فصل ۲۶ مهندسی نرم افزار «اطاق پاک»(Clean Room)

مفاهیم کلیدی (مرتب بر حروف الفبا)

آزمون استفاده آماری ، اثبات صحت ، پالایش طراحی ، توزیع احتمال آزمون ، راهبردهای اطاق پاک ، ساختار جعبه ، گواهی ، محرک ، مشخصه های جعبه سیاه ، مشخصه های جعبه شفاف ، مشخصه های جعبه وضعیت ، مشخصه های طراحی ، مشخصه های کارکردی ، تعیین صحت

KEY CONCEPTS

black-box spec , box structure , certification , cleanroom strategy , clear-box spec , design refinement , functional spec , proof of correctness , state-box spec , statistical use testing , stimulus , test probability distribution , verification

نگاه اجمالی

مهندسی نرم‌افزار "اطاق پاک" چیست؟ تا به‌حال چند دفعه شنیده‌اید کسی بگوید «این کار را اولین بار درست انجام بده!»؟ این فلسفه تکراری این شیوه است یعنی فرآیندی که بر شناسایی درستی چیزی از نظر ریاضی پیش از شروع برنامه و گواهی قابلیت اطمینان نرم‌افزار به‌عنوان بخشی از کار آزمون تأکید دارد. خط انتهایی رسیدن به حداقل اشکال که رسیدن به آن بدون استفاده از روش‌های رسمی سخت یا غیرممکن است.

چه کسی عهده دار این امر می باشد؟ یک مهندس نرم‌افزار متخصص و آموزش دیده.

چرا "اطاق پاک" از اهمیت برخوردار است؟ اشتباهات باعث انجام کار مجدد می‌شود. کار مجدد زمان می‌برد و هزینه را افزایش می‌دهد. آیا بهتر نیست در هنگام طراحی و ساخت نرم‌افزار تعداد خطاها را به میزان چشمگیری کاهش دهیم؟

مراحل کار چیست؟ مدل‌های تحلیل و طراحی با استفاده از نمایش ساختار جعبه‌ای یا باکس ایجاد می‌شوند. هر کادر، جعبه یا «باکس» در برگیرنده سیستم (یا جنبه‌ای از آن) در سطح تجریدی به‌خصوصی است. تشخیص درستی زمانی به‌کار گرفته می‌شود که صورتی کامل شده باشد. وقتی درستی آن تأیید شد، آزمون استفاده آماری آغاز می‌شود. نرم‌افزار توسط مجموعه‌ای از سناریوهای کاربرد مورد آزمون قرار می‌گیرد که احتمال استفاده از هرسناریو تعیین شده و سپس آزمونهای تصادفی که

با این احتمالات مطابقت دارند به‌کار می‌رود. این خطاهای ثبت شده از نظر نتیجه مورد تحلیل قرار می‌گیرند تا محاسبه ریاضی درستی و اطمینان از پروژه در مورد هر جزء نرم‌افزاری ممکن شود.

محصول کار چیست؟ مشخصات State Box, Black Box و Clear Box (جعبه سیاه، جعبه وضعیت، جعبه سفید) ارائه می‌شوند. نتایج استدلال‌های رسمی از نظر درستی و آزمونهای استفاده آماری ثبت می‌شوند.

چگونه مطمئن شوم که کارم را درست انجام داده‌ام؟ اثبات صحت رسمی در مورد مشخصات ساختار جعبه ارائه می‌شود. آزمون استفاده آماری به‌کارگیری سناریوها را اجرا می‌کند تا مطمئن شود که خطاهای کارکرد کاربر، تحت نظر قرار گرفته و اصلاح شده‌اند. از اطلاعات آزمون برای مهیا کردن معیارهای اطمینان از نرم‌افزار استفاده می‌شود.

استفاده منسجم و یکپارچه از مدل‌سازی قراردادی مهندسی نرم‌افزار (و احتمالاً روش‌های رسمی)، شناسایی برنامه (دلایل درستی) و SQA آماری، در تکنیکی تلفیق شده که می‌تواند منجر به نرم‌افزاری با کیفیت بسیار بالا شود. مهندسی نرم افزار اطاق پاک[1] رهیافتی است که بر نیاز به برقراری صحت و درستی در نرم‌افزار تولیدی، تأکید دارد. به‌جای تحلیل کلاسیک، طراحی، برنامه‌نویسی، آزمون و چرخه اشکال‌زدایی، رهیافت "اطاق پاک" بیان‌گر دیدگاه متفاوتی است.[LIN94] [2]

فلسفه نهفته در پس این روش عبارتست از پرهیز از وابستگی به فرآیندهای پر هزینه رفع‌اشکال، با نوشتن انبوه برنامه‌های درست در همان اولین بار، و شناسایی و تصدیق درستی آنها قبل از آزمون. مدل فرآیند آن، گواهی کیفی آماری، افزایش برنامه را به‌کار می‌گیرد که این امر، با انباشته شدن آنها در سیستم رخ می‌دهد.

از بسیاری جهات، رهیافت اطاق پاک سطح مهندسی نرم‌افزار را ارتقاء می‌دهد. مانند روش‌های رسمی ارائه شده در فصل ۲۵، فرآیند اطاق‌پاک بر سختی و استحکام مشخصات و طراحی تأکید داشته و شناسایی رسمی هر عنصر طرح با استفاده از دلایل و استدلال درستی است که بر پایه ریاضی می‌باشند. با بسط هر روش به‌کار رفته در شیوه های رسمی که روش "اطاق پاک" نیز بر فنون کنترل کیفی آماری متمرکز است، فنونی که مشتمل بر آزمون نرم‌افزار توسط مشتری و در مسیر پیش‌بینی شده، می باشد.

وقتی در محیط واقعی نرم‌افزار نمی‌تواند عمل کند، خطرات بلادرنگ و دراز مدتی پدیدار می‌شوند. این خطرات را می‌توان به ایمنی انسان، ضرر اقتصادی یا عملیات مؤثر تجاری و زیربنای اجتماعی ربط داد. مهندسی نرم‌افزار "اطاق پاک" یک مدل پردازشی است که نقایص را پیش از آن‌که خطرات جدی ایجاد کنند از بین می‌برد.

1.cleanroom software engineering
2.Linger,R.

۲۶-۱ رهیافت اطاق پاک

نقل قول

مهندسی "اطاق تمیز" کنترل کیفیت آماری را در طول توسعه نرم افزار سرعت می دهد. این مهم با جداسازی دقیق فرآیند طراحی از فرآیند آزمون، در یک توسعه افزایشی نرم افزار انجام می پذیرد.

هارلان میلز

فلسفه، "اطاق پاک" در فنآوری‌های ساخت سخت‌افزار کاملاً ساده است: از نظر هزینه مقرون به‌صرفه و از نظر زمانی نیز صرفه‌جویی شده تا روش تولیدی به‌وجود آورد که نقایص محصول را پیش‌بینی می‌کند. علاوه بر ساخت یک محصول و سپس کار رفع نواقص، این روش نیازمند دیسیپلین و نظم لازم برای حذف و از بین بردن نقایص در مشخصات و طراحی و سپس ساخت آن به شیوه‌ای راحت و پاکیزه خواهد بود.

فلسفه "اطاق پاک" ابتدا در مورد طراحی نرم‌افزار توسط میلز و همکارانش[MIL87][1] در طول ده ۸۰ ارائه شد. گرچه تجربیات اولیه در مورد این روش نظامند[HAU94][2] در کار با نرم‌افزار تعمد مهمی را نشان داد اما کاربرد چندان گسترده‌ای نیافت. هندرسون[HEN95][3] سه دلیل احتمالی برای آن بیان می‌دارد:

۱- این عقیده که فراروش "اطاق پاک" ، برای تولید نرم‌افزار به‌صورت واقعی بسیار تئوریک ، ریاضی و افراطی است .

۲- این روش از آزمون واحد توسط تولیدکننده حمایت نمی‌کند، اما در عوض گواهی و تأییدیه درستی و کنترل کیفی آماری را جایگزین آن می‌کند ـ مفاهیمی که نمایان‌گر یک تفکیک عمده از شیوه‌ای است که اکثر نرم‌افزارها امروزه بر پایه آن تولید شده‌اند.

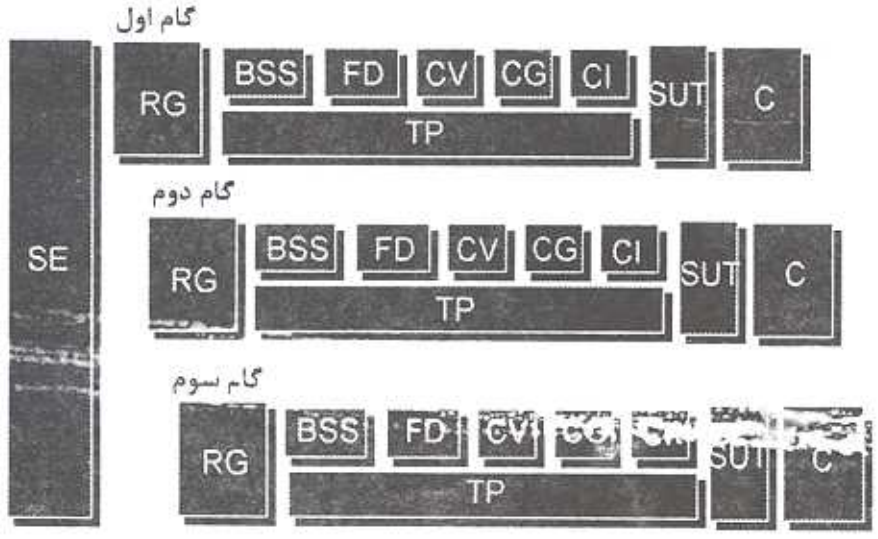
۳- به بلوغ رسیدن صنعت تولید نرم‌افزار. استفاده از فرآیندهای "اطاق پاک" مستلزم به‌کارگیری مستمر و جدی و سخت فرآیندهای تعریف شده در تمام فازهای چرخه حیات است. از آن‌جا که قسمت عمده‌ای از صنعت هنوز در سطح ویژه‌ای کار می‌کند (همان‌گونه که توسط مدل بلوغ قابلیت توسط مؤسسه مهندسی نرم‌افزار تعریف شده) و این صنعت هنوز *آماده به‌کارگیری چنین فنونی نیست.*

گرچه عناصر حقیقی در هر یک از مفاهیم فوق‌الذکر وجود دارد، اما مرایای بالقوه مهندسی نرم‌افزار "اطاق پاک" بیشتر از سرمایه‌گذاری لازم برای فایق شدن بر مقاومت فرهنگی است که در مرکز این دل مشغولی‌ها قرار دارد.

