح می شود. در زمان بهره برداری سیستم نیز روش های نگهداری سیستم سعی در حذف خطا های گزارش شده دارد و در ضمن همچنان سیستم مورد ارزیابی قرار می گیرد تا خطاهای احتمالی که هنوز ظاهر نشده اند، نیز کشف و برطرف شوند.

❖ تحمل عیب

به روشهایی اطلاق می شود که سعی دارند حتی با بروز خطا، از گسترش آن و خراب شدن سیستم جلوگیری کنند . این روش ها می توانند مبتنی بر کشف اشتباه و ترمیم آن باشند .گونه دیگر این روش ها بر اساس

پوشش دادن خطا عمل می کنند که سعی دارد با افزودن افزونگی به سیستم بدون نیاز به کشف خطا، اثر آن را خنثی نماید.

❖ پیش بینی عیب

در این روش ها سعی می شود رفتار سیستم در مقابل رخ دادن خطاهای مختلف ارزیابی شود .ارزیابی سیستم میتواند به صورت کیفی و یا کمی باشد .روش های احتمالی مانند زنجیره مارکف و یا دیاگرام بلوکی قابلیت اطمینان (RBD) برای ارزیابی استفاده می شود. از نتایج پیش بینی خطا می توان نقاط ضعف سیستم را ارزیابی کرد و سپس به کمک روش های دیگر با آن مقابله کرد.

کاربردهای یک سیستم تحمل پذیر خطا

کاربردهای موجود برای سیستم های تحمل پذیر خطا به چهار دسته عمده تقسیم می شوند : کاربردهای با طول عمرطولانی ، محاسبات بحرانی ، تعمیرات و نگهداری های دشوار و در نهایت کاربردهایی که لازم است بسیار دسترس پذیر باشند . هر یک از این کاربردها ، نیازهای مختلف و در نتیجه تکنیک های متفاوتی را می طلبند .

کاربردهای با طول عمر طولانی

سیستم هایی که از آنها انتظار عمکرد در مدت طولانی داریم . مهم ترین مثال از این نوع کاربرد ، سفینه های فضایی بدون سرنشین است ، مثلا سفینه فضایی**Pionner10** که در سال **1972** به فضا پرتاب شد یا **Mariner** ، **Explorer** یا **Voyager** نمونه های جالبی از این دست هستند . این سفینه ها باید در فضا برای مدت طولانی با صحت و سلامت کامل ، کار می کردند و از آنجا که هزینه طراحی و ساخت و راه اندازی چنین سیستم هایی بسیار بالاست ، عقلانی نیست که خرابی های الکترونیکی در فضاهای دور از دسترس اجازه بروز پیدا کنند . کاربردهای نوعی مشابه ، احتمال متوسط **95** درصد را برای مدت ده سال که طول عمر مفید آنهاست ، پیشنهاد می کنند .

کاربردهای با محاسبات بحرانی

شاید مهمترین کاربرد برای سیستم های تحمل پذیر خطا ، کاربردهایی باشند که در آنها عملیات محاسباتی در سلامتی انسان دخالت داشته باشد ؛ مواردی از قبیل سیستم کنترلی هواپیما ، سیستم های نظامی و برخی کنترلرهای صنعتی . یک کاربرد نوعی از این دست ، دارای قابلیت اطمینانی **0.9** در یک دوره تناوب سه ساعته است که البته بسته به نوعکاربرد می تواند کمی متفاوت باشد . سیستم های کنترل

صنعتی مثل کنترل یک راکتور هسته ای باید تا حدی با دقت انجام پذیرد که هرگز انفجار ناخواسته ای رخ ندهد.

کاربردهایی با تعمیر و نگهداری دشوار

در این نوع کاربردها هزینه نگهداری بسیار بالاست و به سختی انجام می پذیرد . سیستم های پردازشی راه دور یا برخی کاربردهای فضایی از این دسته اند . هدف اصلی استفاده از تحمل پذیری خطا در این گونه سیستم ها آن است که عملیات تعمیر و نگهداری به زمان های بعد که احتمالا امنیت بیشتری وجود خواهد داشت منتقل شود . مثلا هر ماه یکبار پرسنل تعمیرات به سیستم سرکشی خواهند نمود و اگر مشکلی ایجاد شده باشد آنرا برطرف می کنند ، ولی در این میان سیستم باید قادر باشد با خرابی های به وجود آمده کنار بیاید .

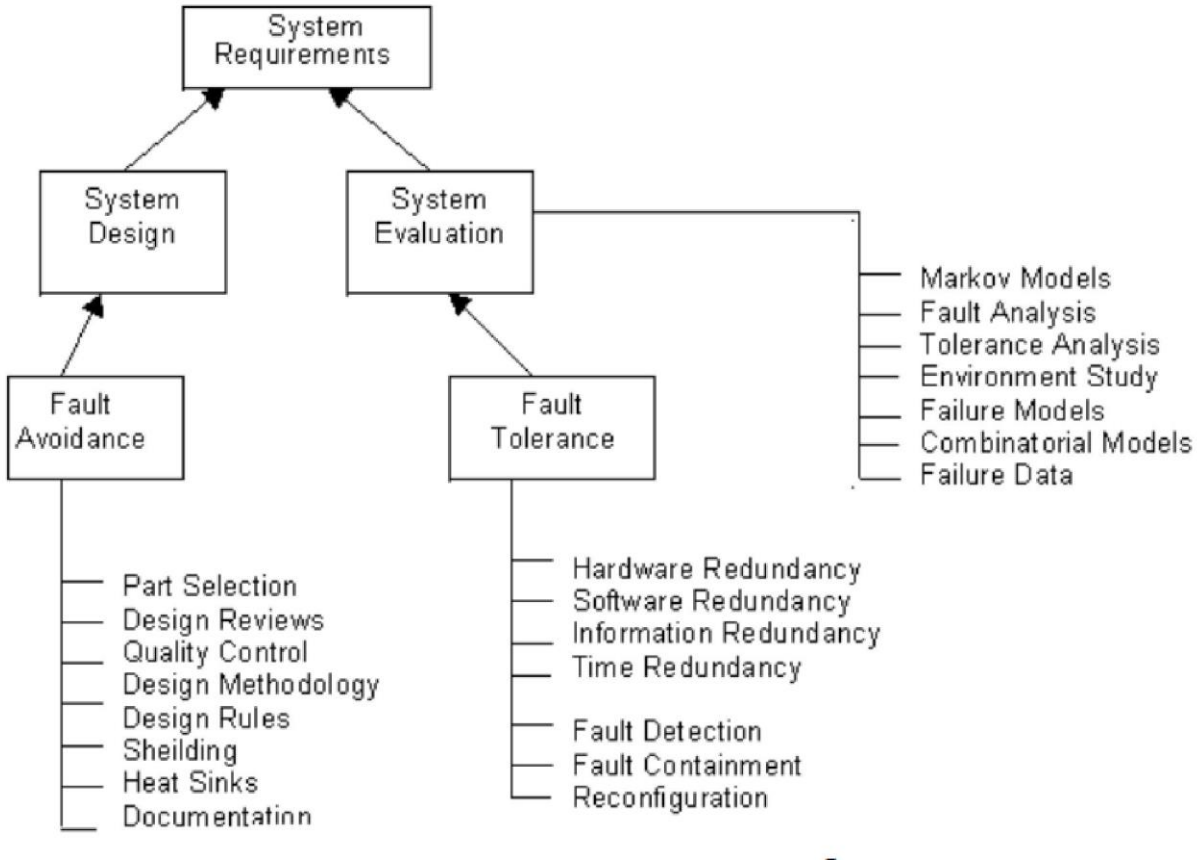
کاربردهای با دسترس پذیری بالا

دسترس پذیری به مرور به یک پارامتر کلیدی در بسیاری از کاربردها تبدیل شده است . بانکداری و دیگر سیستم های اشتراک زمانی ، مثال های مناسبی برای چنین سیستم هایی هستند . کاربران این سیستم ها مایلند که با احتمال بالایی در هر زمان که از این سیستم ها درخواست سرویس نمودند سیستم قادر به دادن سرویس مورد نظر آنها باشد .

تحمل پذیری خطا به عنوان یک هدف طراحی

تحمل پذیری خطا به قابلیتی از سیستم اشاره دارد که سیستم را توانا می سازد تا حتی در صورت بروز خطا در آن بتواند آن خطا را پوشش داده و از خرابی سیستم جلوگیری نماید . این قابلیت در بسیاری از سیستم ها در زمره مهمترین ملزومات غیر وظیفه مندی سیستم است . همان گونه که در شکل زیر قابل ملاحظه است در فرآیند طراحی سیستم نیازهای سیستم از قبیل قابلیت اطمینان به دو طریق به دست آمده است :

الف – طراحی سیستم
ب – ارزیابی سیستم



شکل 2 – نیاز هاي سیستم

طراحی سیستم شامل هر دو تکنیک اجتناب از خطا و تحمل پذیری خطا می باشد . اجتناب از خطا با این هدف انجام می پذیرد که از خرابی های سخت افزاری و خطاهای نرم افزاری در ابتدای طراحی اجتناب نماییم . مثال هایی از این دسته تکنیک ها عبارتند از : انتخاب مولفه های با کیفیت مناسب ، اعمال قواعد طراحی و بررسی مجدد طرح به صورت متناوب . در اصل با اضافه کردن سخت افزار افزونه ، تکنیک های رای گیری و سازماندهی مجدد ، می توان به اهداف تحمل پذیری خطا دست یافت . اگر بخواهیم که فرآیند طراحی موفقیت آمیز باشد ، ارزیابی باید به صورت موازی با فرآیند طراحی صورت پذیرد . برای مثال اگر یک مشکل طراحی پیش از آنکه طرح نهایی برای پیاده سازی تهیه شود ، مشخص گردد ، معمولا به راحتی قابل تغییر است . روش های ارزیابی به ما این امکان را می دهند که مکان هایی از سیستم را که احتمال بروز خرابی در آنها وجود دارد را شناسایی کنیم.

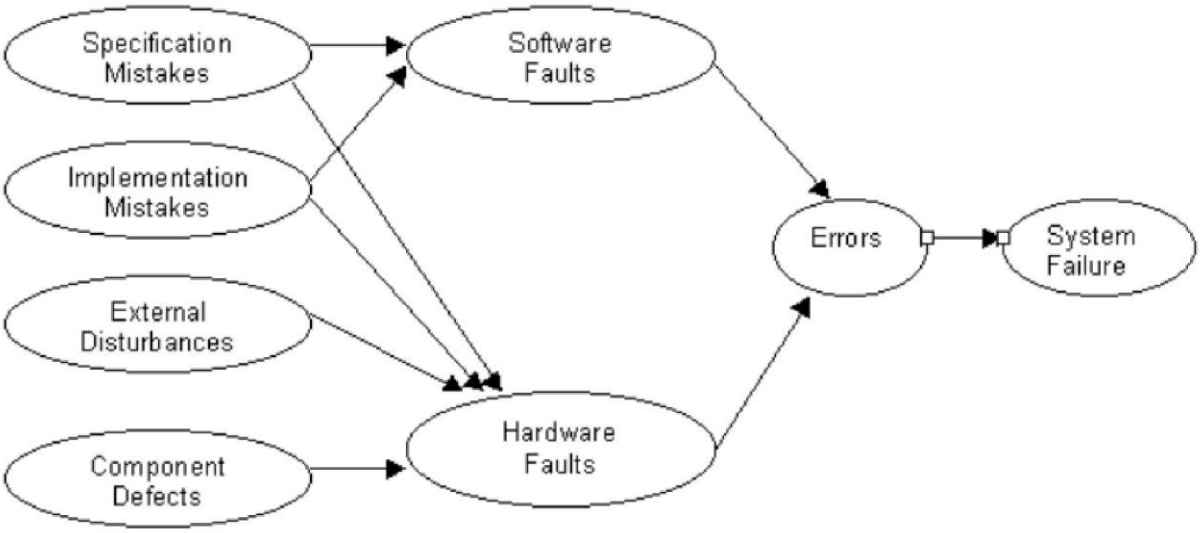
علل بروز خطا

رخ دادن خطا می تواند نتیجه عوامل متعددی باشد که گاه در مرحله طراحی و گاه در فاز پیاده سازی به آنها توجه نشده است . بسیار مهم است که بتوانیم خطا را تشخیص دهیم . یک فرآیند طراحی معمولا با یک جمله شرح مساله آغاز می شود ، سپس طراح براساس درک خودش از مساله ، جزئیات بیشتری به شرح مساله می افزاید . با توجه به شرح مساله یک راه حل سطح بالا ارائه می شود و سپس توسعه الگوریتم ها و

