روش‌های صوری در مهندسی نرم‌افزار

مولف:

دکتر محمد علی ترکمانی

سرشناسه	:	ترکمانی، محمدعلی، ۱۳۵۴ -
عنوان و نام پدیدآور	:	روش‌های صوری در مهندسی نرم‌افزار/ مولف محمد علی ترکمانی.
مشخصات نشر	:	مشهد: ارسطو: سامانه اطلاع رسانی چاپ و نشر ایران،۱۳۹۵.
مشخصات ظاهری	:	۲۶۴ص.: مصور، جدول، نمودار .
شابک	:	۲۵۰۰۰۰ ریال: 978-600-432-025-2
وضعیت فهرست نویسی	:	فاپا
یادداشت	:	کتابنامه: ص. ۲۶۲- ۲۶۴.
موضوع	:	نرم‌افزار -- مهندسی
موضوع	:	Software engineering
رده بندی کنگره	:	QA۷۶/۷۵۸/ت۴ر۹ ۱۳۹۵
رده بندی دیویی	:	۰۰۵/۱
شماره کتابشناسی ملی	:	۴۳۳۸۶۵۶

نام کتاب : روش های صوری در مهندسی نرم افزار

موضوع: توصیف، پالایش و اثبات با Z

مولف : دکتر محمد علی ترکمانی

ناشر : ارسطو (با همکاری سامانه اطلاع رسانی چاپ و نشر ایران)

صفحه آرایی ، تنظیم و طرح جلد : محمدعلی ترکمانی و علی بیات

تیراژ : ۱۰۰۰ جلد

نوبت چاپ : سوم- ۱۳۹۹

تعداد صفحات: ۲۵۵ صفحه

چاپ : مدیران

قیمت : ۴۰۰۰۰ تومان

شابک : ۲ – ۰۲۵– ۴۳۲– ۶۰۰ – ۹۷۸

تلفن مرکز پخش : ۰۹۱۵۹۲۲۴۳۲۰ – ۰۹۱۷۷۱۶۴۹۴۰

این اثر مشمول قانون حمایت از مولفان و منصفان و هنرمندان است. هر کس تمام یا قسمتی از این اثر را بدون اجازه مولف نشر یا پخش یا عرضه کند، مورد پیگرد قانونی قرار خواهد گرفت.

فهرست مطالب

فصل اول: مقدمه‌ای بر مهندسی نرم‌افزار و روش‌های صوری .. ۱۵

۱-۱-روش‌های صوری ۱۵

۲-۲- مولفه‌های یک روش رسمی ۲۱

۳-۲-تمرین‌ها ۲۱

فصل دوم: حیطه‌های به کارگیری روش‌های صوری ۲۳

۱-۲- مقدمه ۲۳

۲-۲-انواع ویژگی‌های یک سیستم ۲۴

۳-۲-درست‌یابی ۲۵

۴-۲-تولید برنامه صوری ۲۶

۱-۲-۲- روش تقلید ۲۷

۲-۲-۲- روش پالایش ۲۷

۳-۲-۲- روش ساختی ۲۹

۳-۲-روش‌ها یا پارادایم‌های مختلف توصیف صوری ۳۱

۱-۳-۲-ویژگی های زبان Z ۳۲

۴-۲-مقایسه زبانهای رسمی Z، B و VDM ۳۴

۱-۴-۲- مقایسه بر اساس مشخصات پایه ۳۴

۲-۴-۲-مقایسه بر اساس همزمانی ۳۴

۳-۴-۲-مفاهیم شیءگرا ۳۶

۴-۴-۲-پشتیبانی ابزاری ۳۶

۵-۴-۲-تولید کد ۳۷

۵-۲-مقایسه روش‌های مهندسی صوری(FEM) و روش‌های صوری(FM) ۳۸
۶-۲-تمرین ها ۴۰

فصل سوم: کاربردها و تجربه‌های صنعتی ۴۳

۱-۳-استفاده از روش‌های صوری در خط تولید نرم‌افزار ۴۳
۲-۳-استفاده از روش‌های صوری در پروژه‌های فضایی ۴۵
۳-۳-استفاده از روش‌های صوری در توسعه نرم‌افزارهایی که ایمنی در آنها مسئله‌ای بحرانی است ۴۶
۴-۳-استفاده از روش‌های صوری در سیستم‌های آموزشی و چندرسانه‌ای ۴۷
۵-۳- استفاده از روش‌های صوری در سیستمهای کنترل صنعتی و پزشکی ۴۸
۶-۳-کاربرد روش‌های صوری در انتقال دانش ۴۹
۷-۳-کاربرد روش‌های صوری در امنیت اطلاعات ۴۹
۸-۳-تمرین ها ۵۰

فصل چهارم: معرفی زبان Z و منطق گزارهای ۵۱

۱-۴-مقدمه ۵۱
۲-۴-جنبه های زبان Z ۵۱
۳-۴- انواع گزاره ها ۵۲
۴-۴-جداول درستی ۵۳
۵-۴-اولویت عمگرهای منطقی ۵۳
۶-۴-قواعد استنتاج (INFERENCE RULES) ۵۴
۷-۴-نحوه اثبات در Z ۵۵
۸-۴-مفهوم ASSUMPTION ۵۷
۹-۴- SCOPE چیست؟ ۵۸

۴-۱۰-Implication ۶۰
۴-۱۱-قوانین هم ارزی ۶۳
۴-۱۲-روش برهان خلف ۶۶
۴-۱۳-قوانین False ۶۶
۴-۱۳-۱-False - elim ۶۶
۴-۱۳-۲- False – introduction ۶۷
۴-۱۴-لم ها ۷۱
۴-۱۵-تاتولوژی ۷۲
۴-۱۶-Argument یا قانون با استفاده از implecation ۷۲
۴-۱۷-ساخت قانون یا equivalence ۷۳
فصل پنجم: منطق مستندات مرتبه اول ۷۵
۵-۱-مفهوم Predicate ۷۵
۵-۲-سورها (Quantification) ۷۶
۵-۲-۱-سور وجودی ۷۷
۵-۳- Constraints (محدودیت) ۷۷
۵-۴-متغیرهای آزاد و متغیرهای محدود(bound) ۷۹
۵-۵-جایگزینی ۸۰
۵-۶-تغییر نام دادن متغیرهای bound ۸۱
۵-۷- Variable capture ۸۱
۵-۸-قوانین سور عمومی و سور وجودی ۸۲
۵-۸-۱-سور عمومی ۸۳
۵-۸-۲-سور وجودی ۸۹