۲۶-۱-۱ راهبردهای اطاق پاک

این رهیافت ازنسخه خاصی از مدل فزاینده و تکاملی نرم‌افزاری (فصل ۲) استفاده می‌کند. مسیری از تکثیر نرم‌افزارها توسط تیم‌های کوچک نرم‌افزاری مستقل ایجاد می‌شود. از آن‌جا که هر افزایشی تأیید شده می‌باشد، در قالب یک کل منسجم و یکپارچه تلفیق خواهدشد. بنابراین، قابلیت کارآیی سیستم با زمان افزایش می‌یابد [LIN94]

1.Mills,H.D.
2.Hausler,P.A.
3.Henderson,J.



شکل ۲۶ـ۱ مدل فرآیند اطاق پاک

توالی کارهای "اطاق پاک" برای هر افزایش در شکل ۲۶-۱ آمده است. نیازمندیهای کلی سیستم یا محصول با استفاده از روش‌های مهندسی سیستم که در فصل ۱۰ مورد بحث قرار گرفته ارائه می‌شوند. وقتی قابلیت کارکردی به عنصر نرم‌افزاری سیستم ...گذا... [illegible] ...طاق پاک ... ری شد و وظایف زیر رخ می‌دهند:

طرح ریزی و برنامه‌ریزی افزایشی. طرح پروژه که در برگیرنده راهبرد افزایشی است، ارائه می‌شود. قابلیت کارکرد هر تکثیر، اندازه پروژه آن و جدول برنامه‌ریزی توسعه "اطاق پاک" ایجاد می‌شود. باید مراقبت ویژه‌ای درنظر گرفت تا مطمئن شویم افزایش‌های به تأیید رسیده به‌صورت زمان‌بندی شده و به‌موقع به‌هم مرتبط می‌شوند.

جمع‌آوری نیازمندیها. با استفاده از فنونی مشابه آن‌چه که در فصل ۱۱ معرفی شدند، توصیف دقیق‌تری از نیازمندیهای مشتری (در مورد هر افزایش) ارائه می‌شود.

پرسش

وظائف اصلی که به عنوان بخشی از مهندسی نرم افزار " اطاق تمیز " قلمداد می شوند، کدام هاست؟

ارجاع به وب

منبعی ممتاز از اطلاعات و منابع مهندسی نرم افزار " اطاق تمیز " در آدرس زیر وجود دارد
www.cleansoft.com

مشخصه های ساختار جعبه. شیوه مشخصاتی که از ساختارهای جعبه استفاده می‌کند برای توصیف مشخصه عملکردی به‌کار می‌رود. مطابق با اصول بحث در فصل ۱۱، ساختارهای جعبه، تعریف رفتار، داده‌ها و رویه‌های هر سطح از روال کار را تعیین [Hev۸۸][1]

طراحی رسمی. با استفاده از روش ساختار جعبه، طراحی "اطاق پاک" عبارتست از بسط و گسترش طبیعی و بدون اشکال مشخصات. گرچه امکان دارد تمایز مشخصی بین دو فعالیت ایجاد نمود اما مشخصات (به‌اصطلاح جعبه های سیاه) بمعنای بکراری مو شکافی می‌گردند تا مشابه طراحی در سطح معماری و اجراء شوند.(که به ترتیب جعبه‌های وضعیت و سفاف نامیده می شوند).

تعین صحت درستی. تیم "اطاق پاک"، یکسری کارهای سخت برای تأیید درستی روی طرح و سپس روی برنامه انجام می‌دهند. تأیید (بخش‌های ۲۶-۳ و ۲۶-۴) با بالاترین سطح ساختار جعبه شروع شده (مشخصات) و به طرح جزئیات طراحی و برنامه حرکت می‌کند. اولین سطح این تأییدیه با به‌کارگیری مجموعه‌ای از «سؤالات در مورد درستی» آغاز می‌شود.[LIN88]

اگر این موارد نشان ندهند که مشخصات درستند، روش‌های رسمی‌تری برای این تأیید استفاده می‌شوند.

ایجاد، بازرسی و تعیین صحت کد و برنامه. مشخصات ساختار جعبه که به زبان خاصی نمایش داده شده، به زبان برنامه‌نویسی مناسب انتقال می‌یابند. فنون استاندارد بازرسی (فصل ۸) برای اطمینان از تطابق معنایی برنامه و ساختار جعبه‌ها و درستی معنایی برنامه استفاده می‌شوند. سپس تأییدیه درستی برای برنامه منبع اجرا می‌گردد.

نقل قول

کیفیت یک اقدام نیست، یک عادت است.
آریستوتل

برنامه نویسی آزمون آماری. کاربرد نرم‌افزار در بروژه تحلیل شده و یکسری آزمونهای مناسب که بوریع احتمالی کاربرد را [illegible]، این کار "اطاق پاک" به‌صورت موازی با مشخصات، تأییدیه و [illegible]

" اطاق پاک " تاکید بر آزمونهایی دارد که اجرای واقعی نرم افزار را آزمایش می کنند. Use_Case ها ورودی ممتازی برای فرآیند طرح ریزی آزمون آماری، فراهم می سازند.

آزمون استفاده آماری. به یاد دارید که آزمون جامع و کامل نرم‌افزار کامپیوتری غیرممکن است (فصل ۱۷) و همیشه لازم است چند آزمون محدود را طراحی نمود. فنون استفاده آماری [POO88][2] یکسری آزمون اجرا می‌کنند که از یک نمونه آماری از تمام اجراهای احتمالی برنامه به‌وسیله کاربران از یک جمعیت تعیین شده، مشتق شده‌اند.

گواهی. هنگامی که تأیید، بازرسی و آزمون کاربرد تکمیل شد (و همه خطاها اصلاح شده‌اند) این افزایش از نظر آمادگی برای یکپارچه‌سازی گواهی می‌گردد.

همچون سایر مدل‌های فرآیند نرم‌افزاری که در جای دیگری در این کتاب مورد بحث قرار گرفت، فرآیند "اطاق پاک" تا حد زیادی به مدل های تحلیل و طراحی دارای کیفیت بالا، تکیه دارد همان‌گونه که

1.Hevner, A.R.
2.Poore,J.H.

بعداً در این فصل خواهیم دید، ساختمان باکس صرفاً شیوه دیگری برای نمایش نیازمندیها و طراحی توسط مهندس نرم‌افزار است. تفاوت واقعی رهیافت "اطاق پاک" این است که تأییدیه و تعیین صحت رسمی در مورد مدل‌های مهندسی به‌کار گرفته می‌شود.

۲۶-۱-۲ چه چیز اطاق پاک را متفاوت می سازد؟

دایر[DYE92][1] در ضمن تعریف فرآیند، اشاره‌ای دارد به تفاوت‌های روش "اطاق پاک":

"اطاق پاک" نمایان‌گر اولین تلاش عملی در گذاردن فرآیند تولید نرم‌افزار، تحت کنترل کیفی آماری با یک راهبرد به دقت تعریف شده برای پیشرفت مستمر فرآیند انجام خواهد شد. به‌منظور رسیدن به این هدف، چرخه حیات منحصر بفرد "اطاق پاک" تعریف شد که بر مهندسی نرم‌افزار مبتنی بر ریاضی از نظر طراحی درست نرم‌افزاری و آزمون مبتنی بر آمار از نظر تأیید قابلیت اطمینان نرم‌افزار تأکید داشت.

مهمترین ویژگی برجسته "اطاق نمیز" اثبات صحت و آزمون کارکرد آماری می باشد.

مهندسی نرم‌افزاری "اطاق پاک" از دیدگاه‌های شی‌گرا و قراردادی به نمایش درآمده در بخش‌های سه و چهار این کتاب متفاوت است، زیرا:

۱- استفاده صریحی از کنترل کیفی آماری می‌کند.

۲- مشخصات طراحی را با استفاده از استدلال درستی مبتنی بر ریاضی، تأیید می‌کند.

۳- شدیداً بر آزمون استفاده آماری برای مشخص کردن خطاهای مؤثر پنهان مانده، تکیه دارد.