طراحی معماری آغاز می گردد . هنگامی که سخت افزار و نرم افزار تهیه شد ، طراحی باید از یک مرحله آزمایشی به صورت مجتمع عبور نماید به طوریکه در نهایت عملکرد سیستم چنان باشد که شرح اولیه مساله را برآورده سازد . مشکلاتی که در هر یک از فازهای طراحی بروز نماید ، میتواند موجب بروز خطا در سیستم شود . علل خطاهای احتمالی می تواند به هر یک از چهار مورد زیر مرتبط باشد :

- **توصیف اشتباه**
- **پیاده سازی اشتباه**
- **خرابی عناصر**
- **عوامل خارجی**

مورد اول احتمال خرابی به علت اشتباه در توصیف است . این مساله به هر یک از موارد از قبیل طراحی الگوریتم اشتباه ،معماری سخت افزاری و یا نرم افزاری اشتباه بر می گردد. مورد بعدی اشتباهات پیاده سازی است . پیاده سازی در اصل فرآیند تبدیل توصیف سخت افزار یا نرم افزار به سخت افزار واقعی و نرم افزار حقیقی است . به علت ضعف پیاده سازی گاه ممکن است خطاهایی در این مرحله ایجاد شود. علت دیگر ، خرابی عناصر است . ضعف های ساخت و خرابی تصادفی هر یک از اجزاء ، نمونه هایی از این مورد هستند. اجزای فیزیکی به راحتی ممکن است دچار خطا شوند و خرابی هر یک از این عناصر مهمترین علت خطاست. آخرین علت بروز خطا اغتشاشات خارجی محیط هستند . برای مثال تشعشعات و تداخل های الکترومغناطیسی یا اشتباهات اپراتوری و در نهایت عوامل محیطی می توانند موجب خطا شوند . گاهی وجود عدم قطعیت در محیط و نپرداختن صحیح به این مساله می تواند موجب بروز خطا شود . اگر یک مدار الکترونیکی در یک کاربرد نظامی در سرمای بیش از اندازه قرار گیرد ، ممکن است نتایج غیرقابل قبول تولید نماید . شکل زیر عللی که بحث شد را به همراه ارتباط میان آنها نشان می دهد.

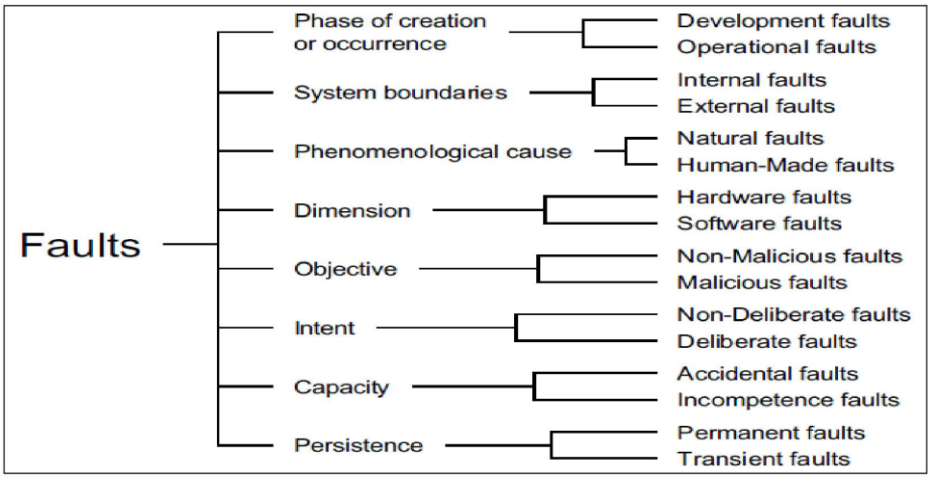


شکل – 3 رابطه علت و معلولی میان عیب ،خطا و خرابی سیستم

همان گونه که از شکل فوق پیداست با بروز یک نقص خواه سخت افزاری یا نرم افزاری ، خطا به وجود می آید . و درصورتی که خطا برطرف نشود ، خرابی سیستم را در پی دارد .

رده بندی خطاها

تمام خطاهایی که در یک سیستم در طول حیاتش ممکن است اتفاق بیافتد در هشت دیدگاه مطابق شکل پایین تقسیم شده است . کلاس های خطا در اینجا به خطاهای مقدماتی و اصلی نام گذاری شده اند.



شکل 4 - تقسیم بندي خطاها

ویژگی های خطا

در این بخش برای مشخص تر شدن بحث در مورد خطا یک سری ویژگی های رفتاری و ذاتی را بیان می نماییم . طبیعت خطا ، نوع خطا را مشخص می کند . به عنوان مثال آیا خطا سخت افزاری است یا نرم افزاری و یا آنالوگ است یا دیجیتال.

طول مدت خطا ، مدت زمانی که خطا در سیستم فعال است را مشخص می کند . اولین نوع خطا ، خطای ماندگار است که اگر عملی در قبال رفع آن انجام نگیرد ، برای مدت طولانی در مدار باقی می ماند . مورد بعدب خطای گذرا است که می تواند در مدت کوتاهی از زمان ظاهر شده و دوباره از بین برود . مورد سوم خطای تناوبی است که در آن هر خرابی می تواند ظاهر شده ، از بین برود و دوباره به صورت متناوب این رخداد تکرار شود. مساله محدوده خطا مشخص می کند که آیا خطا به یک واحد سخت افزاری/ نرم افزاری محدود می شود یا اینکه کل سیستم را فرا می گیرد . مثلا خرابی منبع تغذیه سیستم می تواند به عنوان نمونه ای از خطاهای فراگیر بیان شود.

فلسفه طراحی براي فائق آمدن بر خطا

در حالت کلی سه روش عمده برای فائق آمدن برخطا و نگهداشتن سیستم در کارایی نرمال خود وجود دارد که عبارتند از:

- اجتناب از خطا
- پوشاندن خطا
- تحمل پذیری خطا

اجتناب از خطا ، هر تکنیکی که برای جلوگیری از رخدادن خطا در سیستم انجام شود را در بر می گیرد ، مثل : بررسی مجدد طراحی ، آزمودن و دیگر روش های کنترل کیفی . اگر در بررسی مجدد طرح ، یک مشکل در توصیف نیازهای سیستم مشخص شد ، برطرف می گردد . مشکلات متعددی که می توانستند موجب بروز خطا شوند ، قابل مرتفع نمودن خواهند شد یا با آزمودن یک طرح ، خطاهای بسیاری قابل تشخیص و سپس برطرف نمودن هستند. پوشاندن خطا ، هر فرآیندی که موجب شود پس از بروز خطا ، سیستم لااقل با اشتباه مواجه نگردد را شامل می شود.

تحمل پذیری خطا ، توانایی سیستم است بر ادامه دادن به وظیفه خود پس از اینکه در سیستم با خطا مواجه شدیم. هدف نهایی تحمل پذیری خطا ، جلوگیری از بروز خرابی در سیستم است . تحمل پذیری خطا با اعمال تکنیک های مختلفی قابل ایجاد در سیستم است مثلا پوشاندن خطا یکی از روش های کنار آمدن با خطا در سیستم است و روش دیگر حذف عنصر خراب از کل سیستم می باشد . تغییر ساختار فرآیند ، حذف یک ماهیت خراب از سیستم و باز گرداندن مجدد سیستم به وضعیت کارآمد خود می باشد. اگر تکنیک های تغییر ساختار اعمال شود ، طراح باید به فرآیندهای زیر به خوبی پرداخته باشد:

1: تشخیص خطا : فرآیند شناسایی خطا در سیستم که معمولا پیش از هر فرآیند دیگری باید انجام بگیرد
2: تعیین موقعیت خطا : فرآیند مشخص نمودن این مساله است که در کجا خطا رخ داده است و بنابراین رفع خطای مناسب قابل پیاده سازی است.
3: محدود نمودن حوزه خطا : فرآیند ایزوله نمودن خطا است و جلوگیری از اینکه اثرات آن در کل سیستم انتشار یابد.
4: برطرف نمودن خطا : فرآیند بازگرداندن وضعیت کاری و عملکردی مناسب به سیستم است پس از اینکه به علت خطا ، با مشکل مواجه شده است .

یک سیستم نوعی که تحمل پذیری خطا را پشتیبانی نماید و از تکنیک سازمان دهی مجدد استفاده نماید ، مطابق زیر عمل می کند :

فرض کنید که یک خطا در سیستم رخ داده است و تکنیک های تشخیص خطا ، مشخص می نمایند که خطا در سیستم وجود دارد و روال های تعیین موقعیت خطا ، منبع خطا را تعیین می کنند . رفع خطا وسازماندهی مجدد ، واحد خراب را از سیستم حذف می کنند و آنرا با یک مولفه سالم جایگزین می نمایند و یا با کاهش قابل قبولی در کارایی ، سیستم را به صورت فعال باقی نگه می دارند .

افزونگی به عنوان تکنیکی برای تحمل پذیری خطا

مهم ترین تکنیکی که تا به حال برای تحمل پذیری خطا در سیستم ها به کار رفته است ، استفاده از تکنیک های افزونگی است . در این قسمت ابتدا مفهوم افزونگی و مصادیق افزونگی را در سیستم شرح می دهیم .

افزونگی ، اضافه نمودن اطلاعات ، منابع یا زمانی بیش از آنچه که برای عملکرد حالت عادی سیستم لازم

است به اجزاسیستم است . افزونگی در هر یک از چهار مورد زیر می تواند مصداق پیدا کند :

افزونگی سخت افزاری : اضافه نمودن سخت افزار اضافه ، غالبا به منظور تشخیص خطا یا تحمل پذیری خطا

افزونگی اطلاعاتی : اضافه نمودن اطلاعات بیش از آنچه برای پیاده سازی تابع مورد نظر لازم است . برای مثال کدهای تصحیح خطا یک نوع از افزونگی اطلاعات را به کار می برند .

افزونگی نرم افزاری : اضافه کردن نرم افزار ، بیش از آنچه که برای کارکردهای سیستم لازم است .

افزونگی زمانی : استفاده از زمانی بیش از آنچه توابع سیستم لازم دارند که اجرا شوند ، به طوریکه عملیات تشخیص خطا و تحمل پذیری خطا قابل انجام باشد . حال به توضیح مفصل تر هر یک از موارد افزونگی می پردازیم .

افزونگی سخت افزاری:

تکرار نمودن فیزیکی سخت افزار ، رایج ترین شکل افزونگی است که در سیستم های امروزی هم استفاده می شود . همانطور که مولفه های سخت افزاری روز به روز ارزان تر و قابل استفاده تر می شوند ، اعمال این نوع افزونگی به سیستم آسان تر و عملی تر می شود . در عمل سه نوع افزونگی سخت افزاری وجود دارد :