فصل ششم: تساوی و تعریف توصیف ۹۷

۶-۱-تساوی ((EQUALITY) ۹۷
۶-۲-قانون انعکاسی (AXIOM OF REFLECTION) ۹۸
۶-۳-قانون جابجایی (SYMMETRY) ۹۸
۶-۴-قانون تعدی (TRANSITIVITY) یا قانون تراگذاری ۹۸
۶-۵-قانون جایگزینی تساوی (SUBSTITUTION OF EQUALS) ۹۹
۶-۶-قانون تک نقطه ای (ONE – POINT) ۹۹
۶-۷-سور یکتایی(UNIQUENESS) ۱۰۱
۶-۸-M-EXPRESSION ۱۰۲
۶-۸-۱-اهمیت μ کجاست؟ ۱۰۳
۶-۸-۲-قانون μ-introduction ۱۰۴

بخش هفتم: مجموعه ها در Z ۱۰۷

۷-۱-مقدمه ۱۰۷
۷-۲-عضویت (MEMBERSHIP) ۱۰۸
۷-۳-قانون EXTENSIONALITY ۱۰۸
۷-۴-اصل SUBSET ۱۰۸
۷-۵-قانون مجموعه تهی (THE EMPTY SET) ۱۰۹
۷-۶-مجموعه اعداد طبیعی (THE NATURAL NUMBERS) ۱۰۹
۷-۷- SET COMPREHENSION ۱۰۹
۷-۸- TERM COMPREHENSION ۱۱۰
۷-۹- POWER SET ۱۱۱
۷-۱۰- حاصلضرب دکارتی (CARTESIAN PRODUCT) ۱۱۲

۱-۱۰-۷-قانون حاصلضرب دکارتی(pair membership) ۱۱۲
۱۱-۷-تساوی دو تاپل (EQUALITY) ۱۱۳
۱۲-۷- COMPONENT SELECTION ۱۱۳
۱۳-۷-اجتماع (UNION) ۱۱۵
۱۴-۷-اشتراک (INTERSECTION) ۱۱۶
۱۵-۷-تفاضل(DIFFERENCE) ۱۱۶
۱۶-۷-نوع‌ها (TYPES) ۱۱۷
۱-۱۶-۷-مزایای Type ۱۱۹
۲-۱۶-۷- Type مجموعه تهی چیست؟ ۱۱۹

فصل هشتم: تعاریف ۱۲۱

۱-۸-مقدمه ۱۲۱
۲-۸-نحوه تعریفBASIC TYPE ۱۲۱
۳-۸-ABRIRATION ها ۱۲۲
۴-۸-AXIOMATIC DEFINATION ها ۱۲۳
۵-۸-مفهوم سازگاری (CONSISTENCY) ۱۲۴
۶-۸-GENERIC DEFINATION ها (تعاریف عمومی) ۱۲۶
۷-۸-GENERIC ABBREVIATION ۱۲۷

فصل نهم: رابطه ها ۱۳۳

۱-۹- مقدمه ۱۳۳
۲-۹-دامنه و برد رابطه : (DOMAIN , RANGE) ۱۳۵
۳-۹-DOMAIN RESTRICTION و RANGE RESTRICTION ۱۳۶
۴-۹-DOMAIN SUBTRACTION و RANGE SUBTRACTION ۱۳۷

۹-۵-معکوس رابطه ها ۱۳۷

۹-۶-تصویر رابطه ای (RELATIONAL IMAGE) ۱۳۸

۹-۷-ترکیب رابطه ای (RELATIONAL COMPOSITION) ۱۴۰

۹-۸-خاصیت انعکاسی (REFLEXIVITY) ۱۴۱

۹-۹-خاصیت تقارنی(SYMMETRIC) ۱۴۱

۹-۱۰-خاصیت تعدی یا تراگذری ۱۴۲

۹-۱۱-خاصیت هم ارزی ۱۴۲

۹-۱۲-مفهوم CLOSURE (بستار) ۱۴۲

۹-۱۳-TRANSITIVE CLOSURES ۱۴۳

۹-۱۴-استراتژی اثبات ۱۴۶

فصل دهم: توابع ۱۴۷

۱۰-۱-مقدمه ۱۴۷

۱۰-۲-تابع جزئی ۱۴۷

۱۰-۳-تابع TOTAL (توابع کلی) ۱۴۹

۱۰-۴-قانون APPLICATION – INTERO: ۱۵۰

۱۰-۵-APPLICATION-DEMINATION ۱۵۰

۱۰-۶-نماد لاندا (LAMBDA NOTATION) ۱۵۱

۱۰-۷-نماد FUNCTION OVERRIDING ۱۵۵

۱۰-۸-مجموعه های متناهی (FINITE SET) ۱۵۸

۱۰-۹-اندازه (SIZE) یا کاردینالیتی یک مجموعه متناهی(HASH OR CARDINALITY): ۱۵۹

۱۰-۱۰-توابع متناهی (FINITE FUNCTIONS) ۱۶۰

۱۰-۱۱-توابع یک به یک متناهی (FINITE INJECTIONS) ۱۶۰

فصل یازدهم: دنباله ها ۱۶۱

۱-۱۱-مفهوم دنباله ۱۶۱

۲-۱۱-عملگر فیلتر (FILTER) ۱۶۲

۳-۱۱-اپراتورهای HEAD و TAIL: ۱۶۲

۴-۱۱-LENGTH یا اندازه دنباله ۱۶۲

۵-۱۱-نماد REVERSE: ۱۶۳

۶-۱۱-مدل کردن دنباله ها ۱۶۳

۷-۱۱-CONCATENATION ۱۶۴

۸-۱۱-توصیف فرمال برای HEAD و TAIL ۱۶۵

۹-۱۱-تعریف RECURSIVE ۱۶۶

۱۰-۱۱-اصل RECURSION ۱۶۶

۱۱-۱۱-عملگر FILTER ۱۶۷

۱۲-۱۱-REVERSE ۱۶۸

۱۳-۱۱-USEFUL LAWS ۱۶۸

۱۴-۱۱-EQUATION UE REASONING ۱۶۸

۱۵-۱۱-استقرار ساختاری (STRUCTURE INDUCTION) ۱۶۹

۱۶-۱۱-BAG ها ۱۷۱

فصل دوازدهم: انواع آزاد ۱۷۳

۱-۱۲-مقدمه ۱۷۳

۲-۱۲-SYNTAX انواع آزاد ۱۷۳

۳-۱۲-کاربردهای NAT: ۱۷۵

۴-۱۲-تعریف درخت باینری کامل ۱۷۵

۵-۱۲-اصل استقرار برای NAT: ۱۷۷
۶-۱۲-اصل CLOSURE برای TREE ۱۷۷
۷-۱۲-اثبات اصل استقرا برایNAT ۱۷۸
۸-۱۲-اثبات اصل استقراء برای BINARY TREE ۱۷۸
۱۰-۱۲-تعریف فاکتوریل در Z ۱۷۸