مشخص است که رهیافت "اطاق پاک" اگر نگوییم همه، اما اکثر اصول مقدماتی مهندسی نرم‌افزار و مفاهیم اولیه ارائه شده در طول این کتاب را به‌کار می‌گیرد. رویه‌های خوب تحلیل و طراحی برای کسب بالاترین کیفیت ضروری هستند. اما مهندسی "اطاق پاک" با عدم تأکید بر (که بعضی‌ها آن را حذف می‌دانند) نقش آزمون واحد و اشکال‌زدایی و کاهش چشمگیر (یا حذف) میزان آزمون صورت گرفته توسط تولیدکننده نرم‌افزار، از شیوه‌های قراردادی نرم‌افزاری منحرف شده است.[2]

در تولید قراردادی نرم‌افزار، خطاها به‌عنوان حقیقتی از زندگی پذیرفته شده‌اند. از آن‌جا که خطاها به ناچار پذیرفته شده‌اند، هر پیمانه برنامه باید از نظر هر واحد آزمون شود (تا خطاها معلوم شده) و سپس اشکال‌زدایی شود. وقتی سرانجام نرم‌افزار ارائه شد، استفاده، نقایص بیشتری را معلوم نموده و چرخه آزمون و اشکال‌زدایی دیگری آغاز می‌شود. کار محدد مربوط به این فعالیتها پرهزینه و وقت‌گیر است. بدتر از همه این‌که، می‌تواند خراب‌کننده و نابودگر باشد، اصلاح خطا می‌تواند منجر به نمایان شدن خطاهای بیشتری شود. در مهندسی نرم‌افزار "اطاق پاک"، آزمون واحد و اشکال‌زدایی با تأییدیه صحت و درستی و آزمون مبتنی بر آمار عوض شده‌اند. این فعالیتها، همراه با نگهداری پرونده که برای پیشرفت ممتد ضروری است، روش "اطاق پاک" را منحصر به‌فرد می‌سازد.

نقل قول

نکته جالب زندگی آن است که : اگر شما همه چیز را به امید بهترین آن پس برنید، اغلب آنرا به دست خواهید آورد (هرچه درین درگه نشانت دهند گر نپسندی به از آنت دهند)

دبلیو سامرست موقام

1.Dyer,M.

۲. آزمون انجام خواهد شد، اما توسط یک تیم مستقل نرم افزاری.

۲۶-۲ مشخصات کارکردی

بدون توجه به روش تحلیل انتخابی، اصول عملیات ارائه شده در فصل ۱۱، به‌کار گرفته می‌شوند. داده‌ها، کارکرد و رفتار مدل‌سازی می‌گردند. باید مدل‌های بدست آمده را تقسیم‌بندی (بالایش) نمود تا جزئیات بیشتری در اختیارمان قرار گیرد. هدف کلی عبارتست از حرکت از سوی مشخصه‌ای که در برگیرنده ماهیت مسئله است به‌سوی مشخصه‌ای که جزئیات ضروری پیاده‌سازی را مهیا می‌سازد.

مهندسی نرم‌افزار "اطاق پاک" با استفاده از روشی به نام مشخصات ساختار جعبه[1] با اصول تحلیل عملیاتی، مطابقت می‌یابد. هر جعبه دربرگیرنده سیستم (یا جهانی از سیستم) از نظر بعضی از جزئیات است. در طول فرآیند پالایش قدم به قدم، جعبه‌ها به‌صورت سلسله مرتبه‌ای درمی‌آیند که در آن هر جعبه دارای شفافیت مرجع[2] است. یعنی محتوای اطلاعاتی هر مشخصه جعبه برای تعریف پاکسازی آن، بدون تکیه بر اجرای جعبه دیگر، کافی است. این کار تحلیل‌گر را قادر به تقسیم‌بندی و پالایش سیستم از نظر سلسله مراتبی ساخته و از نمایش قسمت‌های ضروری در نوک به‌سوی جزئیات خاص پیاده‌سازی در قسمت پایین حرکت می‌کند. سه نوع جعبه مورد استفاده قرار می‌گیرند:

پرسش

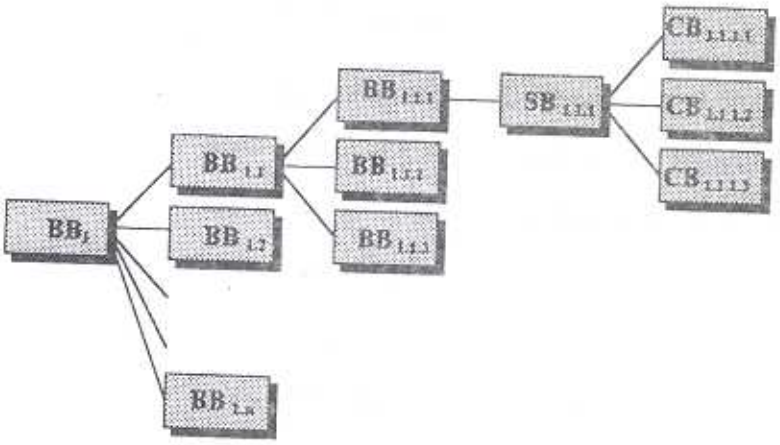
چگونه پالایش، بخشی از تشخیص ساختار جعبه ای را انجام می دهد؟

جعبه سیاه[3]. این جعبه نشانگر رفتار سیستم یا بخشی از آن است. این سیستم به محرکی خاص با به‌کارگیری مجموعه‌ای از قوانین انتقالی پاسخ می‌دهد که محرک‌ها را در برابر واکنش طراحی می‌کنند.

جعبه وضعیت[4]. این جعبه دربرگیرنده اطلاعات وضعیت و خدمات (عملیات) به شیوه‌ای است که برای اشیا مبهم است. دراین دیدگاه، داده‌های جعبه وضعیت (محرک‌ها) و بازده‌ها (واکنش‌ها) به نمایش درمی‌آیند. هم‌چنین این جعبه تاریخچه محرک‌های جعبه سیاه را ارائه می‌دهد. یعنی داده‌ها در جعبه وضعیتی قرار دارند که باید میان انتقالات به‌کار گرفته شده، حفظ و نگهداری شود.

جعبه شفاف[5]. عملیات انتقالی مورد اشاره جعبه وضعیت در این باکس تعریف می‌شوند. اگر بخواهیم ساده‌تر بگوییم **جعبه شفاف** حاوی طراحی رویه‌ای یا روالی **جعبه وضعیت** است.

1.box structure specification
2.refrential transparency
3.Black Box
4.State Box
5.Clear Box



شکل ۲۶-۲ پالایش ساختار جعبه‌ای

شکل ۲۶-۲ رهیافت پالایش را با استفاده از مشخصات ساختار جعبه تشریح می‌کند. BB_1 واکنش‌های مجموعه کاملی از محرک‌ها را تعریف می‌کند. می توان BB_1 را به‌صورت مجموعه‌ای از جعبه‌های سیاه یا BB درآورد که هر کدام یک کلاس رفتار را مورد خطاب قرار می‌دهند. این پالایش تا وقتی ادامه می‌یابد که یک کلاس رفتار منسجم شناسایی می‌شوند. (مثل $BB_{1.1.1}$). پس جعبه وضعیت ($SB_{1.1.1}$) برای جعبه سیاه ($BB_{1.1.1}$) تعریف شده است. در این مورد $SB_{1.1.1}$ حاوی تمام اطلاعات و خدمات لازم برای اجرای رفتار تعریف شده توسط $BB_{1.1.1}$ است. در آخر، $SB_{1.1.1}$ به‌صورت جعبه شفاف ($CB_{1.1.1.n}$) درآمده و جزئیات روال طراحی مشخص می‌شوند.

با وقوع هر مرحله از پالایش، تأیید درستی نیز صورت می‌گیرد. مشخصات جعبه وضعیت مورد تأیید و تحقیق قرار می‌گیرند تا مطمئن شویم که هر کدامشان با رفتار تعریف شده توسط مشخصه اصلی جعبه سیاه مطابقت دارند. به همین شکل، مشخصات جعبه شفاف در برابر جعبه وضعیت اصلی تأیید می‌گردند.

باید توجه داشت که روش‌های مشخصاتی بر پایه روش‌های رسمی (فصل ۲۵) را می‌توان در روش مشخصات ساختار باکس استفاده نمود. تنها چیز مورد نیاز این است که هر سطح از مشخصات به‌صورت رسمی مورد تأیید قرار گیرد.

۲۶-۲-۱ مشخصه های جعبه سیاه

مشخصه های جعبه سیاه با استفاده از عبارت نشان داده شده در شکل۲۶-۳، تجرید، محرک‌ها و واکنش را توصیف می‌کند.[MIL88]' تابع f در رشته S^* از ورودیها (محرک‌ها) S به‌کار رفته و آنها را تبدیل به خروجی R (واکنش) می‌نماید. در مورد اجرای نرم‌افزاری ساده ممکن است f یک تابع ریاضی باشد اما به‌طور کلی f با استفاده از زبان طبیعی توصیف می‌شود (با زبان مشخصاتی رسمی).

بسیاری از مفاهیم تعریف شده (در مورد سیستم‌های شی‌گرا در مورد جعبه سیاه نیز قابل استفاده هستند. تحرید داده‌ها و عملیاتی که آنها را دستکاری می کنند، در احاطه جعبه سیاه هستند. مانند سلسله مراتب کلاس، مشخصات جعبه سیاه می‌تواند سلسله مراتب کاربرد را نشان دهد که در آن جعبه‌های سطح پایین خواص جعبه‌هایی را که در ساختار درختی در قسمت‌های بالاتری هستند، به ارث می‌برند.