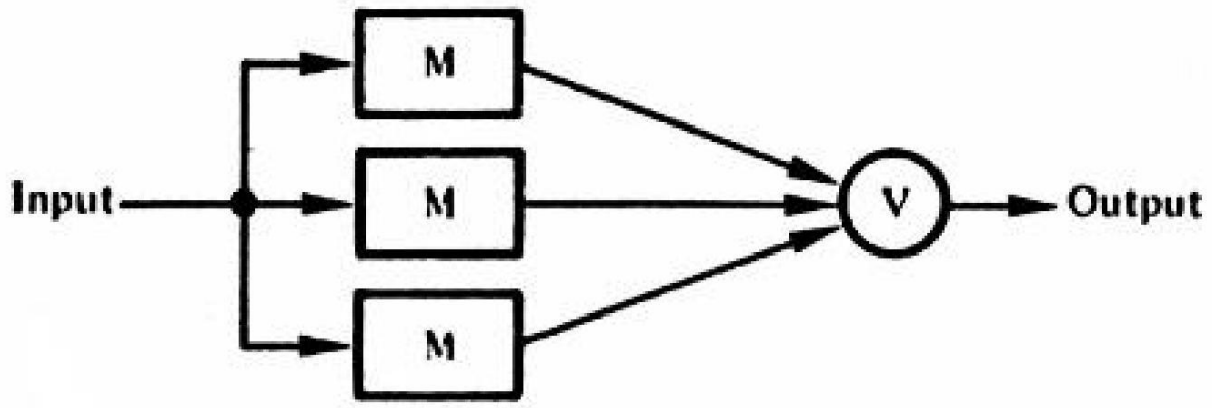
فعال یا پویا **(Active)**

ایستا یا غیرفعال **(Passive)**

ترکیبی **(Hybrid)**

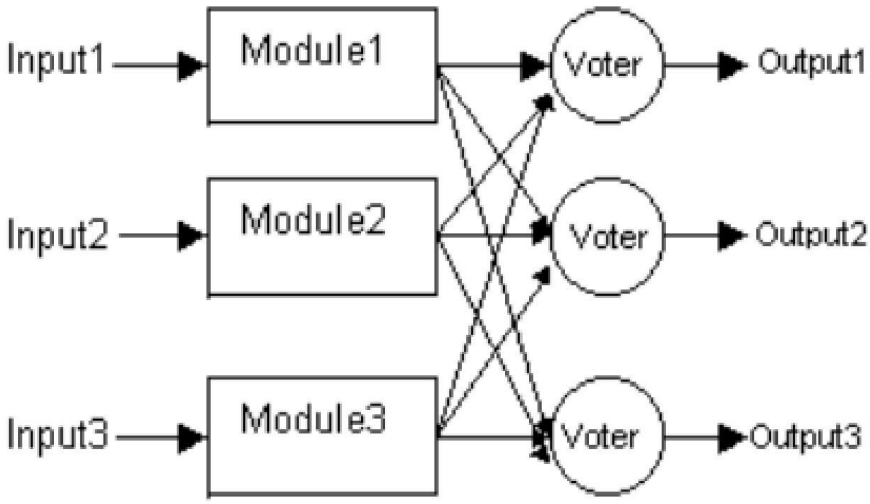
در تکنیک هایی که به نوع افزونگی غیرفعال مجهز هستند ، از مفهوم پوشاندن 35 خطا استفاده می شود که از بروز خرابی در سیستم جلوگیری می کنند . به عبارتی در این روش ها ، بیش از آنکه به تشخیص خطا بها داده شود ، به پوشاندن آن پرداخته می شود . این نوع افزونگی که به آن افزونگی ایستا هم می گویند به مکانیزم رای گیری که از آن برای پوشش رخداد خطا استفاده می شود بسیار وابسته است . اکثر روش های غیرفعال از تکنیک رای گیری براساس رای اکثریت استفاده می کنند . مهم ترین روش برای این نوع از افزونگی ، ساختار **TMR** است . اساس کار این روش بدین گونه است که سخت افزار خود را به صورت

افزونه سه تایی قرار می دهیم و سپس برای تعیین خروجی نهایی از یک مکانیزم رای گیری استفاده نماییم . اگر یکی از ماجول ها با خطایی مواجه شود ، دو ماجول دیگر که بدون خطا هستند ، خروجی نهایی را تولید می کنند .



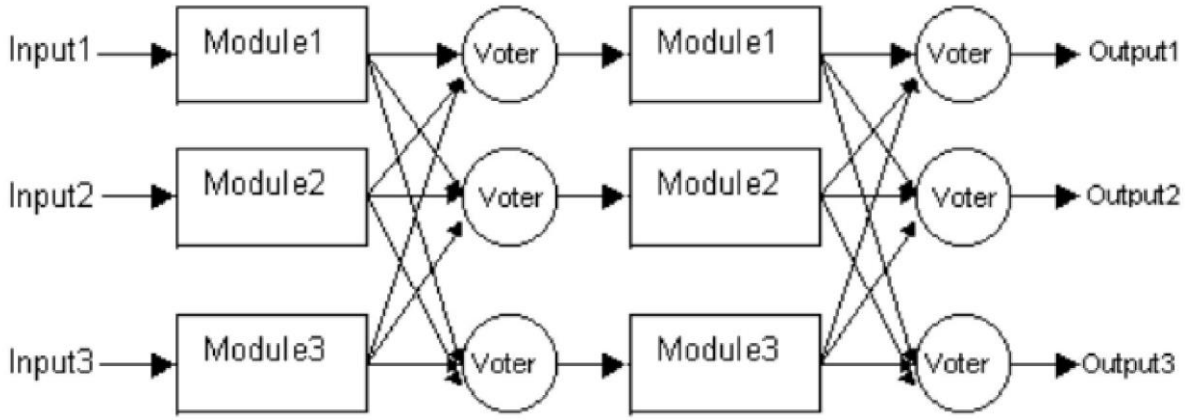
شکل – 5 ساختار TMR

مشکل اساسی این روش هنگامی است که داور با مشکل مواجه شود که در آن صورت کل سیستم از کار خواهد افتاد . به عبارت دیگر ، قابلیت این نوع از TMR به هیچ وجه بهتر از قابلیت اطمینان داور به تنهایی نیست . هر مولفه منفردی که خرابی آن موجب خرابی کل سیستم می شود ، نقطه منحصربفرد خرابی نامیده می شود . یک روش برای بهبود این مساله ، استفاده از افزونگی در خود داور می باشد که در شکل زیر قابل مشاهده است.



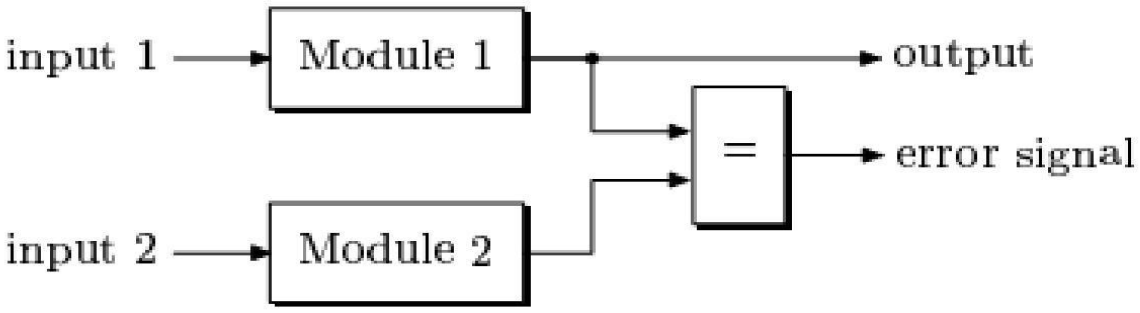
شکل 6 – TMR

در صورتیکه پس از هر مرحله در سیستم از یک داور افزونه استفاده شود و خطای هر مرحله پیش از انتقال به لایه بعد ، تصحیح شود TMR با چند گام حاصل خواهد شد.



شکل 7 – TMR با چند گام

در حالت کلی به جای TMR می توان از NMR استفاده نمود . مهم ترین مصالحه در NMR درجه تحمل پذیری خطای بدست آمده در مقایسه با میزان سخت افزار لازم است. در روش فعال که غالبا روش دینامیک هم نامیده می شوند ، تحمل پذیری خطا به وسیله تشخیص وجود خطا و انجام عملی در قبال آن صورت می پذیرد. به عنوان مثال حذف عامل خرابی و تغییر ساختار سیستم به نحوی که مجددا به مد عملیاتی بازگردد. افزونگی سخت افزاری فعال از تشخیص خطا و تعیین موقعیت آن و رفع خطا استفاده میکند تا به تحمل پذیری خطا نائل آید . این گونه افزونگی در کاربردهایی بیشتر مورد توجه است که می توانیم سیستم را با تغییر ساختار در یک مدت زمان قابل قبول دوباره به حالت عملیاتی مناسب باز گردانیم . از ساده ترین انواع این نوع افزونگی می توان به تکنیک سخت افزار مضاعف به همراه مقایسه اشاره کرد.

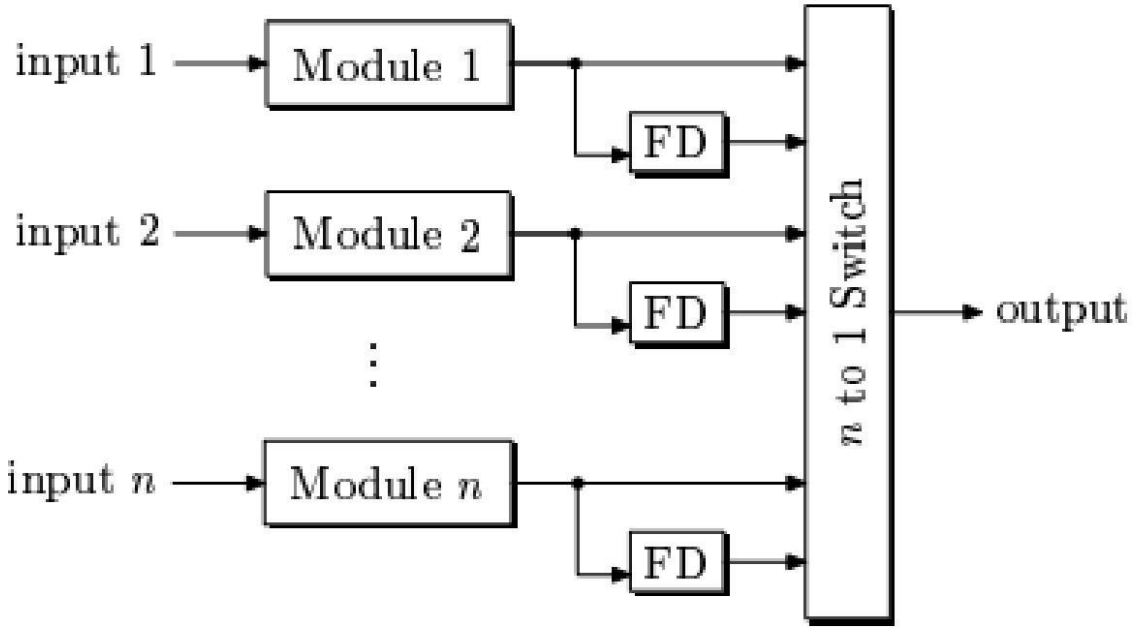


شکل 8 - Duplication with Comparison

در این تکنیک دو ماجول مشابه به صورت موازی ، عمل یکسانی را انجام می دهند و سپس با انجام یک مقایسه در صورت بروز تفاوت ، پیام خطایی صادر می شود که البته در این نسخه ساده شده ، معلوم نخواهد شد که کدام عنصر با خطا مواجه شده است . یکی دیگر از مشکلات این روش آنست که اگر هر دو مولفه به علت مواجهه با خطا ، جواب یکسانی را تولید نمایند ، به علت عدم وجود تفاوت ، پیام خطایی صادر نخواهد شد در صورتیکه در اصل خطایی بروز کرده است .

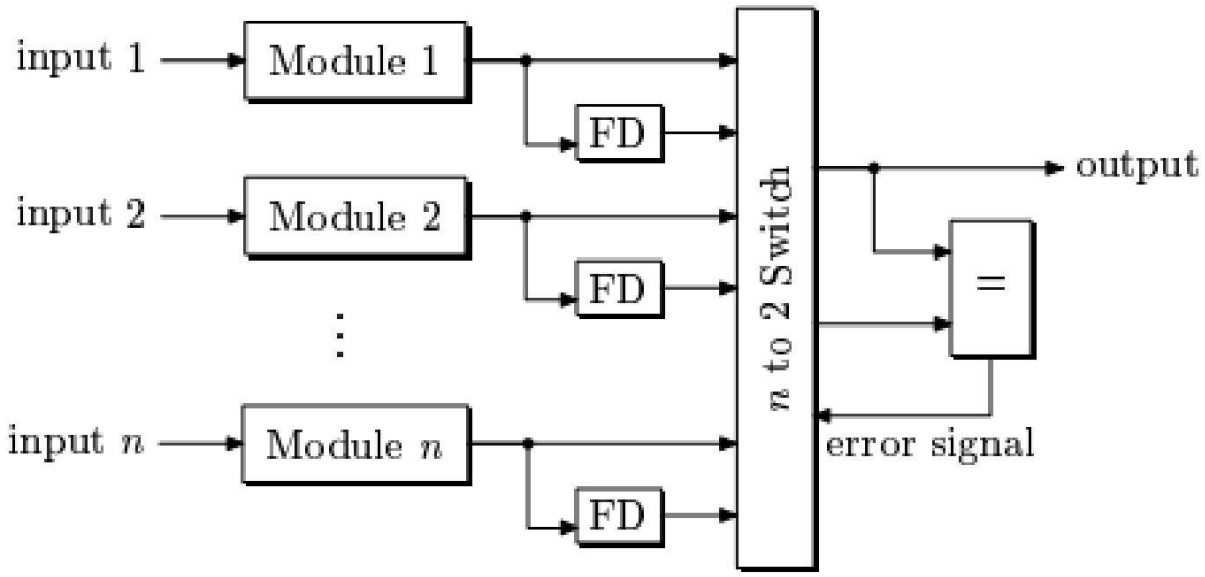
روش دیگری برای اعمال این نوع افزونگی ، سخت افزار اضافه و آماده باش است . در این روش یک ماجول اصلی و فعال است و دومی به عنوان پشتیبانی برای اولی قرار گرفته است . وقتی که بروز خطایی مشخص شد ، ماجول اصلی با ماجول پشتیبان جایگزین می شود که در اینجا فرآیند تغییر ساختار به شکل سوییچ نمودن میان این دو انجام می پذیرد .