فصل سیزدهم: شِما ها ۱۸۱

۱-۱۳-مقدمه ۱۸۱
۲-۱۳-نماد شِما ۱۸۲
۳-۱۳-هم ارزی شماها ۱۸۵
۴-۱۳-کاربرد شماها ۱۸۵
۵-۱۳-شما به عنوان مجموعه ها: ۱۸۹
۶-۱۳- COMPONENT SELECTION ۱۹۰
۷-۱۳-نرمال سازی (NORMALIZATION) ۱۹۳
۸-۱۳-تغییر نام درشماها (RENAMING) ۱۹۴
۹-۱۳-DECORATION ۱۹۵
۱۰-۱۳-تساوی دو BINDING ۱۹۶

فصل چهاردهم: عملیات اسکیما ۱۹۷

۱-۱۴-مقدمه ۱۹۷
۲-۱۴-راهکار توصیف سیستمها در Z: ۱۹۷
۳-۱۴-اپراتورهای اسکیما ۱۹۸
۴-۱۴-عملیات ANDکردن شماها (SCHEMA CONJUNCTION) ۱۹۸
۵-۱۴- SCHEMA IN CLUSION ۲۰۱

۶-۱۴-تغییر حالت ۲۰۱
۷-۱۴-نماد دلتا ۲۰۴
۸-۱۴-نمادΞ ۲۰۵
۹-۱۴-حالت شروع(INITIALIZATION STATE) ۲۰۶
۱۰-۱۴-عملیات جزئی و کلی ۲۰۸
۱۱-۱۴-تمرین ها ۲۰۹
۱۲-۱۴-ساخت اپراتورها ۲۰۹
۱۳-۱۴-عملگر NEGATION یا نفی شماها ۲۱۱
۱۴-۱۴-سور روی شماها (SCHEMA QUANTIFICATION) ۲۱۲
۱۵-۱۴-مفهوم HIDING برای سور وجودی ۲۱۴
۱۶-۱۴-SCHEMA COMPOSITION ۲۱۵

فصل پانزدهم: پیش شرط ها (PRECONDITIONS) ۲۲۳

۱-۱۵-مقدمه ۲۲۳
۲-۱۵-تعریف پیش شرط ۲۲۴
۳-۱۵-تئوری INITIALIZATION ۲۲۷
۴-۱۵-مقایسه پیش شرط های ضمنی و صریح ۲۳۱
۵-۱۵-قانون DISJUNCTION ۲۳۵
۶-۱۵-قانون CONJUNCTION ۲۳۶

فصل شانزدهم: فایل سیستم ۲۳۷

۱-۱۶-مقدمه ۲۳۷
۲-۱۶-توصیف عملیات فایل‌ها ۲۳۸
۳-۱۶-TOTAL کردن توصیف ۲۴۸

۴-۱۶-مرحله تحلیل ۲۵۰
منابع: ۲۵۳

مقدمه:

چرخه توسعه نرم‌افزار از مرحله تحلیل نیازمندی‌ها آغاز می‌شود و نهایتا به مرحله تست و نگهداری ختم می‌شود. طی این چرخه مسائل مختلفی وجود دارد که ممکن است باعث شود نرم‌افزار فاقد قابلیت اطمینان باشد. یکی از این مسائل آن است که نیازمندی‌های مشتری به درستی توسط توسعه دهندگان درک نشود. مشکل دیگر به ماهیت روش‌های تست مربوط می‌شود. روش‌های تست فقط می‌توانند به شما نشان دهند که چه خطاهایی در سیستم وجود دارد، ولی نمی‌توانند تضمین کنند که خطایی در سیستم وجود نداشته باشد. به همین علت نرم‌افزارهای زیادی را می‌توان نام برد که توسعه‌دهندگان آن از بهترین روش‌های تست استفاده نموده‌اند، اما بعد از گذشت مدتی اشکالات زیادی در آنها مشاهده می‌شود. نمونه بارز این نرم‌افزارها سیستم عامل ویندوز است.

یکی از راه‌حل‌های غلبه یافتن بر اینگونه مشکلات، استفاده از روش‌های صوری است. با توجه به اینکه این روش‌ها مبنای ریاضی دارند، فاقد مشکلات موجود در سایر روش‌ها هستنند.

اهمیت استفاده از روش‌های صوری در سیستم‌هایی که قابلیت اطمینان مسئله‌ای کلیدی و بحرانی باشد، بیشتر آشکار می‌شود. نمونه‌ای از اینگونه سیستم‌ها، نرم‌افزارهای کنترل راه آهن و همچنین نرم‌افزار مانیتورینگ مسیرهای پرواز است که ایمنی و سلامت انسان‌ها به عملکر صحیح آنها بستگی دارد.

علیرغم مزایای روش‌های صوری و نیاز جامعه علمی کشور به منابع فارسی، تاکنون در کشور ما کتابی در این زمینه منتشر نشده است. لذا این کتاب به منظور برآورده کردن نیاز متخصصن و صنعت‌گران تالیف گردید. در این کتاب بعد از بیان اهمیت و مزیت به کارگیری روش‌های صوری در چرخه حیات توسعه نرم‌افزار، حیطه‌های به کارگیری روش‌های صوری شامل توصیف صوری، درستی‌یابی صوری، تولید برنامه صوری، روش‌ها، ابزار و زبان‌های شناخته ارائه می‌گردد. همچنین یک بخش از کتاب به بررسی تجربه‌های صنعتی روش‌های صوری اختصاص داد. نهایتا نیز زبان توصیف Z که یکی از متداول‌ترین و محبوب‌ترین زبان‌های توصیف رسمی می‌باشد، مورد بررسی قرار خواهد گرفت.

از این کتاب می‌توان به عنوان مرجع دروس روش‌های صوری در مهندسی نرم‌افزار، توصیف و وارسی نرم‌افزار، تولید برنامه صوری و مباحث ویژه در مهندسی نرم‌افزار استفاده نمود.

امید است این اثر مورد توجه اساتید، همکاران و دانشجویان گرامی قرار گرفته و به دانشجویان رشته کامپیوتر در مقاطع مختلف کمک نماید. از اساتید و دانشجویان گرامی تقاضا دارم دیدگاه‌های خود را از طریق ایمیلm.a.torkamani@gmail.com با مولف در میان بگذارند تا انشالله در چاپ بعدی کتاب اشکالات یا کاستی‌های احتمالی آن مورد تجدید نظر قرار گیرد. در پایان وظیفه خود می‌دانم از زحمات آقای مهندس بیات به خاطر طراحی جلد کتاب و همچنین از مدیریت سامانه اطلاع رسانی چاپ و نشر ایران، جناب آقای حسین قنبری، تشکر و قدردانی نمایم.