۲۶-۲-۲ مشخصه های جعبه وضعیت

این جعبه، یک تعمیم ساده‌ای از وضعیت ماشین است. با یادآوری بحث مدل‌سازی رفتاری و نمودارهای انتقال وضعیت در فصل ۱۲، یک وضعیت ، حالات قابل رؤیتی از رفتار سیستم است. با انجام پردازش، سیستم به‌وسیله انجام انتقالی از حالت کنونی به یک حالت جدید، به حوادث واکنش نشان می‌دهد. با انجام کار انتقال، عملی رخ می‌دهد. جعبه وضعیت یا حالت از تحرید داده‌ها استفاده می‌کند تا کار انتقال به مرحله بعدی را تعیین نموده و این عمل (واکنش) به‌عنوان نتیجه این انتقال رخ می‌دهد.

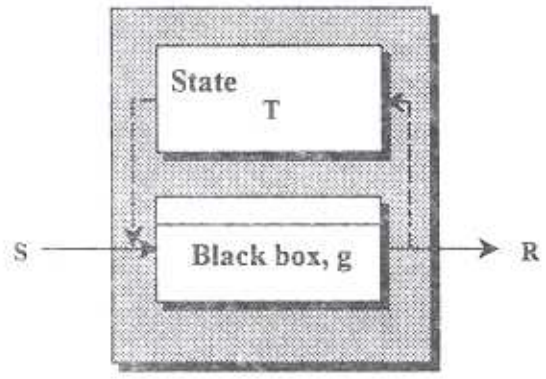
با رجوع به شکل ۲۶-۴، جعبه وضعیت دارای جعبه سیاه است. محرک S که به جعبه سیاه وارد می‌شود از یک منبع خارجی و مجموعه‌ای از حالات درونی سیستم (T) دریافت می‌شود. میلز توضیح ریاضی از تابع F در جعبه سیاه که درجعبه وضعیت قرار دارد ارائه می‌دهد:

$$g : S^* \times T^* \rightarrow R \times T$$

که در آن g زیر تابع است که به وضعیت خاص t مرتبط شده است. وقتی به‌طور جمعی درنظر می‌گیریم جفت‌های زیرتابع ـ وضعیت (t, g) تابع *f* جعبه سیاه را تعریف می‌کنند.

شکل ۲۶-۳ یک مشخصه جعبه سیاه

1.Mills,H.D.



شکل ۲۶ـ۴ مثالی از مشخصات جعبه حالت

۳-۲-۲۶ مشخصه های جعبه شفاف (سفید)

طراحی رویه ای و برنامه سازی ساختیافته در فصل ۱۶ توضیح داده شده اند.

مشخصات جعبه سفید شدیداً با طراحی رویه‌ای و برنامه‌سازی ساختیافته مرتبط است. در اصل، زیرتابع g در جعبه وضعیت یا برنامه‌سازی ساختیافته که اجرای g را صورت می‌دهد، عوض شده‌اند.

به‌عنوان مثال، جعبه شفاف نشان داده شده در شکل ۲۶-۵ را درنظر بگیرید. جعبه سیاه g که در شکل ۲۶-۴ نشان داده شده، با رشته‌ای عوض می‌شود که یک شرطی را در خود دارد. در عوض اینها را می‌توان به‌صورت جعبه های‌شفافی سطح پایینی درآورد که با بالایش مرحله به مرحله درست می‌شوند.

نکته مهم مورد توجه این است که مشخصه‌های رویه‌ای توصیف شده در سلسله مراتب جعبه شفاف را می‌توان از نظر درستی اثبات کرد. این عنوانی است که در بخش بعدی درنظر گرفته شده است.

۳-۲۶ طراحی اطاق پاک

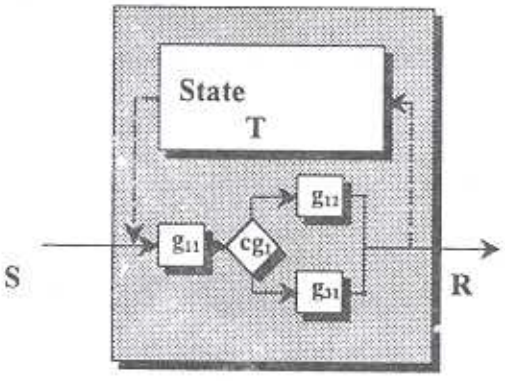
برنامه " داداستارز " رهنمودهای متنوعی از " اطاق تمیز " توسعه داده و مستند ساخته اند :
ftp.cdrom.com/prb/ada/docs/cleanrm/

رهیافت طراحی به‌کار رفته در مهندسی نرم‌افزار اطاق پاک استفاده شدیدی از فلسفه برنامه‌سازی ساختیافته می‌نماید. اما در این مورد، برنامه‌ریزی ساختاری به‌صورت سخت‌تری به‌کار گرفته شده است.

توابع پردازشی اولیه (که در طول اصلاحات مشخصات توصیف شدند) با استفاده از توسعه مرحله به مرحله توابع ریاضی به‌صورت ساختارهای ارتباطات منطقی و زیر تابع‌ها، بالایش می‌گردند و این توسعه تا زمانی صورت می‌گیرد که زیر توابع شناسایی شده را بتوان مستقیماً در زبان برنامه‌نویسی استفاده شده برای پیاده‌سازی بیان نمود. **[DYE92]**

رهیافت برنامه‌نویسی ساختیافته را می‌توان به‌صورت مؤثر برای بالایش کارکرد (تابع) استفاده نمود، اما در مورد طراحی داده‌ها چه؟ در اینجا تعدادی از مفاهیم بنیادی طراحی (فصل ۱۳) وارد عمل

می‌شوند. داده‌های برنامه به‌عنوان مجموعه‌ای از موارد انتزاع و تجرید بسته‌بندی شده‌اند که زیر تابع‌ها (تابع‌های فرعی) به آنها خدمات می‌دهند. مفاهیم بسته‌بندی داده‌ها، پنهان‌سازی و احتفای آنها و نوع این داده‌ها، برای ایجاد طراحی داده‌ها به‌کار می‌رود.



شکل ۲۶ـ۵ مثالی از مشخصات جعبه سفید

۲۶ – ۳ – ۱ پالایش و تعیین صحت طراحی

پرسش

چه شرایطی برای اثبات صحت ساختارها باید به کار گرفته شوند؟

هر مشخصه جعبه سفید(شفاف) نمایان‌گر طرح یک رویه لازم برای رسیدن به انتقال جعبه وضعیت است. با جعبه سفید، برنامه‌نویسی ساختیافته ایجاد شده و پالایش گام به گام همان‌گونه که در شکل ۲۶-۶ آمده، استفاده می‌گردد. تابع برنامه به نام *f* در زنجیره‌ای از زیرتابع‌های g و h قرار می‌گیرد. اینها نیز به نوبه خود در ساختارهای شرطی (if- then - else و do-while) هستند. اصلاح و بهبود بیشتر پالایش و اصلاح منطقی متوالی را نشان می‌دهد.

در هر سطح از این کار، تیم "اطاق پاک"[1]، تأییدیه درستی را به‌طور رسمی اعلام می‌کند. برای رسیدن به این امر مجموعه‌ای از شرایط کلی درستی[2] به ساختار برنامه‌نویسی مرتبط می‌شود. اگر تابع f در زنجیره g و h بسط یابد، شرط درستی برای همه ورودیهای تابع f عبارتست از:

- **Does *g* Followed by *h* do *f*?**

وقتی تابع P در شرط (if - then - else) ادغام می‌شود، شرط درستی برای همه ورودیهای P می‌شود:

- **Whenever condition <c> is true does *q* do *p* and whenever <c> is false, does *r* do *p*?**

(هرگاه شرط c درست است *q* و *p* هم همین‌طور و هرگاه شرط C غلط باشد r و *p* هم همین‌طور)

وقتی تابع m به‌عنوان یک حلقه درمی‌آید، شرایط درستی برای همه ورودیهای m می‌شود:

ترفند

اگر شما طی طراحی روال ها، خود را مقید به ساختارهای داده ای نمایید، اثبات صحت ساده و قابل فهم خواهد بود. اگر شما ساختارهایی غیر سالم را به کار گیرید، اثبات صحت سخت و ناممکن می شود.

---

۱. به دلیل آنکه کل تیم درگیر فرآیند تعیین صحت خواهد بود، احتمال کمی وجود دارد که یک خطا در تعیین صحت بوجود آید.