در صورتی که بخواهیم این زمان سوییچ کوتاه باشد باید از آماده باش داغ استفاده نماییم که در آن ماجول افزونه به صورت همزمان با ماجول اصلی ، وضعیت های سیستم را ردیابی می کند و در هر لحظه آماده جایگزینی است . در حالیکه در نوع آماده باش سرد ، تا زمانی که نیازی نباشد ماجول افزونه خاموش است و بنابراین جایگزینی با آن صرف زمان بیشتری می طلبد .



شکل 9 - Standby Sparing

روش بعدی زوج- و یک واحد اضافه است که در این روش ویژگی های هر دو روش قبلی را با هم آمیخته است . در این روش دو مولفه آماده باش استفاده می شود و در عین حال دو ماجول هم به صورت موازی در زمان با هم فعالند و نتایج آنها مقایسه شده ، خاصیت تشخیص خطا فراهم می شود . سیگنال خطای حاصل از مقایسه ، برای آغاز نمودن روال فرآیند تغییر ساختار که ماجول های خراب را حذف می کند و با واحد اضافه جایگزین می نماید ، به کار می رود .



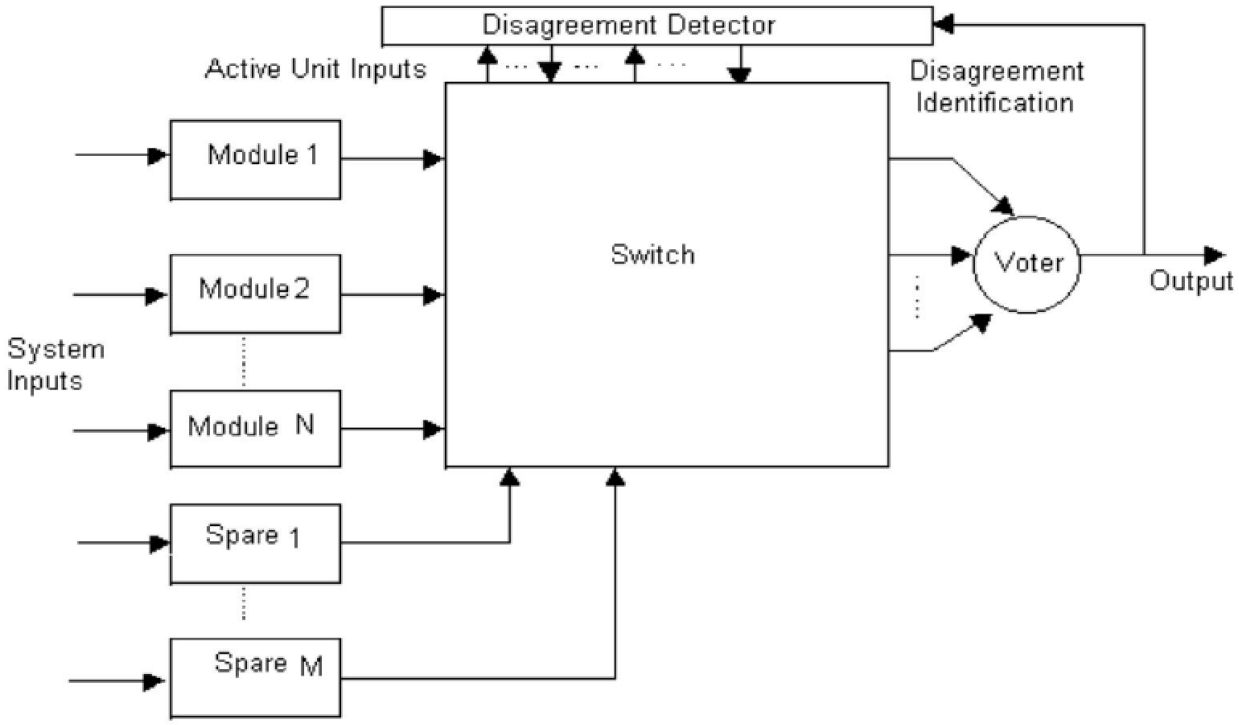
شکل 10 - Pair&Spare

تکنیک آخر استفاده از **Watchdog Timer** است که برای تشخیص خطا در سیستم های بسیاری مورد استفاده قرار گرفته است . مفهوم این تکنیک آنست که از عدم رخداد یک اتفاق ، نتیجه می گیریم که در سیستم خطا رخ داده است . این تایمر باید به صورت پریودیک ریست شود . هر خرابی در سیستم که باعث شود این عمل انجام نگیرد ، موجب خواهد شد سیستم خاموش شده و به این ترتیب دیگر خرابی عمده رخ ندهد . فرض اساسی در این تکنیک این است که سلامتی سیستم موجب می شود که این تایمر به صورت متناوب ریست شود.

افزونگی سخت افزاری ترکیبی

در روش های ترکیبی که ویژگی های جالب هر دو روش فوق را ترکیب می کنند و در واقع رایج ترین نوع افزونگی سخت افزاری هستند ،و البته این روش ها بسیار پرهزینه است و بنابراین در کاربردهایی به کار گرفته می شود که تحمل پذیری خطا بسیار ضروری می باشد . یکی از مهم ترین تکنیک های این روش

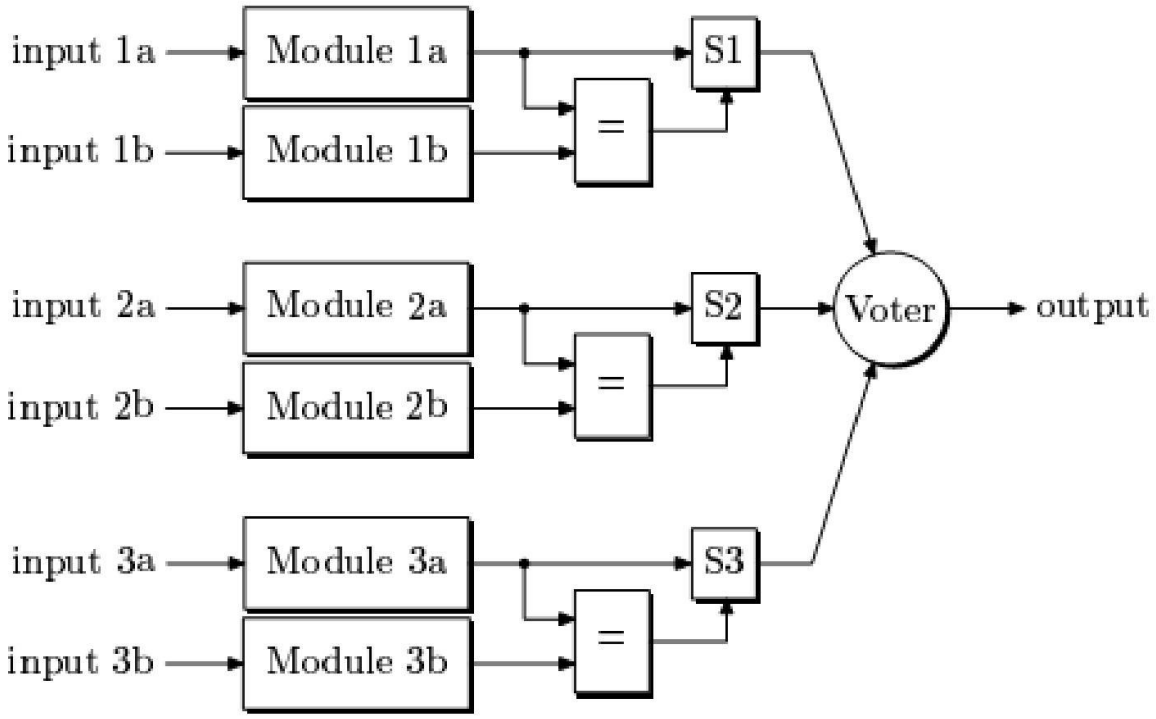
NMR with Spares است . این روش هر دو ایده NMR و Standby Sparing را با هم در می آمیزد .شکل زیر بیانگر همین ایده می باشد.



شکل 11- NMR with Spares

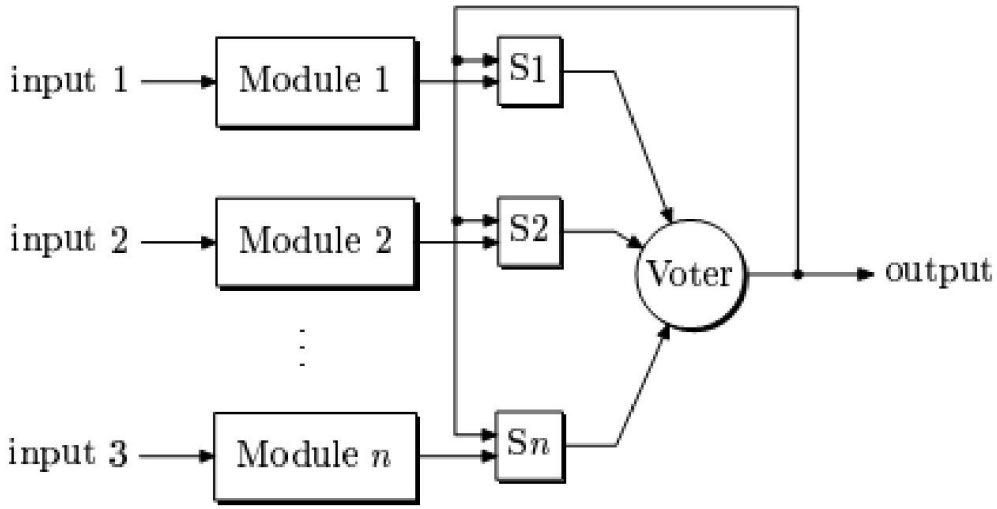
در هر دوره زمانی بخش تصمیم گیر مدار خروجی داور را با خروجی تک تک ماجول ها مقایسه می کند . اگر تفاوتی مابین خروجی داور و خروجی ماجول بود ، آن ماجول معیوب تلقی می شود و با یک ماجول رزرو جایگزین می گردد.

روش بعدی معماری Triplex – Duplex است که دو ایده TMR و Duplication With Compariso را باهم ترکیب می کند . استفاده از TMR این امکان را فراهم میکند که خطا پوشانده شود ، در حالیکه استفاده از سخت افزار مضاعف به همراه مقایسه کننده موجب می شود که خطاها تشخیص داده شود و ماجول معیوب از فرآیند رای گیری حذف گردد.



شکل 12 - Triple – Duplex

یکی دیگر از روش های افزونگی هیبریدی افزونگی Self-Purging می باشد . در این روش همان گونه که از شکل پیداست ، هر یک از ماجول ها دارای یک سوییچ هستند که با مقایسه خروجی داور با خروجی ماجول در صورت اینکه توافقی وجود نداشته باشد ، آن ماجول را از سیستم حذف می کند . قابل توجه اینکه این ساختار نیازمند داوری به صورت Threshold Gate می باشد .