محمد علی ترکمانی

پاییز ۱۳۹۵

فصل اول

مقدمه‌ای بر مهندسی نرم‌افزار و روش‌های صوری

هدف از این فصل، بیان اهمیت و مزیت به کارگیری روش‌های صوری در چرخه حیات توسعه نرم‌افزار است.

۱-۱-روش‌های صوری [۱]

تعاریف مختلفی برای روش‌های صوری وجود دارد. یکی از جامع‌ترین تعاریف به صورت زیر است:
روش‌های صوری روش‌هایی مبتنی بر ریاضیات هستند که در هر مرحله از چرخه حیات نرم‌افزار قابل به کارگیری هستند و هدف از به کارگیری آنها افزایش قابلیت اطمینان [۲] است.
این تعریف دارای سه قسمت است. اول این که روش‌های صوری مبتنی بر ریاضیات است. یعنی هر روشی که مبنی بر ریاضیات نباشد، روش صوری نیست. دوم اینکه روش‌های صوری در هر مرحله از چرخه حیات نرم‌افزار (اعم از توصیف نیازمندی‌ها، تحلیل، طراحی، پیاده‌سازی و تست) قابل به کارگیری است. بنابراین این روش‌ها صرفا محدود به یکی از مراحل چرخه توسعه نیستند، بلکه در تمام مراحل قابل استفاده هستند.
همانطور که در تعریف فوق ذکر شده است، هدف قابلیت اطمینان است. البته در این تعریف آن جنبه‌ای از قابلیت اطمینان که برای ما اهمیت دارد، درستی [۳] است.
همانطور که می‌دانید قابلیت اطمینان چندین جنبه از نرم‌افزار را پوشش می‌دهد که عبارتند از قابلیت دسترسی [۴]، تحمل پذیری در برابر خطا [۵]، درستی و غیره. به عبارت دیگر جنبه‌های مختلفی از نرم‌افزار به

---

1 Formal Methods
2 Reliability
3 Correctness
4 Availability
5 Fault Tolerance

قابلیت اطمینان ارتباط دارد که اگر نرم‌افزار در این جنبه‌ها رفتار خوبی داشته باشد، می‌توانیم به نرم‌افزار تکیه کنیم.

اما مفهوم ‘‘درستی’’ که برای ما مهم است این است که نرم‌افزار بر اساس نیازمندی‌های اولیه که مشتری درخواست کرده، رفتار می‌کند. درستی یعنی تطابق رفتار نرم‌افزار با نیازمندی‌های اولیه.

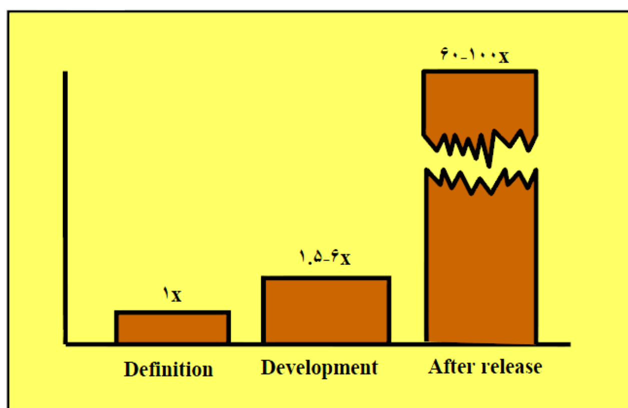
به طور کلی وقتی که بحث روش‌های صوری در مهندسی نرم‌افزار مطرح می‌شود، مفهومش این است که ما بتوانیم کل فرایند توسعه نرم‌افزار را روی ریاضیات سوار کنیم.

ایده روش‌های صوری در مهندس نرم‌افزار از دو نمودار ۱-۱ و ۱-۲ ریشه گرفت. این نمودارهای در کتاب مهندسی نرم‌افزار آقای پرسمن نیز آورده شده است و هدف آن این است که نشان دهد که نرم‌افزار از سایر محصولات متمایز است.

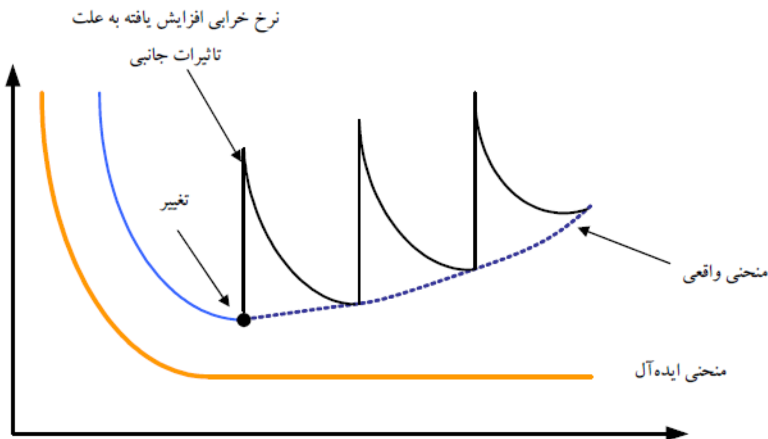
شکل ۱-۱ بیان می‌کند که هر چقدر در فازهای توسعه نرم‌افزار جلوتر می‌رویم، هزینه رفع خطای ما بیشتر می‌شود. این نمودار کاملا طبیعی است. زیرا اگر شما در مراحل اول توسعه (نظیر تحلیل) خطایی را کشف کنید، کار خود را عوض کرده و مشکل را به راحتی برطرف می‌کنید تا خواسته مشتری بهتر برآورده شود. این کار هزینه چندانی هم ندارد. زیرا هنوز طراحی و پیاده‌سازی انجام نشده است. اما هر چقدر جلوتر بروید هزینه‌ها زیادتر می‌شود. به عنوان مثال اگر بخواهید در مرحله پیاده‌سازی خطایی را رفع کنید، باید کل فاز را به عقب برگردید و تحلیل، طراحی و پیاده‌سازی را تصحیح کنید.

بعد از تحویل نرم‌افزار به مشتری، مشکلات بیشتر و وضعیت وخیم‌تر هم می‌شود. در این مرحله شما فقط با مسائل فنی سرو کار ندارید، بلکه ممکن است جریمه شوید، مشکلات حقوقی پیدا کنید و یا مشتری از شما شکایت کند. هزینه رفع خطا در این مرحله بسیار زیاد است.

نهایتا اگر خطاها خیلی زیاد باشد، رفع خطاها تقریبا غیر ممکن است و نرم‌افزار خواهد مرد.