2. orrectness conditions

- **Is Termination Guaranteed?** (آیا خاتمه تضمین شده است؟)

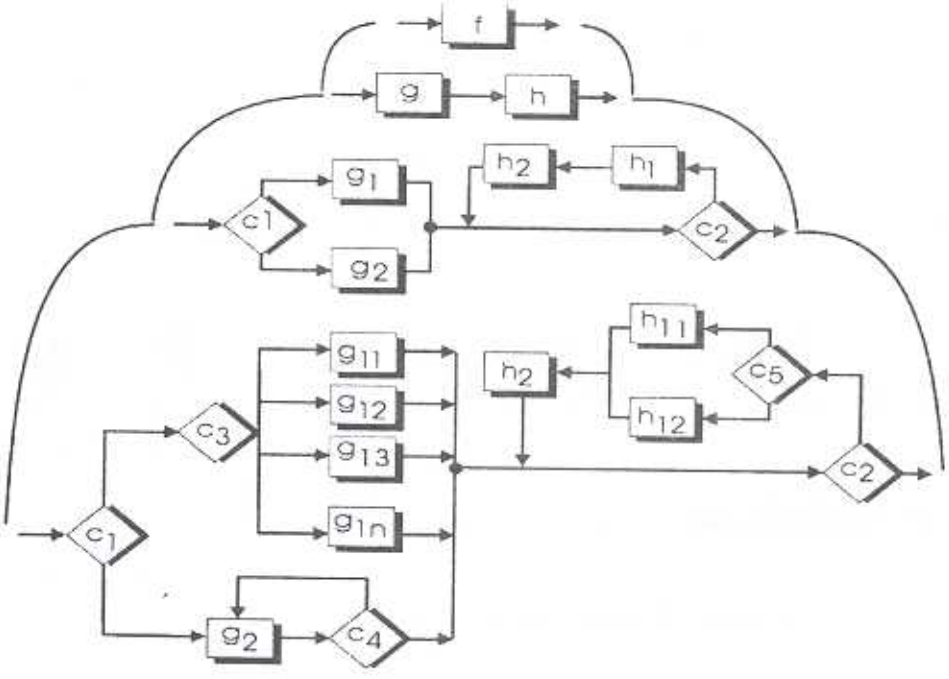
- **Whenever <c> is True Does n Followed by M do m; and Whenever <c> is False, does skipping the loop still do m?**

(هرگاه c درست باشد m به دنبال m آمده و هرگاه c نادرست باشد چشم‌پوشی از m در حلقه رخ می‌دهد؟)

هرگاه یک جعبه شفاف در سطح جزئیات بعدی قرار گیرد، شرایط درستی مورد اشاره فوق به‌کار می‌روند.

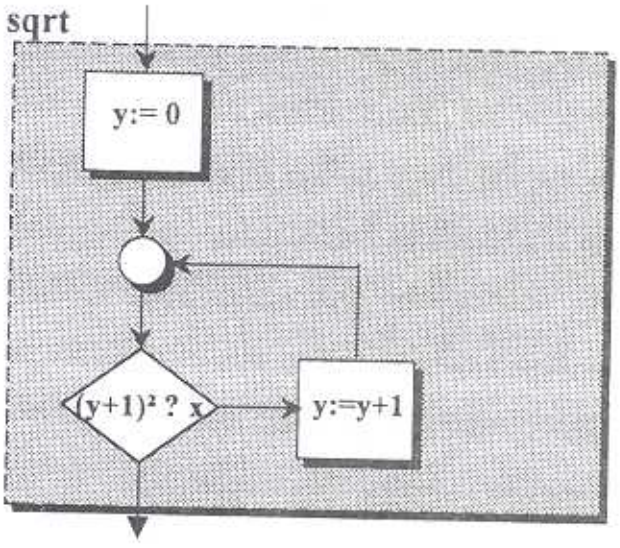
نکته مهم مورد توجه این است که استفاده از ساختارهای برنامه‌نویسی ساختیافته تعداد آزمونهای مربوط به‌درستی کار را که باید انجام گیرند، محدود می‌سازد. یک شرط برای زنجیره‌ها چک شده است، دو شرط برای if - then - else آزمون شده و سه شرط برای حلقه‌ها تأیید شده است.

برای توضیح اعتبار طراحی رویه ای از مثالی ساده بهره می بریم که اول مرتبه توسط لینگر، میلز و ویت [LIN79][1] استفاده شد. مقصود طراحی و تایید یک برنامه کوچک است که جزء صحیح ریشه دوم عدد صحیحی چون x را (که y می نامیم) به دست آورد. طراحی رویه ای به صورت فلوچارت شکل ۲۶-۷ ارائه گردیده است.

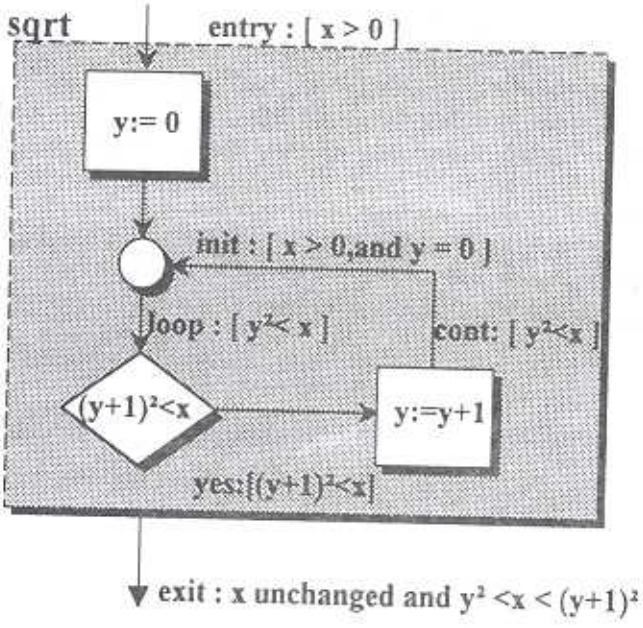


شکل ۲۶-۶ پالایش مرحله ای (گام به گام)

1.inger,R.M.



شکل ۲۶ـ۷ محاسبه جزء صحیح ریشه دوم [LIN79]



شکل ۲۶ـ۸ اثبات صحت طراحی [LIN79]

برای تأیید درستی این طراحی باید شرایط ورودی و خروجی را همان‌گونه که در شکل ۲۶-۸ آمده تعریف کنیم. شرط ورودی بیان می‌دارد که X باید بزرگتر یا مساوی صفر باشد. شرط خروجی نیازمند این است که X بدون تغییر باقی‌مانده و مقداری داشته باشد که در دامنه این شکل آمده است. به‌منظور اثبات

درستی این طرح، لازم است درستی شرایط *exit, yes, cont, loop, init* را که در شکل ۲۶-۸ آمده در همه موارد اثبات کنیم. گاهی به اینها استدلال‌های فرعی[1] می‌گویند.

۱- شرط init نیازمند این است که [x≥o, y= o] باشد. بر اساس شرایط مسئله، شرط ورودی درست فرض می‌شود.[2] بنابراین اولین بخش از شرط init یعنی x≥o برقرار است. با اشاره به فلوچارت، این وضعیت بلافاصله به‌دنبال شرط init می‌آید که y=o را قرار می‌دهد. بنابراین، دومین بخش از شرط init نیز برقرار می‌شود. بنابراین init درست است.

برای اثبات صحت یک طراحی شما ابتدا باید تمام شرایط را شناسایی کنید و آنگاه اثبات کنید که هرکدام مقداری دو ارزشی خواهند داشت. اینها " زیر اثباتها " نامیده می شوند.

۲- شرط loop را ممکن است از دو نظر مورد بررسی قرار داد: ۱) مستقیماً از init (در این مورد شرط loop مستقیماً برقرار شده است) یا از طریق جریان کنترلی که از شرط cont می‌گذرد. از آن‌جا که شرط cont مشابه شرط loop است، بدون توجه به مسیر جریانی که به آن منتهی می‌شود، loop درست است.

۳- شرط cont تنها وقتی برقرار می‌شود که مقدار y به میزان 1 افزایش یابد. علاوه بر این، مسیر جریان کنترلی که به cont منتهی می‌شود تنها در صورتی می‌تواند تحریک شود که شرط yes درست باشد. بنابراین اگر $x \leq (y+1)^2$ باشد این‌طور می‌شود که $y^2 \leq x$ است. شرط cont برقرار است.

۴- شرط yes به‌صورت منطق شرطی نشان داده شده، آزمون می‌شود. بنابراین، شرط yes وقتی درست باشد جریان کنترلی در طول مسیر نشان داده شده حرکت می‌کند.

۵- شرط exit ابتدا نیازمند این است که x بدون تغییر بماند. بررسی این طراحی نشانگر این است که به‌نظر می‌رسد x هیچ کجا در سمت چپ یک اپراتور تخصیصی ظاهر نمی‌شود. هیچ‌گونه فراخوانی تابعی وجود ندارد که از x استفاده کند. بنابراین بدون تغییر می‌ماند. از آن‌جا که آزمون شرطی $x \leq (y+1)^2$ از رسیدن به شرط

1.subproofs

۲. مقدار منفی برای ریشه دوم در این متن بی معناست.

Subproofs:

```
[f1]                    f1 = [DO g1; g2; [f2] END] ?
 DO
   g1
   g2
   [f2]                 f2 = [WHILE p1 DO [f3] END] ?
    WHILE
      p1
    DO [f3]             f3 = [DO g3; [f4]; g8 END] ?
      g3
      [f4]              f4 = [IF p2; THEN [f5] ELSE [f6] END] ?
      IF
        p2
      THEN [f5]         f5 = [DO g4; g5 END ] ?
        g4
        g5
      ELSE [f6]         f6 = [DO g6; g7 END] ?
        g6
        g7
      END
      g8
   END
END
```

شکل ۲۶ـ۹ یک طراحی با زیر اثبات‌ها (subproofs) [LIN94]

exit باز می‌ماند، به این شکل دنبال می‌شود $(y+1)^2 \leq x$ علاوه بر این، باید شرط loop هنوز درست بدست باشد (یعنی $y^2 \leq x$). بنابراین، $(y+1)^2 \leq x$ و $y^2 \leq x$ را می‌توان تلفیق کرد تا شرط exit برقرار شود.

باید هر چه بیشتر مطمئن شویم که حلقه خاتمه می‌یابد. بررسی شرط loop نشانگر این امر است که از آن‌جا که y افزایش یافته و $x \geq 0$ است، باید حلقه نهایتاً پایان یابد.