شکل 13 - Self Purging Redundancy

افزونگی اطلاعات

افزونگی اطلاعاتی به اضافه نمودن هر مقدار اطلاعات ، بیشتر از داده های اصلی مساله اتلاق می شود به طوری که تشخیص خطا و گاه تحمل پذیری خرابی انجام پذیر باشد . مثال خوبی از این نوع افزونگی ، کدهای تشخیص خطا و رفع خطاست که به داده های اصلی اضافه می شوند. یک تکنیک اساسی که در هر دو موضوع تشخیص خطا و نیز تصحیح خطا استفاده می شود ، کد Hamming و در اصل استفاده از فاصله همینگ است . از انواع کدهای مورد استفاده در این نوع افزونگی می توان به موارد زیر اشاده نمود:

- Parity Code
- M of N Codes
- Checksum
- Cyclic Codes
- Residue Codes
- Berger Codes

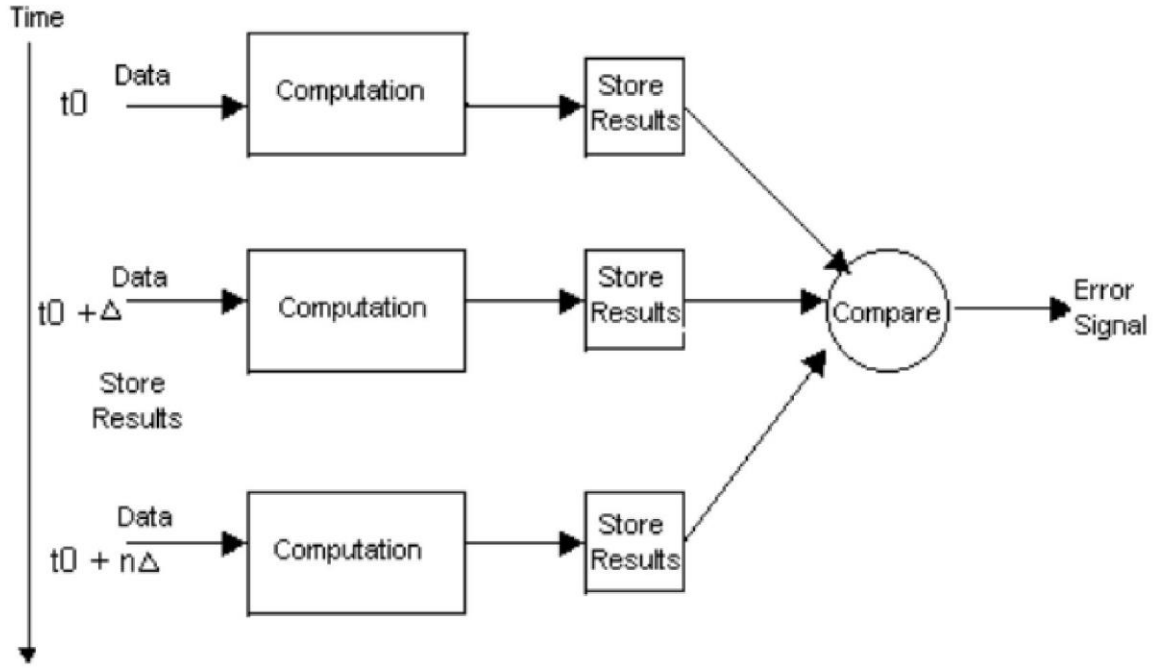
اگر چه سربار زمانی روش های افزونگی اطلاعات نسبت به روش تکرار زمانی وظایف(افزونگی زمانی) ، کمتر است ، اما این روش ها معمولا کلی نیست و برای هر وظیفه باید روش کشف خطای مخصوصی را بکار برد. به عنوان مثال ، برای آرایه ای از رشته ها می توان از کدهای توازن و یا جمع وارسی ، برای بررسی مقدار داده ها و نیز از بررسی مرتب بودن خروجی ، برای صحت عملکرد مرتب سازی استفاده کرد . در حالیکه برای بررسی صحت عملکرد ضرب ماتریس ، استفاده از کدهای حسابی مناسب تر است و برای بسیاری از وظایف ، روش بررسی روند کنترل مناسب تر می باشد.

افزونگی زمانی

مهمترین مشکل استفاده از افزونگی هایی که تا به حال شرح داده شدند نیاز آنها به مقدار زیادی سخت افزار و بیت های اطلاعاتی در تکنیک های مختلف بود . اگر بخواهیم هر دو نوع منابع مورد نیاز یعنی داده و سخت افزار را کاهش دهیم ،استفاده از زمان بیشتر ، بهایی است که باید بپردازیم. در بسیاری از کاربردها ، زمان به اندازه سخت افزار ارزشمند نیست چرا که برای سخت افزار باید به مساله هزینه ،سایز ، توان و وزن دستگاه ها توجه نمود . انتخاب نوع افزونگی لازم ، بستگی مستقیمی به نوع کاربرد دارد.

تشخیص خطای گذرا

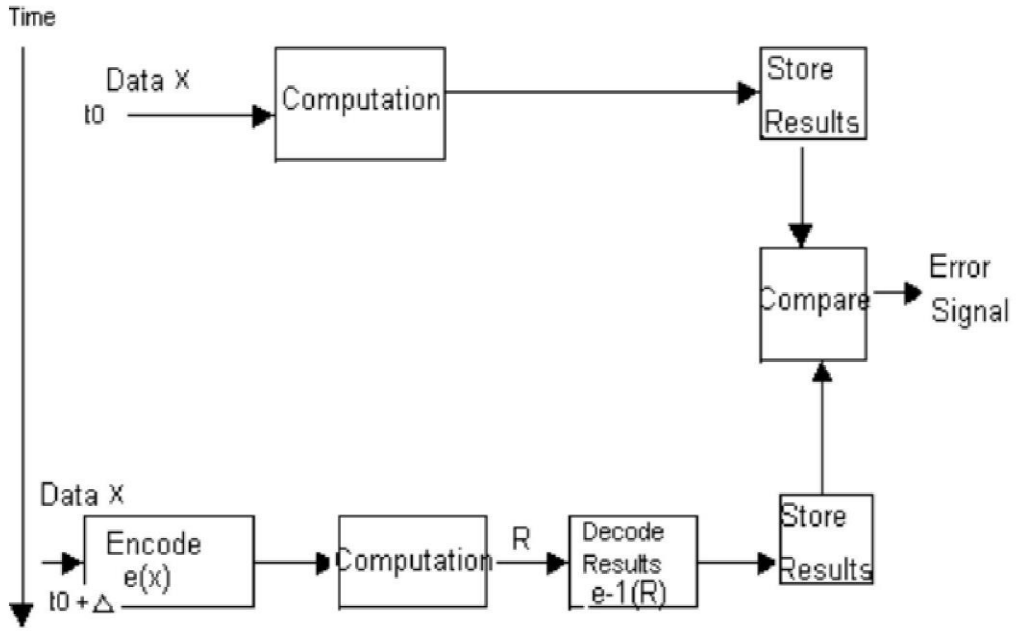
مفهوم اصلی در افزونگی زمانی ، تکرار محاسبات در طول زمان است به نحوی که وجود خطا را بتوان تشخیص داد ، یعنی اگر یک کار را در طول زمان چندبار انجام دهیم و سپس نتیجه را با هم مقایسه کنیم ، می توانیم در صورت بروز تفاوت ، به وجود خطا پی ببریم و در این صورت محاسبات را تکرار کنیم تا بفهمیم که آیا خطا هنوز وجود دارد یا از بین رفته است . شکل زیر این ایده را نشان می دهد .



شکل - افزونگی زمانی و تکرار محاسبات و مقایسه نتایج

تشخیص خطای ماندگار

یکی از مهم ترین فواید استفاده از افزونگی زمانی ، توانایی این روش در تشخیص خطاهای ماندگار است . مفهوم اصلی مورد استفاده در این روش در شکل زیر نشان داده شده است.



شکل 15 – تشخیص خطای ماندگار

در اولین مرحله محاسبه یا انتقال ، اپرندها همانطور که نمایش داده شده اند مورد استفاده قرار گرفته و نتایج در یک رجیستر ذخیره می شوند . قبل از انجام محاسبات بعدی یا انتقال ، اپرند ها رمز می شوند) با

استفاده از تابع رمز . (بعد از آنکه عملیات روی داده های رمز شده ، انجام گرفت ، نتایج رمزگشایی می شوند و با نتیجه فاز اول مقایسه می شود . دراصل تابع رمزکننده باید طوری انتخاب شود که عمل تشخیص خطا ممکن باشد.

تحمل پذیری خطای نرم افزاری

برای بسیاری از سیستم ها ، عملیات نرم افزاری امن و قابل اعتماد یک نیاز مهم به شمار می رود ، مانند کاربردهای هوافضایی ، کنترل ترافیک هوایی ، تجهیزات پزشکی ، هسته ای ، بانکداری الکترونیکی . هزینه و نتیجه خرابی این سیستم ها می تواند در گستره وسیعی از صدمات انسانی ، مالی و غیره مطرح شود . از آنجایی که امروزه نرم افزار نقش اصلی را در انجام وظایف سیستم ها برعهده دارد ، بنابراین اهمیت ویژه ای در میزان قابلیت اطمینان سیستم ها دارد . لذا برای افزایش قابلیت اطمینان ، لازم است سیستم ها به صورت تحمل پذیر در برابر خطا طراحی شوند . تحمل پذیری خطای نرم افزاری معمولا به وسیله افزونگی و گوناگونی به دست می آید که پیچیدگی اضافه ای را به طراحی نرم افزار وارد می کند . دلیل این امر آنست که تحمل پذیری خطا جزو دغدغه های مداخله ای است که در هنگام پیاده سازی سایر واحدهای نرم افزار را تحت تاثیر قرار می دهد .

پیچیدگی باعث ایجاد نقص های نرم افزاری در سیستم های کامپیوتری می شود . تحمل پذیری خطای نرم افزاری به دلیل اینکه ما در تولید نرم افزارهای بدون اشتباه ناتوان هستیم ، مورد نیاز است . با وجود اعمال روش هایی برای کنترل و کاهش نقص های نرم افزاری ، همچنان نقص هایی در نرم افزار نهفته می ماند و در عمل خود را نشان می دهند . تنها روش مقابله با این نقص ها و عوارض اشی از آن استفاده از تاکتیک های تحمل پذیری خطای نرم افزاری است . این تاکتیک ها روی مولفه هایی اعمال می شوند که نقص های طراحی آنها ناشی از پیچیدگی شان است . با افزایش پیچیدگی نرم افزار احتمال به وجود آمدن نقص های طراحی نرم افزار بیشتر می شود . با اینکه استفاده از روش های پیشگیری مانند روش های مهندسی نرم افزار و آزمون نرم افزار در کاهش نقص های نرم افزاری موثر است اما همواره نقص هایی در نرم افزار نهفته می ماند که به هنگام عملیاتی شدن نرم افزار بروز می کنند . تحمل پذیری خطای نرم افزاری می تواند در لایه های مختلف نرم افزار یا ساختارهای نرم افزاری از قبیل سیستم عامل ، برنامه های کاربردی ، فرآیندها ، اشیاء ، توابع و متدها فراهم شود .

تحمل پذیری خطاهای نرم افزاری می تواند به دو گروه تقسیم شوند :

الف – نرم افزارهای تک نسخه ای

ب – نرم افزارهای چند نسخه ای

تکنیک های تک نسخه ای با پیاده سازی یک نسخه از نرم افزار خطاهای نرم افزاری را تحمل می کنند . تکنیک های تک نسخه ای معمولا از تکنیک های بازگشت به عقب ، بازگشت به جلو و همچنین از افزونگی زمان و افزونگی اطلاعات استفاده می کنند . مثال هایی از تکنیک های تک نسخه ای شامل کشف خطا ،

checkpoint و شروع مجدد می باشد. تکنیک های چند نسخه ای ، دو یا تعداد بیشتری از نسخه های نرم افزار را به سریالی یا همروند اجرا می کنند . این نسخه ها به وسیله تنوع طراحی از قبیل تیم های برنامه نویسی مختلف یا الگوریتم های مختلف با هدف جلوگیری از خطاهای طراحی ایجاد می شوند . تکنیک های چند نسخه ای معمولا خیلی پرهزینه هستند لذا آنها بیشتر در سیستم های ایمنی – بحرانی از قبیل سیستم های کنترل پرواز ایرباس A380 ، سیستم های حمل و نقل ، سیستم های ، فضایی مانند شاتل فضایی ناسا و غیره استفاده می شوند .

تحمل پذیری خطای نرم افزاری تک نسخه ای

این تکنیک ها به یک واحد نرم افزاری قابلیت ها و توانایی هایی را اضافه می کنند که در یک محیط عاری از نقص غیرضروری است و سیستم را در مقابل نقص های نرم افزاری بعد از مرحله طراحی و ساخت ، تحمل پذیر خطا می کنند این تکنیک ها با رخداد یک نقص ، مکانیزم هایی را به نرم افزار ارائه می دهند که از رخداد هر گونه خرابی در سیستم و انتشار آن جلوگیری می کنند . تکنیک های تحمل پذیری خطای نرم افزاری تک نسخه ای به سه گروه زیر تقسیم می شوند:

- آشکارسازی نقص (Fault Detection)
- محدود سازی نقص(Fault Containment)
- بازیابی نقص (Fault Recovery)

آشکارسازی نقص

هدف از آشکارسازی نقص در نرم افزار تشخیص رخداد نقص در سیستم است . تکنیک های تک نسخه ای معمولا انواع مختلفی از آزمون های پذیرش را به منظور آشکارسازی نقص به کار می برند . خروجی نرم افزار در معرض آزمون پذیرش قرار می گیرد ، اگر نتیجه آزمون موفقیت آمیز بود برنامه به اجرای خود ادامه می دهد . در غیر اینصورت آن خروجی به عنوان رخداد نقص شناخته می شود . تکنیک های موجود شامل بررسی های زمانی ، بررسی های کد ، بررسی های معکوس ، بررسی های منطقی و بررسی های ساختاری می باشد .