شکل ۱-۱: هزینه تغییر در زمان‌های متفاوت



شکل ۱-۲: منحنی نرخ خرابی نرم‌افزار نسبت به زمان(منشاء بروز خطاها)

اگر نمودار دیگر که در شکل ۲-۱ نمایش داده شده است را کنار نمودار قبلی قرار دهیم، نکات جالبی مشخص می‌شود. در این شکل دقیقا یک نمودار برعکس داشتیم. یعنی بیشتر خطاهایی که در نرم‌افزار پیدا می‌شود، منشاء بروز آن در مراحل اولیه توسعه است. خطاهای جدی‌تر و خطاهایی که هزینه بیشتری برای رفع آنها لازم است، همین خطاها هستند. به عنوان مثال خطایی که برنامه‌نویس ایجاد کرده باشد، خطاهایی هستند که خیلی زود کشف می‌شوند، خطاهایی در حد Run time error، syntax error هستند که راحت کشف می‌شوند و رفع آنها نیز هزینه چندانی ندارد.

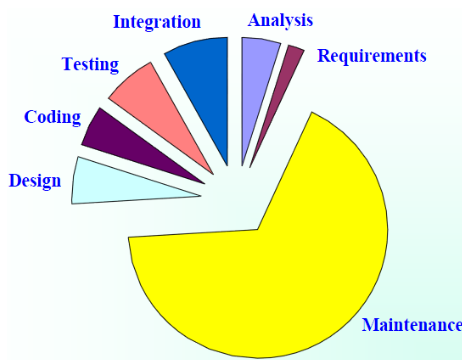
اما خطاهایی که ناشی از عدم درک درست ما از نیازهای مشتری هستند بسیار مشکل ساز هستند. به عنوان مثال توسعه دهنده حرف مشتری را درست متوجه نمی‌شود و یا مشتری یک مورد خاص را همان ابتدای کار از توسعه دهنده نمی‌خواهد و بعداً می‌گوید (اینقدر به نظر وی بدیهی است که نمی‌گوید)، اما برای توسعه دهندگان این مسئله بسیار مشکل است.

این موارد تحلیل را دچار مشکل می‌کند و هزینه بیشتری دارد و در ادامه به تبع آن طراحی و پیاده‌سازی هم نیاز به تغییر دارد. گاهی اوقات حتی ممکن است چیزی که برای مشتری خیلی مهم بوده است[1] توسط توسعه دهنده دیده نشده باشد و توسعه دهنده مجبور می‌شود معماری نرم‌افزار را عوض کند.

1 Design Force

حال اگر این دو نمودار را با هم مقایسه کنیم، نمودار اول می‌گوید که هر چقدر جلوتر برویم، هزینه رفع خطاها زیادتر می‌شود، از طرف دیگر منشاء خطاها به خصوص خطاهایی که هزینه رفع آنها زیاد است، در مراحل اولیه تولید نرم‌افزار است.

از سوی دیگر در رویکرد سنتی توسعه نرم‌افزار تاکید ما روی این دو نکته است: اول اینکه برنامه‌نویس‌های خوبی داشته باشیم که کدهای خوبی می‌نویسند و نکته دوم تست است. یعنی آنقدر نرم‌افزار را تست می‌کنیم که مطمئن شویم نرم‌افزار ما یک نرم‌افزار قابل اتکا است. حال اگر این رویکرد را که در حال حاضر نیز رویکرد متداولی است با دو نمودار قبل مقایسه کنیم، متوجه می شویم که رویکرد بدی است. زیرا تکیه ما روی مراحل انتهای توسعه (یعنی خلاقیت برنامه نویس و تست) است، در حالی که منشاء بیشتر خطاها مربوط به مراحل اولیه توسعه است. سئوال مهم این است که ما برای مراحل اولیه (تحلیل، طراحی و معماری) چه فکری کرده‌ایم؟ به همین علت است که نرم‌افزار یک محصول bug دار است و هرکس که آنرا خریداری می‌کند، می‌داند که باید آنقدر آن را تحمل کند تا عملیاتی شود. علت این موضوع نیز همین رویکرد غلط است. شکل ۱-۳ نیز این موضوع را ثابت می‌کند. همانطور که در این شکل مشاهده می‌کنید هزینه نگهداری نرم‌افزار بسیار زیادتر از فازهای دیگر است.



شکل ۱-۳ : هزینه فازهای مختلف توسعه

انگیزه مطرح شدن روش‌های صوری همین مطلب بود. یعنی رویکرد سنتی توسعه نرم‌افزار را کنار بگذاریم و به گونه دیگری به قضیه نگاه کنیم. در بحث روش‌های صوری اولین موردی که مطرح شد بحث توصیف‌های صوری[1] بود.

با توجه به این واقعیاتی که در روش‌های سنتی فرایند تولید نرم‌افزار وجود دارد، ما چند تا تغییر در این فرایند انجام می‌دهیم. ایده اول استفاده از توصیف‌های صوری است. توصیف یعنی تشریح و بیان

1 Formal Specification

نیازمندی‌های مشتری در یک سیستم نرم‌افزاری. توصیف صوری یعنی توصیفی که مبتنی بر روش‌های ریاضی است. به عبارت دیگر باید بتوانیم توصیف‌های غیر صوری را به وسیله روش‌های ریاضی بیان کنیم.
پس با توجه به اینکه بیشترین مشکل ما در مرحله اولیه تولید نرم‌افزار است، یعنی جایی که می‌خواهیم نیازمندی‌ها را بشناسیم و تحلیل کنیم، تا جای ممکن این قسمت را مبتنی بر ریاضیات انجام دهیم. بنابراین باید روی این قسمت تمرکز کنیم و سعی کنیم این قسمت را تا جای ممکن درست انجام دهیم و از ریاضیات برای توصیف استفاده کنیم. استفاده از ریاضیات در توصیف مزایای زیر را دارد:
۱- دقت و عدم ابهام
۲- کامل بودن
۳- عدم تناقض
۴- امکان استدلال روی خواص نرم‌افزار (خواص توصیف)
در ادامه این مراحل را توضیح می‌دهیم.
وقتی از ریاضیات استفاده می‌کنیم، می‌توانیم موضوع مورد بحث را به صورت دقیق‌تر نشان دهیم و ابهام آن کمتر می‌شود. به عنوان مثال به شما می‌گویند برنامه‌ای بنویسید که با دریافت یک عدد طبیعی، جذر آن را برگرداند. این یک توصیف است، اما یک توصیف غیر صوری[1]است. این توصیف چند تا مشکل دارد. اول اینکه دقیق نیست و در آن ابهام وجود دارد ما نمی‌دانیم که به عنوان مثال اگر به عنوان ورودی عدد ۴ را وارد کنیم، به عنوان خروجی باید ۲ را برگردانیم و یا ۲- را برگردانیم و هر دو عدد را.
حال فرض کنید یک سیستم بزرگ دارید و یک سند ۲۰۰ صفحه‌ای در مورد نیازمندی‌های سیستم به شما داده‌اند که داخل آن از اینگونه جملات زیاد است. درست است که با گفتگوی دو طرفه می‌توان همه چیز را درست کرد اما برای یک سند ۲۰۰ صفحه‌ای چند جلسه باید بگذاریم؟
حال به فرض که شما ۱۰ تا جلسه هم گذاشتید، از کجا تضمین می‌شود که همه چیز درست شده است و چیزی مبهم نمانده است؟ البته در دنیای واقعی هیچ‌وقت ۱۰ تا جلسه نمی‌گذاریم. زیرا شما می‌خواهید سریع برنامه را توسعه دهید و از رقبا عقب نمانید و سریع‌تر هزینه پروژه را دریافت کنید. در کنار این مسائل چنیدن مسئله غیر فنی نیز وجود دارد.
پس توصیف‌های غیر صوری دقیق نیستند و درون آنها ابهام وجود دارد. اگر بخواهید یک توصیف صوری برای مثال قبل بنویسید به صورت زیر خواهد شد:

$$\forall A \in N, \exists n \in N: n \times n = A$$

1 informal

در این توصیف کاملا واضح است که عدد n که ما به دنبال آن هستیم، یک عدد طبیعی است. می‌توانیم بنویسیم یک عدد مثبت است. وقتی چنین عبارتی را ببینیم به راحتی با مشتری بحث می‌کنیم و می‌گوییم آیا منظور شما عدد مثبت است؟ بنابراین ریاضیات ما را به سمت دقت و عدم ابهام سوق می‌دهد. ضمنا در جملات شرط گونه معمولا ابهام وجود دارد و فردی هم که می‌خواند خیلی برای رفع ابهام کنجکاو نمی‌شود. در حالی که جملات ریاضی دقیق هستند. نکته دوم کامل بودن است.

در همین عبارت توصیف یک نقص دارد. به عنوان مثال مشخص نیست که اگر عدد ورودی منفی بود باید چه کاری انجام دهیم. می‌خواهیم یک پیغام مناسب چاپ کنیم، می‌خواهیم خود ورودی را برگردانیم؟ رفتار سیستم در مورد شرایط استثناء چگونه است؟ اصطلاحا می‌گوییم سیستم باید به صورت Partial عمل کند. یعنی رفتار total برای آن وجود ندارد. در چنین شرایطی سیستم باید چه کاری انجام دهد؟

وقتی که به صورت ریاضی کار کنیم، اینطورنقص‌ها بیشتر مشخص می شوند. یعنی می‌توانیم از فردی که نیازمندی‌ها را نوشته سئوال کنیم که اگر ورودی عضو N نبود، باید چه اتفاقی بیفتد؟

به عبارت دیگر نوشتن توصیف سیستم به صورت ریاضی ما را به سمت کامل بودن سوق می‌دهد.

موضوع سوم عدم تناقض است. در RFP های فارسی پروژه مشاهده می‌شود که به عنوان مثال در صفحه ۴ یک موضوعی گفته شده است ولی در صفحه ۱۵۰ آن مورد نقض شده است و یا ناسازگار هستند. علت این مسئله این است که هر قسمت از اسناد حجیم را یک نفر تهیه می‌کند که اصلا ممکن است دید این افراد نسبت به موضوع با هم تفاوت داشته باشد.

اما وقتی از ریاضیات برای توصیف و تشریح نیازمندی‌های استفاده می‌کنید، می‌توانید از ابزاری استفاده کنید که این تناقضات را پیدا کنید. ابزارهایی هستند که می‌توانند جاهای مختلف توصیف را با هم مقایسه کرده و تناقضات را پیدا نمایند. نکته آخر، بحث امکان استدلال روی روش‌های صوری است.

هر وقت از ریاضیات برای توصیف یک سیستم استفاده کنیم، می‌توانیم خواصی از آن توصیف را هم اثبات کنیم. یعنی همان کاری است که در تست انجام دادیم، آنقدر تست می‌کردیم که دیگر اشکالی در سیستم نبینیم. اما در اینجا سیستم را توصیف می‌کنیم و بعد چون این توصیف مبتنی بر ریاضیات است، به کمک قوانین ریاضی نشان می‌دهیم که این توصیف این خواص را دارد. این مورد با تست کردن سیستم تفاوت دارد.

دایکسترا یک جمله معروف در مورد تست دارد:

«تست فقط می‌تواند به شما نشان دهد که چه خطاهایی در سیستم وجود دارد، ولی نمی تواند تضمین کند که خطایی در سیستم نیست. این مهمترین نقطه ضعف تست است».

حتی اگر از بهترین روش‌ها و ابزار تست هم استفاده کنید، فقط می‌توانید یک تعدادی از خطاهای موجود در سیستم را پیدا کنید، اما باز هم نمی‌توان تضمین کرد که خطایی در سیستم وجود ندارد. ممکن است اگر یک ماه دیگر هم از نرم‌افزار استفاده کنید، خطایی در سیستم به وجود بیاید.

این مسئله موجب خواهد شد که سیستم‌های نرم‌افزاری که برای تضمین ایمنی و کنترل سیستم‌های بحرانی به کار می‌روند، عملکرد صحیحی نداشته باشند و نیازمندی‌های مشتری را برآورده نکنند.

اما در روش‌های صوری به کمک روش‌های ریاضی ثابت می‌کنیم که در توصیف ما خواص مورد نظر وجود دارد. چون اثبات کرده‌ایم، بنابراین مثال نقض برای آن وجود ندارد و کاملا می‌توان به نرم‌افزار تکیه کرد.

۲-۲- مولفه‌های یک روش رسمی

بعضی از محققین روش‌های رسمی را به صورت ذیل تعریف می‌کنند:

روش‌های رسمی= توصیف رسمی[۱] + تایید رسمی[۲]

بر اساس تعریف فوق، روش‌های رسمی با مولفه‌های زیر شناخته می‌شود:

۱-نماد رسمی[۳] یا زبان رسمی[۴] که برای نوشتن توصیفات به کار می‌رود.

۲-حساب منطق برای تایید[۵] یا اثبات[۶]

۳- روش یا سیستم توسعه نرم‌افزار

۲-۳-تمرین‌ها

۱-مشکلات رهیافت غیر رسمی توسعه نرم‌افزار چیست؟

۲-روش‌های صوری را تعریف کنید.