پنج مرحله فوق دلایل درستی طرح الگوریتم شکل ۲۶-۷ هستند. اکنون مطمئن هستیم که طراحی واقعاً" جزء صحیح ریشه دوم را محاسبه می‌کند.

یک روش ریاضی سخت‌تر برای تأیید طراحی نیز امکان دارد. در هر حال، بحث در مورد این موضوع فراتر از دامنه این کتاب است. خوانندگان علاقه‌مند به [LIN79] رجوع کنند.

۲۶-۳-۲ مزایای مترتب بر تعیین صحت طراحی[1]

پرسش

با اعمال تعیین صحت چه چیز حاصل می شود ؟

تأیید درستی هر اصلاحیه در طرح جعبه شفاف دارای چند مزیت مشخص است. لینگر این مزایا را به‌صورت زیر توصیف می‌کند:[LIN94] [1]

- **این کار تأیید را به یک فرآیند محدود کاهش می‌دهد.** شیوه برگزیده و متوالی که ساختارها در آن در یک جعبه شفاف سازمان‌دهی می‌شوند سلسله مراتبی را تعریف می‌کند که نمایان‌گر شرایط درستی است که باید تأیید شوند. یک قاعده و اصل کلی در مورد جایگزینی، به ما اجازه می‌دهد کارکردهای موردنظر را به‌وسیله پالایش‌هایی در سلسله مراتب اثبات فرعی جایگزین کنیم. مثلاً، اثبات فرعی تابع موردنظر F_1 در شکل ۲۶-۹ نیازمند اثبات این مطلب است که ترکیب عملیات g_1 و g_2 با تابع موردنظر F_2 دارای اثری یکسان بر اطلاعاتی مثل F_1 است. توجه داشته باشید که F_2 جایگزین تمام جزئیات پالایش آن در اثبات است. این جایگزینی مبحث اثبات را در ساختار کنترلی در دست اقدام قرار می‌دهد. در حقیقت به مهندس نرم‌افزار امکان می‌دهد که اثبات را به هر صورتی انجام دهد.

علی رغم تعداد بسیار زیاد مسیرهای اجرای یک برنامه، تعداد گامهای اثبات صحت اندک خواهد بود.

- **ممکن نیست که بر تأثیر مثبتی که تحقیق در مورد یک فرآیند محدود دارای تأثیر کیفی داشته و آن را کاهش می‌دهد، تأکید زیاد ننمود.** با این‌که نه همه، اما اکثر برنامه‌های جزئی اساساً تعداد مسیرهای اجرایی نامحدودی را نشان می‌دهند، اما می‌توان آنها را در تعداد مراحل معدودی شناسایی و تأیید نمود.

- **این کار به تیم‌های "اطاق پاک" اجازه می‌دهد هر خط از طراحی و برنامه را از نظر درستی بسنجند.** تیم‌ها می‌توانند این کار را از طریق تحلیل گروهی بحث در مورد پایه و اساس درست انجام داده و می‌توانند اثباتهای مکتوبی وقتی‌که اطمینان بیشتری در مورد سیستم مهم مأموریتی یا طول عمر لازم است، ارائه دهند.

- **این کار منجر به میزان معایبی نزدیک به صفر می‌شود.** در طول بازنگری تیمی، هر شرط درست بوده، بنابراین اشتباه تنها در صورتی ممکن است که هر عضو تیم به اشتباه چیزی را تأیید کند. شرط این توافق مشخص بر اساس رؤیت تحقیقات فردی است که منجر به پیدایش نرم‌افزار شده که قبل از اجرا بدون اشتباه یا با حداقل اشتباه است.

- **پیشرونده است.** هر سیستم نرم‌افزاری بدون توجه به این‌که چقدر بزرگ است دارای یک سطح برتر، شیوه‌های جعبه شفاف متشکل از ساختارهای تسلسلی، تناوب و تکراری است. هر یک از این موارد معمولاً یک سیستم فرعی دیگر با هزاران خط برنامه را تحریک نموده و هر کدام از این سیستم‌های فرعی دارای توابع و شیوه‌های مخصوص به‌خود است. بنابراین شرایط درستی این ساختارهای کنترلی سطح بالا

---

۱. این بخش و شکل های ۲۶-۷ تا ۲۶-۹ مطابق با [LIN94] (با کسب اجازه) ارائه گردیده است.

تا حدودی شبیه به موارد مربوط به ساختارهای سطح پایین هستند. اثبات درستی موارد سطح بالا ممکن است وقت زیادتر بگیرد که ارزشش را هم دارد اما تئوری بیشتری صرف آن نمی‌شود.

- **این عمل، برنامه بهتری نسبت به آزمون واحد ارائه می‌دهد.** آزمون واحد تنها اثرات احرای مسیرهای آزمونی انتخابی را از میان مسیرهای متعدد احتمالی، بررسی می‌کند. با مبدأ قرار دادن این اثبات روی تئوری تابع، رهیافت "اطاق پاک" می‌تواند هرگونه نقص احتمالی روی همه داده‌ها را شناسایی کند، زیرا در عین حالی‌که برنامه مسیرهای اجرایی متعددی دارد تنها یک عملکرد دارد. هم‌چنین تأیید درستی بسیار کارآمدتر از آزمون واحد است. اکثر شرایط مربوط به تأیید را می‌توان ظرف چند دقیقه بررسی کرد اما آزمونهای واحد از نظر آماده‌سازی، اجرا و چک کردن وقت زیادی می‌گیرند.

نکته مهم مورد توجه این است که تأیید طراحی باید نهایتاً در مورد خود برنامه منبع اعمال گردد. در این‌جا، به آن اغلب اثبات صحت و درستی[2] یا تأیید درستی می‌گوییم.

۲۶-۴ آزمون اطاق پاک

راهبرد و تاکتیک‌های آزمون "اطاق پاک" اساساً متفاوت از رهیافتهای آزمون متعارف هستند. روش‌های قراردادی مجموعه‌ای از موارد آزمون را برای مشخص کردن خطاهای برنامه نویسی و طراحی به‌کار می‌گیرند. هدف از آزمون "اطاق پاک" عبارتست از اربابی نیازمندیهای نرم‌افزاری به‌وسیله تشریح این‌که یک نمونه آماری از موارد کاربرد با موفقیت به انجام رسیده است.

۲۶-۴-۱ آزمون استفاده آماری

کاربر برنامه کامپیوتری به‌ندرت نیاز به درک جزئیات فنی طراحی دارد. رفتار مشهود برنامه برای کاربر برگرفته از ورودیها و اتفاقاتی است که اغلب به‌وسیله کاربر تولید می‌شوند. اما در سیستم‌های پیچیده، این طیف احتمالی ورودیها و رویدادها می‌تواند بسیار وسیع باشد. زیرمجموعه موارد کاربردی که به حد کافی رفتار و نحوه عمل برنامه را مشخص می‌کنند، چیست؟ این اولین سؤالی است که در این آزمون مورد خطاب قرار می‌گیرد.

آزمون کاربرد آمار نرم‌افزار به شیوه‌ای است که کاربران می‌خواهند با آن کار کنند.[LIN94] برای رسیدن به آن، تیم‌های آزمون اطاق پاک[3] (یا تیم‌های گواهی تأیید)[4] باید توزیع احتمال کاربرد را در مورد نرم‌افزار تعیین کنند. مشخصات (جعبه سیاه) در مورد هرگونه افزایشی در نرم‌افزار تحلیل می‌گردد تا مجموعه‌ای از محرک‌ها (ورودیها یا رویدادها) را که باعث تغییر عملکرد نرم‌افزار می‌شوند، مشخص شود. بر

1 Linger,R.
2.correctness verification
3.cleanroom testing teams
4.certifications teams

اساس مصاحبه‌هایی با کاربران بالقوه، ایجاد طرح‌های کاربری و درک کلی از دامنه کاربرد، یک احتمال استفاده به هر یک از محرک‌ها تخصیص داده می‌شود.

موارد آزمونی تولید شده برای هر محرک[1] طبق احتمال توزیع کاربرد آن است. برای تشریح این مطلب، سیستم امنیتی خانهٔ امن را در نظر بگیرید که در فصول پیشین تشریح شد. مهندسی نرم‌افزاری "اطاق باک" برای ایجاد اصلاح و پیشرفت نرم‌افزاری ارائه می‌شود که ارتباط متقابل کاربر با صفحه کلید سیستم را مدیریت می‌کند. در این مورد پنج محرک شناسایی شده‌اند. برای سهل کردن انتخاب موارد آزمونی، این احتمالات در فواصلی بین ۱ تا ۹۹ طرح شده و در جدول زیر آمده‌اند:

Program Stimulus	Probability	Interval
Arm/disarm (AD)	50%	1-49
Zone set (ZS)	15%	50-63
Query (Q)	15%	64-78
Test (T)	15%	79-94
Panic alarm	5%	95-99

برای تولید رشته‌ای از موارد کاربرد به‌صورت آزمونی که با توزیع احتمال کاربرد مطابقت داشته باشد، یک سری اعداد تصادفی بین ۱ تا ۹۹ تولید می‌شوند. شماره تصادفی با فاصله توزیع

احتمالی فوق‌الذکر ارتباط دارد. بنابراین، توالی موارد کاربرد به‌صورت تصادفی اما مرتبط با احتمال وقوع هر یک از محرک‌ها به‌صورت درست، تعریف شده‌اند. مثلاً توالی رقم‌های زیر را که به‌صورت تصادفی است در نظر بگیرید:

13-94-22-24-45-5
81-19-31-69-45-9
38-21-52-84-86-4

با انتخاب محرک مناسب بر اساس فواصل توزیع نشان داده شده در جدول فوق موارد کاربرد زیر بدست می‌آیند:

AD- T -AD- AD-AD- ZS
T-AD-AD-AD-Q-AD-AD
AD-AD-ZS- T - T -AD

تیم آزمون موارد کاربرد فوق را اجرا نموده و عملکرد نرم‌افزار را در مقابل مشخصه سیستم ارزیابی می‌کنند. زمان‌بندی آزمون‌ها ثبت می‌شود به‌طوری‌که فواصل زمانی تعیین می‌گردند. با استفاده از زمان فواصل، تیم تأیید و گواهی می‌توانند میانگین زمانی شکست را محاسبه کنند. اگر یک‌سری آزمونهای

---

[1]. ابزارهای خودکار باری این امر موجودند. برای اطلاعات بیشتر به [DYE92] مراجعه کنید.