کاربرد بررسی های زمانی در سیستم هایی است که مشخصات آنها شامل محدودیت های زمانی می باشد . بر مبنای این محدودیت های زمانی ،فرآیند بررسی کردن ، انحراف از رفتار موردنیاز سیستم را نشان می دهد . بطور مثال ، تایمرنگهبان نمونه ای از بررسی های زمانی است . تایمر نگهبان به منظور نمایش کارایی یک سیستم و آشکارسازی نقص های آن سیستم به کار می رود . بررسی های کدینگ در سیستم هایی استفاده می شود که اطلاعات در آن سیستم ها با استفاده از تکنیک های افزونگی اطلاعات رمزنگاری

می شوند . مثلا کدهای محاسباتی به منظور آشکارسازی خطاها در عملیات محاسباتی مورد استفاده قرار می گیرند . این بررسی ها در نمایش اطلاعات ، از افزونگی استفاده میکنند.

آشکارسازی خطا بر مبنای بررسی های میان اطلاعات اصلی و افزونه قبل و بعد از عملیات می باشد . به طور مشابه ، بسیاری از تکنیک های سخت افزاری در نرم افزار هم استفاده می شوند . بسیاری از عملیات حسابی ، بعضی از ویژگیهای خاص میان اطلاعات اصلی و افزونه را حفظ می کنند ، بنابراین می توانند استفاده از این نوع بررسی را برای آشکارسازی خطاهایشان در هنگام اجرا فعال کنند . در برخی سیستم ها ممکن است که مقادیر خروجی معکوس شود و مقادیر ورودی متناظر محاسبه گردد . برای چنین سیستم هایی بررسی های معکوس می تواند اعمال گردد . این روش ورودی های واقعی سیستم را با ورودی های محاسبه شده مقایسه می کنند . نقص در صورت عدم تطابق آشکار می شود . بررسی های منطقی ، ویژگی های مفهومی داده را برای آشکارسازی نقص به کار می برند (از قبیل محدوده داده ، نرخ تغییر و ترتیب) . این ویژگی ها بر مبنای نیازمندی ها یا طراحی ویژه یک ماجول می باشند . بررسی های ساختاری از ویژگی های ساختارهای داده استفاده می کنند . به طور مثال برای تعدادی از عنصرهای یک ساختار می توان لیست ها ، صف ها و درخت ها ، پیوندها و اشاره گرها را بررسی و صحت سنجی نمود . با اضافه کردن داده افزونه به یک ساختار داده می توان بررسی ساختاری را اعمال نمود . مثلا افزودن شمارنده روی تعداد عناصر موجود در یک لیست یا افزودن اشاره گر های اضافی.

محدود سازی نقص

محدود سازی نقص در نرم افزار با اصلاح ساختار سیستم و اعمال محدودیت هایی در سیستم بدست می آید . برای محدودسازی نقص چهار تکنیک وجود دارد که عبارتند از : پیمانه بندی ، تفکیک سازی ، محصورسازی سیستم و عملیات اتمیک .

معمولا یک سیستم نرم افزاری به واحدهایی تجزیه می گردد که یا وابستگی بین آنها وجود ندارد و یا وابستگی کمیوجود دارد . مقوله پیمانه بندی توسط محدودسازی ارتباط میان واحدها از انتشار نقص ها جلوگیری می کند جداسازی بین واحدهای مستقل از هم توسط تفکیک سازی ساختار نرم افزار به صورت ابعاد افقی و عمودی انجام می گیرد. تفکیک به صورت افقی ، عملکرد اصلی نرم افزار را به شاخه های مستقل از هم مجزا می کند . اجرای عملیات و ارتباط میان واحدها توسط واحدهای کنترلی صورت می گیرد . تفکیک به صورت عمومدی عملیات پردازشی و کنترلی را در یک ترتیب بالا به پایین توزیع می کند واحدهای سطح بالا معمولا روی عملیات کنترلی متمرکز شده و واحدهای سطح پایین فرآیند پردازش را اجرا می کنند . تکنیک محصورسازی سیستم بر مبنای این اصل است که هیچ عملیاتی مجاز نیست مگر آنکه صریحا اجازه داده شود . در یک محیط با محدودیت های زیاد و کنترل شدید ، تمامی تعاملات بین اجزای سیستم قابل رویت است بنابراین مکان یابی نقص و از بین بردن آن به راحتی صورت می گیرد .

بازیابی نقص

زمانیکه نقصی آشکار می شود و در سیستم محدود می گردد ، سیستم سعی می کند خود را از وضعیت معیوب بازیابی کند و مجددا خود را به وضعیت عملیاتی اولیه باز گرداند . تکنیک های بازیابی وضعیت سیستم را زمانی که خطایی در برنامه رخ می دهد به یک وضعیت درست و عاری از خطا تبدیل می کنند و به دو روش بازگشت به عقب 58 و بازیابی به جلو این کار را انجام می دهند .

در بازیابی به عقب زمانی ، فرآیند عقب کشیدن سیستم به وضعیت ذخیره شده قبلی صورت می گیرد . معمولا فرض می شود که وضعیت ذخیره شده قبلی ، قبل از آنکه نقص خود را نشان دهد صورت می پذیرد . یعنی وضعیت قبلی عاری از خطا می باشد . اگر وضعیت قبلی عاری از خطا نباشد همان خطا باعث رخداد مشکلاتی در فرآیند بازیابی می شود .

حالت های سیستم در نقاط بازیابی از قبل تعیین شده ای ذخیره می شوند . ضبط یا ذخیره این وضعیت قبلی را نقاط بررسی شده می نامند . این حالت باید در حافظه پایداری وارسی شود که تحت تاثیر هیچ گونه خرابی قرار نگیرد . روش بازیابی به عقب کاربردی ترین تکنیک بازیابی برای تحمل پذیری خطای نرم افزاری می باشد . در بازیابی رو به جلو این وضعیت با پیدا کردن یک وضعیت جدید که از آنجا سیستم می تواند به عملیات خود ادامه دهد

، صورت می پذیرد . این وضعیت می تواند یک مد تنزل یافته و عاری از نقص قبلی باشد . در این نوع بازیابی از جبران خطا استفاده می شود . جبران خطا بر مبنای یک الگوریتمی است که از افزونگی استفاده می کند . همچنین برای نقاط Check Pointing باید آزمون های سازگاری و توانایی انجام پذیرند .

آزمون سازگاری

یک آزمون سازگاری از دانش اولیه ای در مورد ویژگی های اطلاعاتی برای بررسی صحت اطلاعات استفاده می کند . به عنوان مثال در برخی کاربردها ، از ابتدا می دانیم که بزرگی یک سیگنال از یک مقدار معلومی نباید بیشتر باشد . چنین آزمون هایی با وجود اینکه در سخت افزار هم به راحتی قابل پیاده سازی است ، معمولا به صورت نرم افزاری پیاده می شوند . مورد دیگری که برای آزمون سازگاری می توان مثال زد و استفاده در کاربردهای کنترلی دارد ، مقایسه مقدار اندازه گیری شده کارایی با چند مقدار پیش بینی شده است .

آزمون توانایی

آزمون های توانایی به این منظور انجام می گیرند که مشخص نمایند ، یک سیستم توانایی لازم را دارد یا خیر . برای مثال ممکن است مایل باشیم بدانیم آیا کل حافظه ، قابل دسترسی است و یا همه پردازنده های سیستم به درستی کار می کنند یا خیر ؟ مدل های مختلفی برای آزمون توانایی وجود دارد . اولین آنها آزمون ساده حافظه است . مثلا یک پردازنده می تواند یک الگوی مشخص را در مکان معلومی از حافظه بنویسد و آن موقعیت را بخواند تا مطمئن شود که آیا آن داده به درستی ذخیره و بازیابی شده است یا خیر

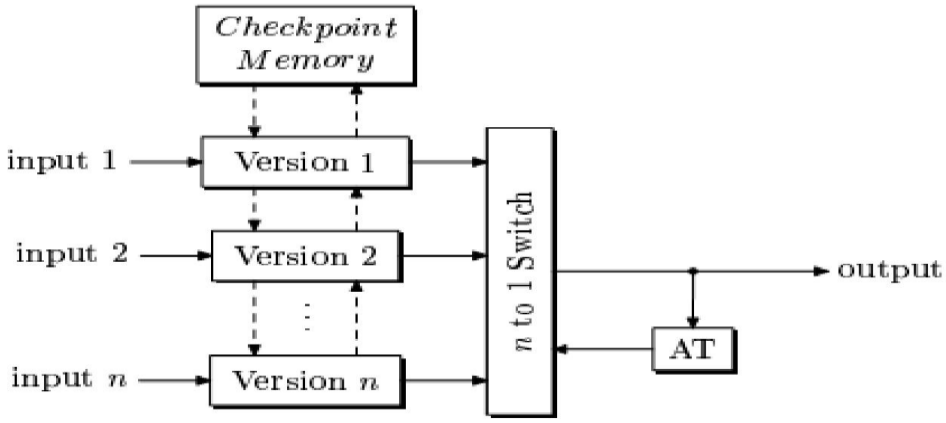
شکل دیگری برای آزمون توانایی ، به این منظور است که مثلا بررسی کنیم که آیا کلیه پردازنده های یک سیستم چند پردازنده ای ، قادرند به درستی با هم ارتباط برقرار نمایند یا خیر . برای اینکار می توان به صورت متناوب ، اطلاعات مشخصی را از یک پردازنده به دیگری منتقل نمود . برای مثال هر پردازنده می تواند یک بیت خاص از یک حافظه اشتراکی را SET نماید تا قابلیت ارتباط او با حافظه و در نتیجه با بقیه مشخص شود .

تکنیک های تحمل پذیری خطای نرم افزاری چند نسخه ای

تکنیک های چند نسخه ای ، دو یا چند نسخه از یک واحد نرم افزاری را به کار می برند که دارای تنوع در طراحی هستند . به عنوان مثال به کارگیری تیم های مختلف کاری ، زبان های کدینگ مختلف یا الگوریتم های متفاوت باعث می شوند که تمامی نسخه های نرم افزار نقص های یکسانی نداشته باشند . تکنیک های مختلفی برای تحمل پذیری خطا در نرم افزارهای چند نسخه ای به شرح زیر وجود دارند :

1 – بلوک های بازیابی

این روش ترکیبی از روش شروع مجدد و نقاط چک شده همراه با افزونگی است . ساختار اصلی در شکل زیر قابل مشاهده است .