۳- مفاهیم زیر را شرح دهید.

الف- قابلیت اطمینان ب ایمنی(Safety) ج- درستی

۴- مزایای استفاده از ریاضیات در توصیف چیست؟

۵- کدامیک از روش‌های زیر روش صوری هستند؟

الف- استفاده از زبان Z برای توصیف یک سیستم

ب- استفاده از ریاضیات برای نمایش نیازمندی‌های یک سیستم نرم‌افزاری

ج- استفاده از یک فرمول ریاضی برای اثبات درستی یک مسئله در مهندسی نرم‌افزار

۶- فرض کنید شما مدیر بخش فناوری اطلاعات یک شرکت هستید و قصد دارید یک برنامه نرم افزاری مطمئن برای استفاده در خط تولید شرکت خود خریداری نمایید. دو فروشنده نرم افزار شما را عرضه نموده‌اند. فروشنده اول از روشهای صوری برای توصیف و نهایتا تولید کد استفاده نموده است. فروشنده

1 Formal Specification
2 Formal Verification
3 Formal Notation
4 Formal Language
5 Verification
6 Proof

دوم از روش غیر رسمی استفاده نموده اما نرم افزار را با استفاده از جدیدترین روش‌های تست آزماش و به مدت ۳ ماه در یک شرکت دیگر عملیاتی کرده است. کدام برنامه را خریداری می‌کنید؟

۶- آیا روش‌های صوری می‌توانند جایگزین مناسبی برای روش‌های تست باشند؟ چرا؟

۷- نقش توصیف در فرایند توسعه نرم‌افزار را توضیح دهید.

فصل دوم

حیطه‌های به کارگیری روش‌های صوری

هدف از این فصل، معرفی حیطه‌های اصلی شامل توصیف صوری، درستی‌یابی صوری، تولید برنامه صوری، روش‌ها، ابزار و زبان‌های شناخته شده است.

۱-۲- مقدمه

به طور کلی سه مبحث مهم به عنوان حیطه به کارگیری روش‌های صوری وجود دارد. این سه حیطه عبارتند از:

۱- توصیف صوری

۲- درست‌یابی صوری[1]

۳- تولید برنامه صوری[2]

در فصل قبل در مورد توصیف صوری و اهمیت آن بحث کردیم و گفتیم که در حال حاضر مهم‌ترین کاربرد روش‌های صوری، بحث توصیف صوری است.

در این فصل در خصوص درست‌یابی صوری و تولید برنامه صوری بحث خواهیم کرد. پیش‌نیاز این موارد بحث توصیف صوری است.

فرض کنید مشتری به صورت غیر صوری یک سیستم را برای شما شرح داده است و شما هم به صورت صوری توصیف سیستم را نوشته‌اید. می‌توان در این مرحله روش صوری را کنار گذاشت و با روش‌های صوری نرم‌افزار بر اساس توصیف نوشته شده تولید نمود. تجربه‌های موجود در صنعت نشان می‌دهد که تا همین جا هم چنین برنامه‌هایی نسبت به برنامه‌هایی که با فرایند تولید سنتی توسعه یافته‌اند، دارای هزینه توسعه و نگهداری کمتر و قابلیت اطمینان بیشتر هستند.

---

1 Formal Verification
2 Formal Program Development

اما هدف آرمانی این است که درست‌یابی صوری و تولید برنامه صوری هم انجام شود. درست‌یابی صوری یعنی این که باید توصیفی که داریم را Verify کنیم. یعنی ببینیم این توصیف با نیازی که مشتری داشته مطابقت دارد یا نه. این کار نیز باید به شکل صوری و با استفاده از ابزار و روش‌های ریاضی انجام شود.

بنابراین مشتری نیازمندی‌هایی دارد که تعدادی از آنها در ذهن وی است و تعدادی نیز در مستندات (نمودارها، شکل ها، توضیحات و نقشه ها و ...) است.

ابتدا یک توصیف صوری از نیازهای مشتری می‌نویسیم، سپس باید این توصیف صوری را در تطابق با نیازمندی‌ها Verify کنیم. یعنی ببینیم این توصیف واقعا همان چیزی است که مشتری می‌خواهد یا نه. ایده درست‌یابی صوری بسیار ساده است. شما توصیف را گرفته و یک سری خواص را در آن اثبات می‌کنید. به عنوان مثال فرض کنید یک نرم‌افزار برای سیستم عامل می‌خواهید که ناحیه بحرانی را کنترل کند. بنابراین نیازمندی شما یک برنامه برای مدیریت ناحیه بحرانی است.

برای مدیریت ناحیه بحرانی چندین هدف وجود دارد: منصف بودن، جلوگیری از بن بست و غیره .... . برای اینکه نشان دهید الگوریتم ارائه شده توسط شما (که به صورت یک توصیف است) هیچگاه سیستم را دچار بن بست نمی‌کند، باید اثبات کنیم. به عبارت دیگر شما یک توصیف دارید که دارای یک خاصیت[۱] است.

مفهوم ویژگی ذکر شده این است که سیستمی که مطابق با این خاصیت است، هیچگاه دچار بن بست نمی‌شود. ما این خاصیت را برای این توصیف اثبات می‌کنیم. در واقع این توصیف را بر اساس این نیاز Verify می‌کنیم و بررسی می‌کنیم که آن را برآورده می‌کند یا خیر؟ به عنوان مثال دیگر خاصیت منصف بودن برای ما مهم است، یعنی هر پردازه‌ای که بخواهد وارد ناحیه بحرانی شود، می‌تواند بالاخره وارد شود.

۲-۲-انواع ویژگی‌های یک سیستم

به طور کلی ویژگی‌های که باید در درست‌یابی باید ثابت شوند، دو دسته هستند:

۱- Safety Properties

۲- Liveness Properties

خواص Safety خواصی هستند که تغییر ناپذیر[۲]هستند و همیشه برقرار هستند. به عنوان مثال در مدیریت ناحیه بحرانی می‌خواهیم بگوییم نرم‌افزار ما deadlock free است. در اینجا dead lock free بودن یک خاصیت Safety است که همیشه باید برقرار باشد.

---

1 Property
2 Invariant

به عنوان مثال دیگر در سیستم‌های کنترل همروندی که چندین تراکنش در آنها وجود دارد، امکان ندارد که دو تراکنش قفل‌های متضاد بگیرند. یعنی روی یک data item دو تراکنش قفل بگذراند (یک قفل نوشتن و دیگری قفل خواندن).

این یک خاصیت Safety است و همیشه برقرار است. یعنی هیچگاه امکان ندارد چنین اتفاقی بیفتد.