طولانی بدون شکست صورت گرفتند MTTF درحد پایین و قابلیت اطمینان نرم‌افزار در سطح بالایی است.

۲۶-۴-۲ تایید و تضمین

فنون تأیید و آزمون که پیش‌تر در این فصل مورد بحث قرار گرفت منجر به جزءهای نرم‌افزاری می‌شوند که می‌توان آنها را گواهی نمود. در بطن روش مهندسی نرم‌افزاری "اطاق پاک" این گواهی دلالت بر این دارد که می‌توان قابلیت اطمینان را در مورد جزء برنامه تأیید و گواهی کرد.

تأثیر بالقوه اجزایی از نرم‌افزار که قابل تأیید هستند فراتر از یک پروژه "اطاق پاک" است. اجزای قابل استفاده مجدد را می‌توان به‌همراه سناریوهای‌کاربردشان، محرک‌های برنامه و توزیع‌های احتمالی ذخیره نمود. هر جزء یک قابلیت اطمینان تأیید شده دارد که تحت سناریو کاربردی و نحوه آزمون، توصیف شده است. این اطلاعات برای کسانی که قصد دارند از این اجزاء استفاده کنند بسیار با ارزش است.

پرسش

یک جزء نرم افزار را چگونه تضمین و تصدیق نماییم؟

روش گواهی پنج مرحله [WOH94][1] دارد:

۱- باید سناریوهای کاربرد ایجاد شوند.

۲- یک شرح کاربرد مشخص گردد.

۳- موارد آزمون از روی این شرح ایجاد گردند.

۴- آزمونهایی اجرا شده و اطلاعات مربوط به شکست کار ثبت و تحلیل شوند.

۵- قابلیت اطمینان محاسبه و گواهی می‌گردد.

مراحل ۱ تا ۴ در بخش پیشین مورد بحث قرار گرفتند. در این بخش روی گواهی قابلیت اطمینان متمرکز می‌شویم.[POO93][2]

گواهی مهندسی نرم‌افزاری "اطاق پاک" نیازمند ایجاد سه مدل است:

مدل نمونه[3]. آزمون نرم‌افزار، m مورد آزمون تصادفی اجرا نموده و اگر مشکلی رخ نداد یا تعداد معینی شکست در کار رخ داد، تأیید می‌شود. ارزش و مقدار m به‌صورت ریاضی تعیین می‌گردد تا مطمئن شویم قابلیت اطمینان لازم حاصل شده است.

مدل جزء[4]. هر سیستم، متشکل از n جزء است که باید تأیید شوند. این مدل تحلیلگر را قادر می‌سازد که احتمال این‌که هر جزء «i» قبل از تکمیل کار منهدم شود را تعیین کند.

مدل تایید. قابلیت اطمینان کلی سیستم، تأیید و گواهی می‌شود.

---

1. Wohlin , C.and P. Poore,J.H.
2. Sampling Model
3. Component Model

با تکمیل آزمون[1] کاربرد آمار، تیم تأیید دارای اطلاعات لازم برای ارائه نرم‌افزاری است که دارای MTTF تأیید شده‌ای است که با استفاده از این مدل‌ها محاسبه شده‌اند.

بحث دقیق‌تر محاسبه نمونه‌گیری، جزء و مدل‌های گواهی، فراتر از دامنه این کتاب است. خوانندگان علاقه‌مند را به [MUS87][2] و [Cur86][3] و [Poo93][1] ارجاع می‌دهیم.

۵-۲۶ خلاصه

مهندسی نرم‌افزاری "اطاق پاک" یک رهیافت رسمی تولید نرم‌افزار است که می‌تواند منجر به نرم‌افزاری شود که دارای کیفیت بالایی است. در این کار از مشخصات ساختار جعبه برای تحلیل و مدل‌سازی طراحی استفاده شده و به‌جای آزمون به‌عنوان مکانیزم اولیه برای یافتن و اصلاح خطاها بر تأیید درستی(اثبات‌صحت) تأکید می‌شود. آزمون استفاده آمار برای ارائه اطلاعات میزان عدم موفقیت به‌کار می‌رود که برای گواهی قابلیت اطمینان نرم‌افزار لازم است.

رهیافت "اطاق پاک" با مدل‌های تحلیل و طراحی آغاز می‌شود که از نمایش ساختار جعبه یا باکس استفاده می‌کنند. هر جعبه در برگیرنده سیستم (یا جنبه‌هایی از سیستم) در سطح تجرید به‌خصوصی است. جعبه‌های سیاه برای نمایش رفتار قابل رؤیت برونی سیستم استفاده می‌شوند. جعبه‌های وضعیت اطلاعات موقعیت و عملیات را در اختیار دارند. جعبه شفاف برای مدل‌سازی طراحی رویه‌ای استفاده می‌شود که تحت تأثیر داده‌ها و عملیات جعبه وضعیت است.

تأییدیه درستی زمانی به‌کار می‌رود که طراحی ساختار جعبه کامل شده است. طراحی رویه‌ای برای هر جزء از نرم‌افزار شامل یکسری زیر توابع است. برای اثبات درستی هر یک از این زیر توابع، شرایط خروج برای هر زیر تابع تعریف شده و مجموعه‌ای از استدلال‌های فرعی به‌کار گرفته می‌شوند. اگر هر شرط خروجی برقرار شود، باید طراحی درست باشد.

وقتی این تأییدیه درستی (یا اثبات صحت) به‌اتمام رسید، آزمون کاربرد آماری آغاز می‌شود. برخلاف آزمون قراردادی مهندسی نرم‌افزاری "اطاق پاک" بر واحد یا آزمون یکپارچگی تأکید ندارد. بلکه، نرم‌افزار با تعریف مجموعه‌ای از سناریوهای کاربردی، مورد آزمون واقع می‌شود که احتمال استفاده از هر سناریو را تعیین نموده و سپس آزمونهای تصادفی را تعیین می‌کند که با این احتمالات مطابقت دارند. خطا ثبت می‌شود که نتایج آن همراه با مدلهای نمونه‌سازی، جزء و گواهی، امکان محاسبه ریاضی قابلیت اطمینان پروژه را برای جزء نرم‌افزاری، مقدور می‌سازد.

فلسفه "اطاق پاک"، رهیافت پیچیده‌ای در مهندسی نرم‌افزار است. این یک مدل فرآیند نرم‌افزاری است که بر تأیید درستی ریاضی و گواهی قابلیت اطمینان به نرم‌افزار تأکید دارد. آخر خط و پایان ماجرا،

1.Certification Model
2.Musa ,J.D.
3.Curritt , P.A.

حداقل میزان شکست برنامه است که رسیدن به آن بدون استفاده از روش‌های رسمی ناممکن یا بسیار سخت است.

1.Poore,J.H.

مسایل و نکاتی برای تفکر و تعمق بیشتر

۲۶-۱ اگر شما مجبور باشید که یکی از جنبه‌های اطاق پاک مهندسی نرم‌افزاری را که تفاوت اساسی از جنبه‌های متداول یا مهندسی نرم‌افزاری شیءگرا دارد را برگزینید، آن چه خواهد بود؟

۲۶-۲ چگونه مدل روند افزایشی و جواز همکاری برای تولید نرم‌افزاری با کیفیتی برتر عمل می‌کند؟

۲۶-۳ با استفاده از ساختار اختصاصی «First-Pass» (اولین مسیر) تجزیه را شرح داده و مدلی را برای سیستم ایمنی خانه طراحی کنید.

۲۶-۴ از ساختار اختصاصی برای قسمتی از سیستم PHTRS معرفی شده در قسمت ۲۱-۱۳ استفاده نمایید.

۲۶-۵ از ساختار اختصاصی برای سیستم E-mail معرفی شده در قسمت ۲۱-۱۵ استفاده نمایید.

۲۶-۶ الگوریتم bubble sort در مثال زیر تعریف شده است:

```
Procedure        bubble sort
:
end
```

این طرح را به زیر برنامه‌های کوچک‌تر تقسیم کرده و گروهی از شرط‌ها را که شما را قادر به اثبات صحت این الگوریتم می‌کند، تعریف کنید.

۲۶-۷ درستی اثبات دلایل را در مرتب سازی حبابی (bubble sort) مطرح شده در مسأله ۲۶-۶ مستند سازید.

۲۶-۸ اجزاء برنامه‌ای که خود در موضوع دیگری طراحی کرده‌اید (با توسط استادتان اختصاص داده شده) انتخاب کرده و دلایل اثبات درستی را برای آن شرح دهید.