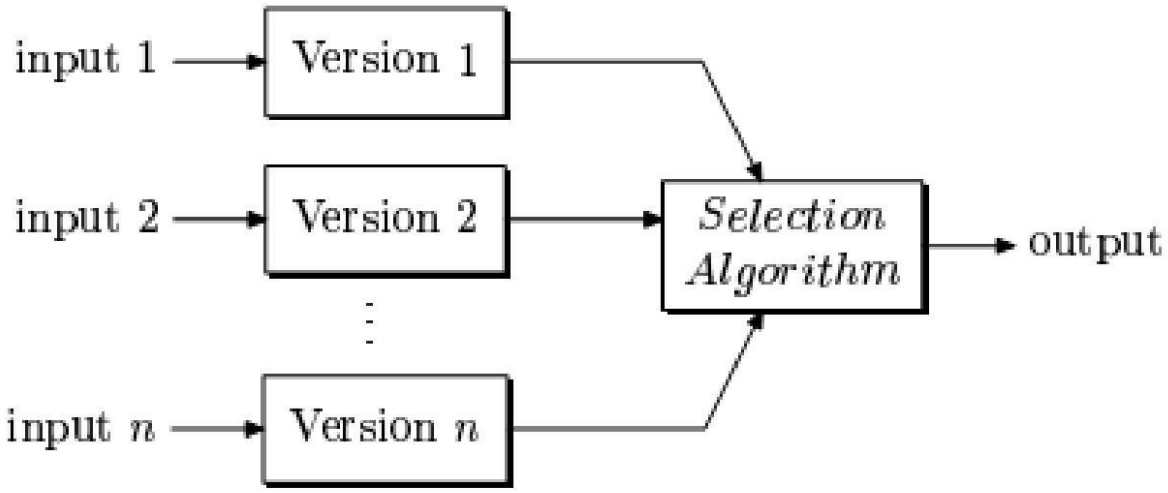


شکل 16 – بلوک های بازیابی

نسخه های 1 تا N نشان دهنده پیاده سازی های مختلف از یک برنامه می باشند . تنها یکی از نسخه ها خروجی سیستم را تولید می کند .اگر خطایی توسط آزمون پذیرش (AT) آشکار شود ، سیگنال Retry به سوییچ ارسال می شود . در اینصورت سیستم به وضعیت ذخیره شده در حافظه نقطه چک شده بر می گردد (عقبگرد می کند) و سوییچ نسخه دیگری از واحد را اجرا می کند . نقاط چک شده قبل از اجرای هر نسخه ایجاد می شوند . بررسی های مختلفی در آزمون پذیرش نسخه فعال واحد صورت می گیرد . فرآیند بررسی کردن یا در خروجی یک واحد اعمال می شود یا برای افزایش اثربخشی آشکارسازی نقص به صورت توکار در کد برنامه قرار می گیرد .

برنامه نویسی چند نسخه ای

این روش مشابه افزونگی سخت افزاری N-Modular است . نمودار بلوکی این تاکتیک در شکل زیر قابل مشاهده است.

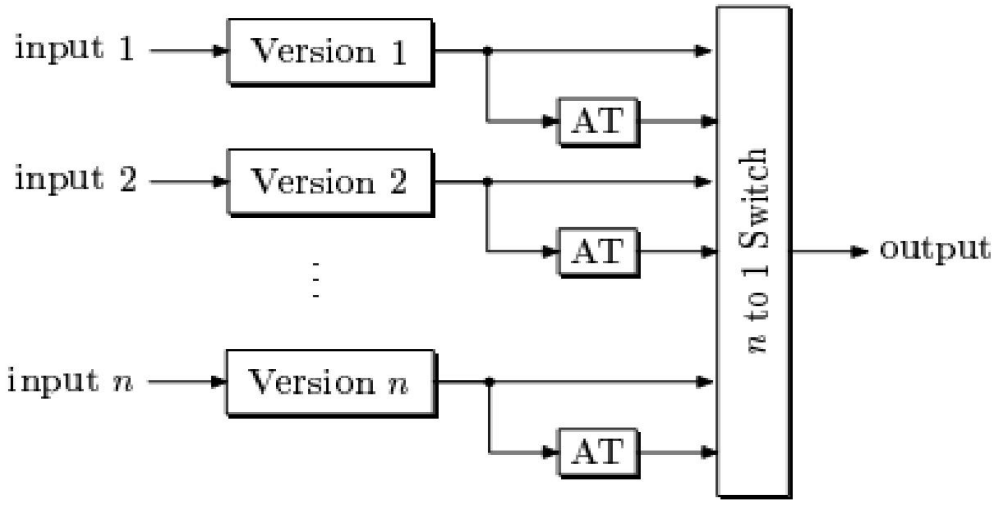


شکل 17 – برنامه نویسی چند نسخه ای

این نمودار بلوکی شامل N تا پیاده سازی مختلف نرم افزاری از یک واحد می باشد که به طور همزمان اجرا می شوند. تمامی نسخه ها یک وظیفه را البته با روش های متفاوت انجام می دهند . الگوریتم انتخاب در این نمودار بلوکی تصمیم می گیرد که کدام یک از جواب ها درست است و جواب درست را به عنوان نتیجه ای از اجرای واحدها برمی گرداند .الگوریتم انتخاب معمولا به عنوان یک رای گیرنده 65 عمومی پیاده سازی میشود .

برنامه نویسی خود بررسی (NSCP)

ترکیبی از تکنیک های برنامه نویس چند نسخه ای و بلوک های بازیابی است . فرآیند بررسی کردن یا توسط آزمون های پذیرش و یا فرآیند های مقایسه صورت می پذیرد . بلوک دیاگرام این روش را می توانید در شکل زیر ملاحظه بفرمایید .

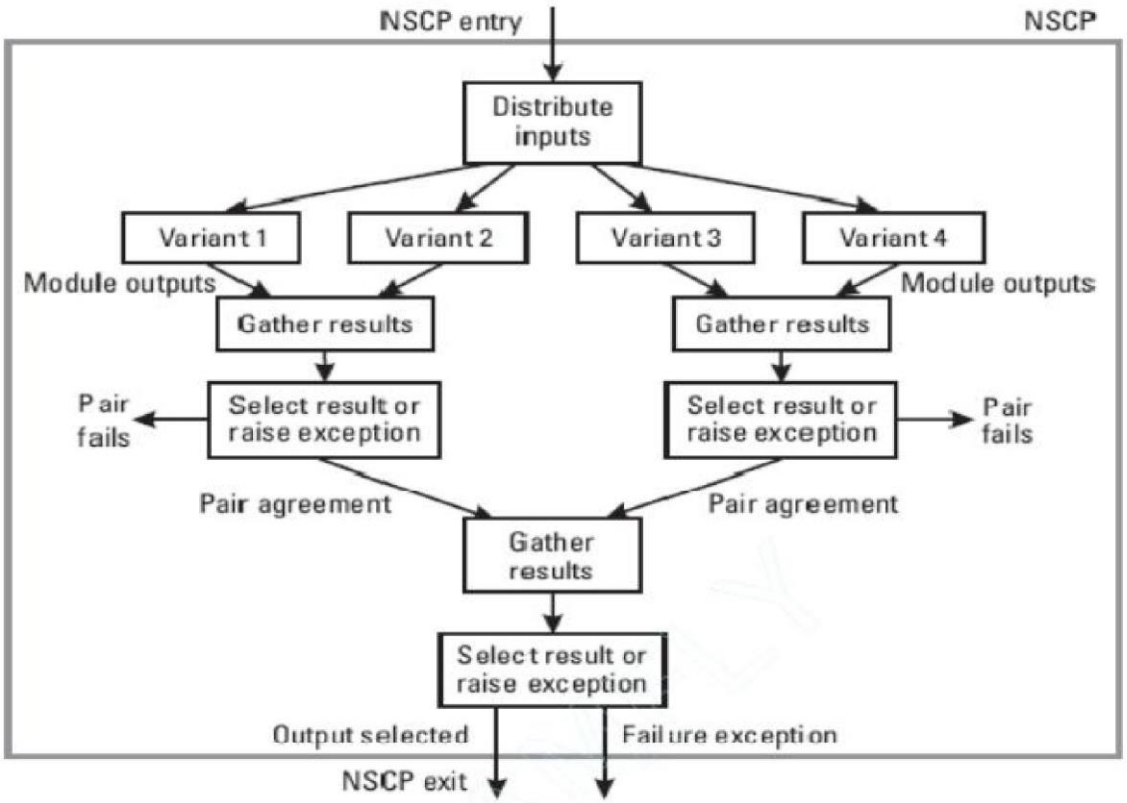


شکل 18 – برنامه نویسی خود برسی

نسخه های مختلف واحدهای برنامه و آزمون های پذیرش ، مستقل از نیازمندی های عمومی طراحی شده اند . فرآیند بررسی کردن یا منحصربفرد برای هر نسخه به صورت توکار در برنامه به کار می رود و یا در خروجی اعمال می شود. تنها تفاوت اصلی این روش با روش بلوک های بازیابی ، در استفاده از آزمون های مجزا برای هر نسخه است . اجرای هر نسخه می تواند به صورت سریال و یا به صورت همزمان انجام گیرد . در هر دو مورد خروجی از نسخه های با اولویت بالا که از مرحله آزمون پذیرش خود عبور کرده اند ، گرفته می شود.

یک روش دیگر پیاده سازی برنامه نویسی خودبررسی ، استفاده از مقایسه به جای AT می باشد . مزیت این روش در این است که یک برنامه مستقل از الگوریتم تصمیم گیری (فرآیند مقایسه) برای آشکارسازی نقص به کار می رود . به عنوان نمونه می توان هواپیمای ایرباس A-340 و همچنین سوییچ تلفن ESS Lucent را نام برد که از این روش استفاده کرده اند.

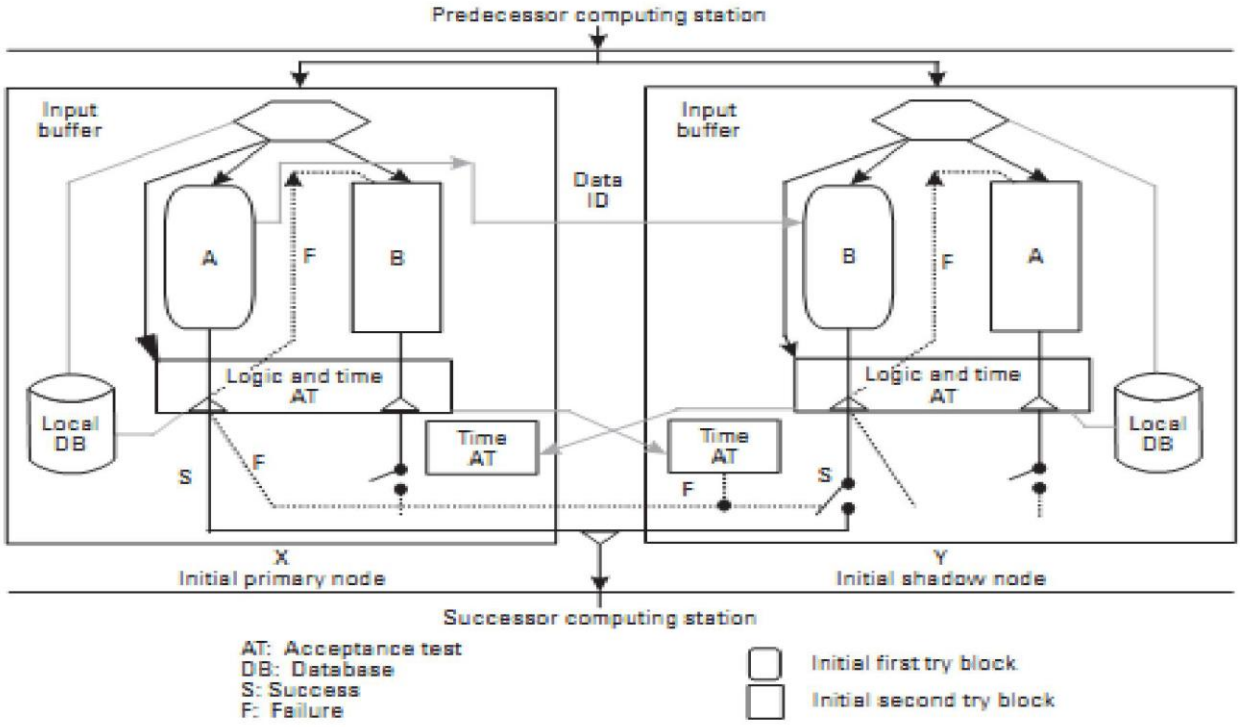
روش برنامه نویسی خودبررسی n نسخه را به صورت همزمان بر روی n/2 جفت سخت افزاری انجام می دهد. نتایججفت نسخه ها با همدیگر مقایسه می شوند اگر نتایج هر جفت مطابقتی نداشت ، پرچم خرابی را نشان می دهد . اگرخرابی فقط در یک جفت روی دهد ، آنگاه نتایج جفت دیگر(بدون خرابی) به عنوان نتیجه این روش برگردانده می شود اگر هر دو جفت دارای خطا بودند ، آنگاه اجرا کننده خطا را مشخص مینماید . اگر نتایج هر جفت با هم مطابقت داشتند ، آنگاه نتایج جفت واحدها با هم مقایسه می شود و در صورتیکه تمامی نتایج با هم مطابقت داشته باشند آنگاه نتیجه مورد نظر به عنوان نتیجه روش برنامه نویسی خودبررسی برگردانده می شود . در غیر اینصورت اجرا کننده خرابی را نشان می دهد . این عملکرد در شکل زیر قابل مشاهده است.