۲۶-۹ برنامه‌ای را که شما به‌طور منظم از آن استفاده می‌کنید (راه‌انداز پست الکترونیک، پردازش لغت و برنامه Spreadsheet) برگزینید. گروهی از شرح مختصر برنامه‌ها ایجاد کنید. احتمالات استفاده از هر شرح مختصر را معرفی کرده و سپس برنامه را به‌حالت پویا (Stimuli) و قابل نشر هم‌چون نمونه‌ای که در بخش ۲۶-۴-۱ آورده شد، گسترش می‌دهید.

۲۶-۱۰ برای این‌که برنامه‌های پویا و قابل انتشار در مسأله ۲۶-۹ داشته باشیم، از تعدادی مولد تصادفی برای گسترش گروهی از موارد آزمون (Test Case) استفاده شده در موارد آماری، استفاده نمایید.

۲۶-۱۱ معنای تأیید را در متون مهندسی نرم‌افزار اطاق پاک، به زبان خودتان شرح دهید.

۲۶-۱۲ متن کوتاهی راجع به استفاده روش‌های ریاضی برای معرفی مدل‌های تأییدیه که در بخش ۲۶-۴-۲ به‌طور مختصر آمده است. بنویسید. برای آغاز کار از [MUS 87]، [CUR86]، [POO93] استفاده نمایید.

فهرست منابع و مراجع

[CUR86] Curritt, P.A., M. Dyer, and H.D. Mills, "Certifying the Reliability of Software," *IEEE Trans, Software Engineering*, vol. SE-12, no. 1, January 1994.

[DYE92] Dyer, M., *The Cleanroom Approach to Quality Software Development*, Wiley,

1992.

[HAU94] Hausler, P.A., R. Linger, and C. Trammel, "Adopting Cleanroom Software Engineering with a Phased Approach," *IBM Systems journal*, vol. 33, no. 1, January 1994, pp. 89-109.

[HEN95] Henderson, J., "Why Isn't Cleanroom the Universal Software Development Methodology?" *Crosstalk*, vol. 8, No.5, May 1995, pp. 11-14.

[HEV93] Hevner, A.R. and H.D. Mills, "Box Structure Methods for System Development with Objects," *IBM Systems journal*, vol. 31, no.2, February 1993, pp. 232-251.

[LIN79] Linger, R.M., H.D. Mills, and B.I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, 1979.

[LIN88] Linger, R.M. and H.D. Mills, " A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," *Proc. COMPSAC* '88, Chicago, October 1 988.

[LIN94] Linger, R., "Cleanroom Process Model," *IEEE Software*, vol. 11, no. 2, March 1994, pp. 50-58.

[MIL87] Mills, H.D., M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, vol. 4, no. 5, September 1987, pp. 19-24.

[MIL88] Mills, H.D., "Stepwise Refinement and Verification in Box Structured Systems," *Computer*, vol. 21, no. 6, June 1988, pp. 23-35.

[MUS87] Musa, J.D., A. Iannino, and K. Okumoto, *Engineering and Managing Software with Reliability Measures*, McGraw-Hill, 1987.

[POO88] Poore, J.H. and H.D. Mills, "Bringing Software Under Statistical Quality Control," *Quality Progress*, November 1988, pp. 52-55.

[POO93] Poore, J.H., H.D. Mills, and D. Mutchler, "Planning and Certifying Software System Reliability," *IEEE Software*, vol. 10, no. 1, January 1993, pp. 88-99.

[WOH94] Wohlin, C. and P. Runeson, "Certification of Software Components," *IEEE Trans. Software Engineering*, vol. SE-20, no. 6, June 1994, pp. 494-499.

خواندنیهای دیگر و منابع اطلاعاتی

Prowell et al. *(Cleanroom Software Engineering: Technology and Process*, Addison-Wesley, 1999) provides an in-depth treatment of all important aspects of the cleanroom approach. Useful discussions of cleanroom topics have been edited by Poore and Trammell *(Cleanroom Software Engineering: A Reader*, Blackwell Publishing, 1996). Becker and Whittaker *(Cleanroom Software Engineering Practices*, Idea Group Publishing, 1996) present an excellent overview for those who are unfamiliar with clean room practices.

The Cleanroom Pamphlet (Software Technology Support Center, Hill AF Base, April 1995) contains reprints of a number of important articles. Linger [LIN94] produced one of the better introductions to the subject. *Asset Source for Software Engineering*

Technology, ASSET, (United States Department of Defense) offers an excellent six volume set of *Cleanroom Engineering Handbooks*. ASSET can be contacted at info@source.assetcom. Lockheed Martin's *Guide to the Integration of Object-Oriented Methods and Cleanroom Software Engineering* (1997) contains a generic cleanroom process for 00 systems and is available at http://www.asset.com/stars/loral/cleanroom/oo/ guidhome.htm.

Unger and Trammell *(Cleanroom Software Engineering Reference Model*, SEI Technical Report CMU/SEI-96- TR-022, 1996) have defined a set of 14 cleanroom processes and 20 work products that form the basis for the SEI CMM for cleanroom software engineering (CMU /SEI -96- TR -023).

Michael Deck of Cleanroom Software Engineering has prepared a bibliography on cleanroom topics. Among the references are the following:

General and Introductory

Deck, M.D., "Cleanroom Software Engineering Myths and Realities, " *Qualify Week* 1997, May
1997.

Deck, M.D. and J. A. Whittaker, "Lessons Learned from Fifteen Years of Cleanroom Testing,"

Software Testing, Analysis, and Review (STAR) '97, San Jose, CA, May 5-9, 1997.

Lokan, C.J., "The Cleanroom Process for Software Development," *The Australian Computer*

Journal, vol. 25, no. 4, November 1993.

Linger, Richard C., "Cleanroom Software Engineering for Zero-Defect Software," *Proc 15th*

International Conference on *Software Engineering*, May 1993.

Keuffel, W., "Clean Your Room: Formal Methods for the '90s," *Computer Language*, July 1992,
pp. 39-46.

Hevner, A.R., S.A. Becker, and L.B. Pedowitz, "Integrated CASE for Cleanroom Development,"

IEEE Software, March 1992, pp. 69-76.

Cobb, R.H. and H.D. Mills, "Engineering Software under Statistical Quality Control," *IEEE Soft-*
ware, November 1990, pp. 44-54.

Management Practices

Becker, S.A., Deck, M.D., and Janzon, T., "Cleanroom and Organizational Change," *Proc. 14th*

Pacific Northwest Software Qualify Conference, Portland, OR, October 29-30, 1996.

Unger, R. C., "Cleanroom Process Model," *IEEE Software*. March 1 994, pp. 50-58.

Unger, R. C. and R.A. Spangler, "The IBM Cleanroom Software Engineering Technology Trans-
fer Program, " *Sixth SEI Conference* on *Software Engineering Education*, San Diego, CA, Octo-

ber 1992.

Specification, Design, and Review

Deck, M.D., "Cleanroom and Object-Oriented Software Engineering: A Unique Synergy," 1996

Software Technology Conference, Salt Lake City, UT, April 24, 1996.

Deck, M.D., "Using Box Structures to Link Cleanroom and Object-Oriented Software Engi-

neering," Technical Report 94.01 b, Cleanroom Software Engineering, 1 994.

Dyer, M., "Designing Software for Provable Correctness: The Direction for Quality Software,"

Information and Software Technology, vol. 30 no. 6, July-August 1988, pp. 331-340.

Testing and Certification

Dyer, M., "An Approach to Software Reliability Measurement," *Information and Software Tech*

nology, vol. 29 no. 8, October 1987, pp. 415-420.

Head, G.E., "Six-Sigma Software Using Cleanroom Software Engineering Techniques," *Hewlett-*

Packard journal, June 1994, pp. 40-50.

Oshana, R., "Quality Software via a Cleanroom Methodology," *Embedded Systems Program-*

ming, September 1 996, pp. 36-52.

Whittaker, J.A. and M.G. Thomason, "A Markov Chain Model for Statistical Software Testing,"

IEEE Trans. Software Engineering, vol. SE-20 October 1994, pp. 812-824.

Case Studies and Experience Reports

Head, G.E., "Six-Sigma Software Using Cleanroom Software Engineering Techniques," *Hewlett-*

Packard journal, June 1994, pp. 40-50.

Hevner, A.R. and H.D. Mills, "Box-Structured Methods for Systems Development with Objects,"

IBM Systems journal, vol. 32, no. 2, 1993, p. 232-251.

Tann, L-G., "OS32 and Cleanroom," *Proc. First Annual European Industrial Symposium on Clean*

room Software Engineering, Copenhagen, Denmark, 1993, pp. 1-40.

Hausler, P.A., "A Recent Cleanroom Success Story: The Redwing Project," *Proc. 17th Annual*

Software Engineering Workshop, NASA Goddard Space Flight Center, December 1992.

Trammel, C.J., L.H. Binder, and C.E. Snyder, "The Automated Production Control Documen

tation System: A Case Study in Cleanroom Software Engineering," *ACM Trans. on Software*

Engineering and Methodology, vol. 1, no. 1, January 1992, pp. 81-94.

Design verification via proof of correctness lies at the heart of the cleanroom approach. Books by Baber *(Error-Free Software*, Wiley, 1991) and Schulmeyer *(Zero Defect Software*, McGraw-Hill, 1990) discuss proof of correctness in considerable detail.

A wide variety of information sources on cleanroom software engineering and related subjects is available on the Internet. An up-to-date list of World Wide Web references that are relevant to cleanroom software engineering can be found at the SEPA Web site:

http://www.mhhe.com/engcs/compsci/pressman/resources/cleanroom.mhtml