شکل 19 – عملکرد NSCP

بلوک های بازیابی توزیع شده

این روش ترکیبی از پردازش موازی و بلوک های بازیابی است که تحمل پذیری خطا را هم در سخت افزار و هم در نرم افزار ایجاد می کند . این تکنیک در کاربردهای بلادرنگ ، سیستم های محاسباتی توزیع شده و موازی و کنترل نقص های سخت افزاری و نرم افزاری به کار می رود . شکل زیر عملکرد این تاکتیک را نشان می دهد.



شکل 20 - عملکرد بلوك هاي بازيابی توزيع شده

مطابق شکل فوق اساس تاکتیک بلوک های بازیابی توزیع شده شامل یک گره اصلی و یک گره جایگزین می باشد ، که هر کدام از آنها یک شمای بلوک های بازیابی توزیع شده را اجرا می نمایند . یک بافر ورودی در هر گره داده ورودی را نگه می دارد که در چرخه بعدی استفاده می شود . آزمون پذیرش زمانی و منطقی ، ترکیبی از یک آزمون پذیرش و تایمر نگهبان است . آزمون پذیرش زمانی ، یک تایمر نگهبان است که گره دیگر را در جفت گره ها بررسی می کند. پایگاه داده محلی نتیجه محلی جاری را نگهداری می کند . کد زیر اجرای بلوک های بازیابی توزیع شده را انجام می دهد .

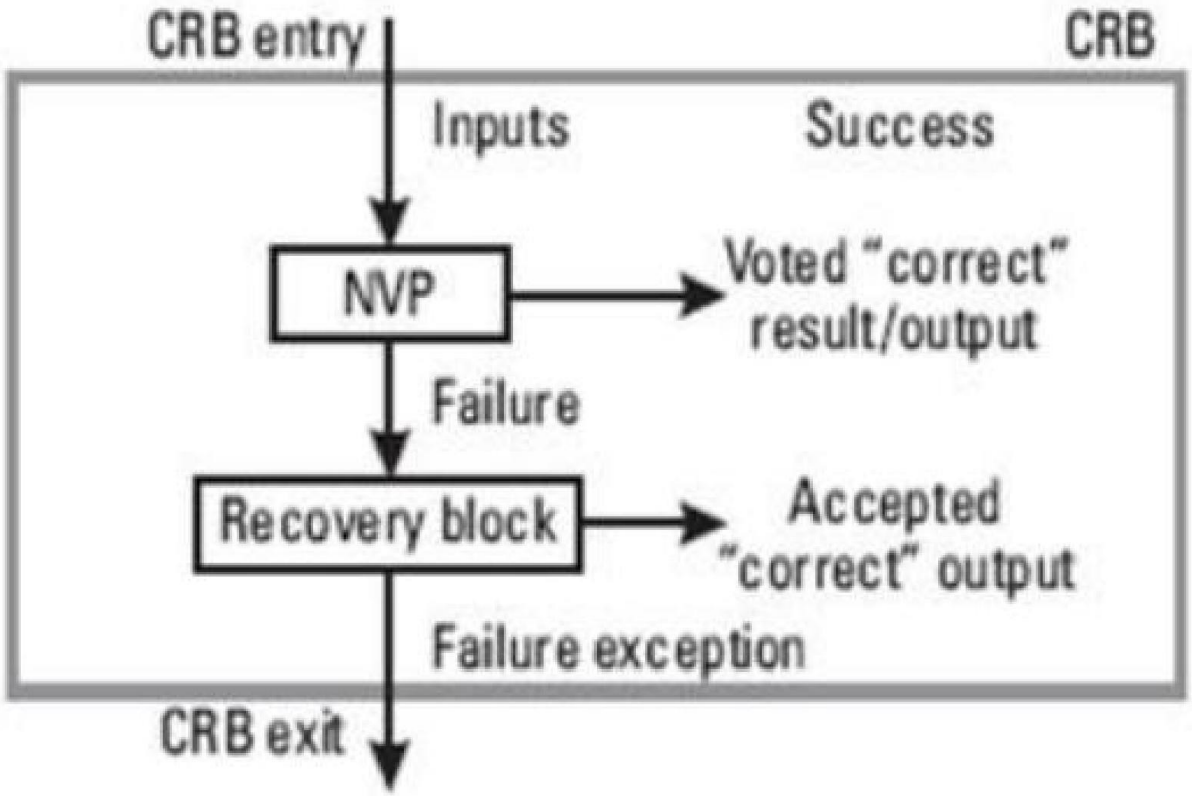
```
run       RB1 on Node 1 (Initial Primary),
               RB2 on Node 2 (Initial Shadow)
ensure    AT on Node 1 or Node 2
by        Primary on Node 1 or Alternate on Node 2
else by   Alternate on Node 1 or Primary on Node 2
else failure exception
```

برنامه نشان داده شده در فوق بیان می دارد که تاکتیک مورد نظر بلوک های بازیابی را به صورت همزمان روی هر دو گره اجرا می کند ، بطوریکه یک گره (Initial primary node) ابتدا الگوریتم اصلی را اجرا می کند و گره دیگر (Initial shadow node) الگوریتم جایگزین را اجرا می کند . در این تاکتیک ابتدا از آزمون پذیرش با الگوریتم اصلی روی نتایج گره اصلی اطمینان حاصل می شود) به طور مثال تولید یک نتیجه درست که از آزمون پذیرش عبور می کند ، اگر نتیجه درستی بدست نیامد (خرابی در نتیجه) ، آنگاه تاکتیک بلوک های بازیابی توزیع شده نتیجه حاصل از الگوریتم جایگزین روی گره دوم امتحان می شود .

حال اگر باز هم از مرحله آزمون پذیرش عبور نکند ، آنگاه تکنیک بازگشت به عقب برای اجرای الگوریتم جایگزین روی گره یک و الگوریتم اصلی روی گره دوم صورت می پذیرد . نتایج این اجراها درمرحله آزمون پذیرش بررسی می شوند . اگر هیچکدام از نتایج ، مرحله آزمون پذیرش را ذر نکنند ، پس خطایی صورت گرفته است . اگر هر کدام از نتایج موفقیت آمیز باشند ، آن نتیجه بر روی خروجی قرار می گیرد.

بلوک های بازیابی اجماعی

این روش ترکیبی از پیاده سازی دو روش بلوک های بازیابی و برنامه نویسی چند نسخه ای است . شکل زیر ساختار این روش را نشان می دهد .

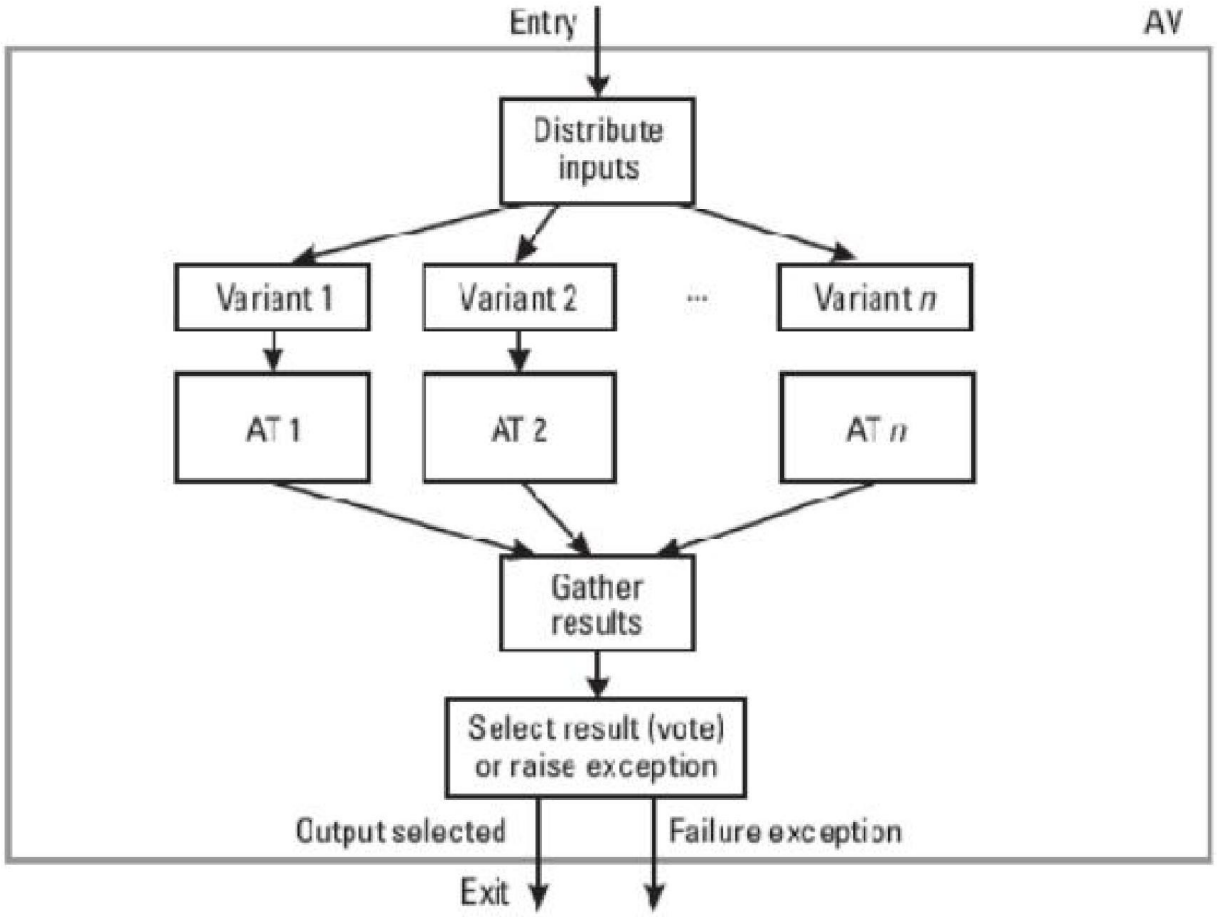


شکل 21 - بلوك هاي بازيابی اجماعی

این روش هم به مانند بلوک های بازیابی از n گونه استفاده می کند . این n گونه در ابتدا بطور همزمان در حالت چند نسخه ای اجرا می شوند و نتایج آنها توسط یک رای گیرنده بررسی می شود . اگر رای گیرنده نتیجه درستی تعیین نکند (توسط یک الگوریتم تصمیم گیری اکثریت) ، آنگاه نتایج بالاترین گونه برای آزمون های پذیرش ارائه می گردد . اگر نتایج گونه مورد نظر درست نباشد ، نتایج یک گونه دیگر برای آزمون پذیرش ارسال می گردد و این روند تا تولید یک نتیجه قابل قبول ادامه دارد .

رای گیری پذیرش

روش رای گیری پذیرش (AV) از یک آزمون پذیرش و یک ماشین تصمیم گیرنده بر اساس رای گیریاستفاده می کند و دارای یک ساز و کار بازیابی رو به جلو برای انجام تحمل پذیری خطا می باشد . در این روش تمامی نسخه ها می تواند به صورت موازی اجرا شوند . نتایج این نسخه ها توسط یک آزمون پذیرش ارزیابی می شود و تنها نتایج پذیرفته شده به رای گیرنده ارسال می شوند . در این روش یک الگوریتم رای گیری پویا وجود دارد که قادر به پردازش تعداد گوناگونی از نتایج می باشد . شکل زیر ساختار و عملکرد روش رای گیری پذیرش را نشان می دهد .



شکل 22 – رای گیری پذیرش

این روش همان گونه که در شکل فوق قابل مشاهده است شامل اجرا کننده ، N نسخه ، آزمون پذیرش و یک ماشین تصمیم گیرنده براساس رای گیری می باشد . اجرا کننده ، عملکرد روش AV را هماهنگ می کند . n نسخه بصورت همزمان اجرا می شوند و نتایج هر یک از این اجراها به آزمون پذیرش اعمال می گردند . برای هر نسخه می توان از یک آزمون پذیرش متفاوت استفاده کرد . هر چند که در عمل از یک الگوریتم آزمون پذیرش منحصربفرد استفاده می شود . تمام نتایجی که از مرحله آزمون پذیرش عبور می کنند به ماشین تصمیم گیرنده (DM) ارسال می شوند . ماشینتصمیم گیرنده اکثریت را انتخاب می کند و به خروجی

می دهد . اگر هیچ نتیجه ای مرحله آزمون پذیرش خود را نگذراند یا هیچ نتیجه اکثریتی وجود نداشته باشد ، در اینصورت استثنایی رخ داده است.

منبع :

Software Fault Tolerance Techniques and Implementation . Laura L .Pullum