خاصیت Liveness خاصیتی است که بالاخره یک زمان برقرار خواهد شد و سیستم به چنین نقطه‌ای خواهد رسید. بهترین مثالی که می‌توان از این خاصیت عنوان نمود، حلقه‌ها است. وقتی که یک حلقه می‌نویسید، باید مطمئن باشید که حلقه بالاخره تمام می‌شود. فقط در این صورت است که می‌توان اطمینان حاصل کرد که حلقه درست کار می‌کند. همچنین در مثال ناحیه بحرانی، بالاخره یک فرایند وارد ناحیه بحرانی خواهد شد این یک خاصیت Liveness است.

۳-۲-درست‌یابی

روش‌های مختلفی برای درست‌یابی وجود دارند که مهم‌ترین آنها عبارتند از:

۱- چک کردن مدل:[1] ایده کلی این روش این است که ما سیستم را به صورت مجموعه‌ای از حالت‌ها و ارتباط بین حالت‌ها در نظر می‌گیریم که یک وضعیت start دارد و تعدادی حالت دیگر که ممکن است به آنها منتقل شویم (شبیه به Finite Automata). این مدل ما خواهد بود. سپس برای اینکه نشان دهید سیستم خاصیت safety دارد، کافی است نشان دهید که در هر کدام از این حالت‌ها این خواص برقرار است. مهم‌ترین چالشی هم که این مدل دارد، انفجار حالت است، یعنی ممکن است آنقدر تعداد حالت‌ها زیاد شود و سیستم آنقدر بزرگ شود که در عمل نتوانیم آن را پیاده‌سازی کنیم. اما خوبی این روش آن است که اولا روشی است که برای هر توصیف قابل استفاده است و دوم اینکه ابزار هم برای آن وجود دارد.

۲- اثبات قضیه:[2] در این روش برای اینکه ثابت کنیم یک خاصیت برقرار است، از قواعد استفاده می‌کنیم. در واقع مجموعه‌ای از قواعد را دارید که سیستم استنتاج را تشکیل می‌دهد. ما از این قواعد استفاده می‌کنیم تا نشان دهیم سیستم خاصیت Safety یا Liveness را دارد. در این روش مشکل انفجار حالت وجود ندارد. اما در توسعه صوری برنامه‌ها باید توصیف را به وسیله روش‌های صوری به برنامه تبدیل کنیم (این کار را با استفاده از ابزارهایی که فرض می‌کنیم آن ابزارها نیز درست هستند، انجام می‌دهیم).

1 Model checking
2 Theorem Proofing

بنابراین یک چارچوب داریم که به این ترتیب است که ابتدا یک توصیف صوری می‌نویسیم و سپس آن را درست‌یابی صوری می‌کنیم و در نهایت آن را به وسیله روش‌های صوری به برنامه تبدیل می‌کنیم. برنامه‌ای که به این ترتیب تولید شده باشد، یک برنامه درست خواهد بود که تمام خواسته‌های مشتری را برآورده می‌کند. در اینگونه روش‌ها دیگر اثری از برنامه‌نویسی نخواهید دید، بلکه بیشتر زمان توسعه در مراحل اولیه است.

بنابراین ابتدا توصیف را می‌نویسیم و سپس آن را درست‌یابی می‌کنیم و یک توصیف درست به دست می‌آوریم. سپس توصیف درست را به وسیله ابزارهای درست (که قبلا امتحان خودشان را پس داده‌اند) به یک برنامه تبدیل می‌کنیم. چون این برنامه از روش‌ها و ابزارهای درست به دست آمده، درست است و نیازی به تست هم ندارد.

۴-۲-تولید برنامه صوری

در تولید برنامه صوری هدف این است که GAP بین توصیف صوری و تولید برنامه نهای از بین برود. به عبارت دیگر در این روش از توصیف صوری به برنامه نهایی خواهیم رسید. در توصیف‌های صوری نباید وارد جزییات پیاده‌سازی، چگونگی و نحوه به دست آمدن خروجی از ورودی شوید. این موارد مربوط به توصیف نیستند بلکه موضوع پیاده‌سازی هستند. در توصیف فقط باید نیاز را مطرح کنید. به عنوان مثال برنامه‌ای می‌خواهیم که یک عدد را بگیرد و جذر آن را محاسبه کند. این یک توصیف است. اما این که جذر چگونه محاسبه می‌شود، الگوریتم آن چیست، مربوط به توصیف نیست.

بنابراین هنگام نوشتن توصیف‌های صوری باید دقت کنیم که چگونگی را ننویسیم. زیرا Syntax زبان‌های صوری این اجازه را به ما می‌دهند که چگونگی را بنویسیم، اما این موضوع با فلسفه توصیف مغایر است و نباید انجام شود. در توصیف فقط باید بگوییم ورودی و خروجی چه رابطه‌ای با یکدیگر دارند. در توصیف نباید بگوییم که خروجی چگونه از ورودی به دست می‌آید(الگوریتم را نمی خواهیم).

نکته: این رویکرد در ابتدا منتقدین زیادی داشت. مهمترین آنها مقاله‌ای بود که توسط C.B.Jones ارائه گردید. در این مقاله نشان داده شده است که از توصیف های صوری (که در آنها جزییات و چگونگی و الگوریتم در نظر گرفته نشده) می‌توان چند برنامه را تولید کرد. به عبارت دیگر توصیفی که به این روش نوشته می شود یک توصیف غیر قطعی[1]است، در حالی که برنامه‌های امروزی قطعی هستند. یعنی به ازای یک ورودی به یک خروجی مشخص می‌روند(از یک حالت به یک حالت دیگر می‌روند). اما توصیف‌ها

1 None-Deterministic

غیرقطعی هستند. این موضوع اصلی بود که توسط C.B.Jones مطرح شد. البته به تدریج این مشکل در حال برطرف شدن است.

به طور کلی سه روش برای تولید برنامه صوری وجود دارد:

۱- تقلید[1]

۲- پالایش[2]

۳- روش استخراج برنامه یا روش ساختی[3]

۱-۲-۲- روش تقلید

روش تقلید روشی است که در آن ما یک توصیف داریم که به یکی از زبان‌های توصیف مانند Z نوشته شده است. یک برنامه نهایی هم داریم که به یکی از زبان‌های برنامه‌نویسی متداول است. در روش تقلید ما یک تعدادی قوانین موردی[4]یا راهنما[5]داریم که به ما می‌گوید چگونه جملات[6]توصیف را به جملات برنامه تبدیل کنید. در واقع تعدادی قانون است که به ما می‌گوید اگر یک عبارت خاص در Z را مشاهده کردید، آنرا به یک یا چند دستور به زبان برنامه‌نویسی مورد نظر (مانند C#) تبدیل کن.

نقطه ضعف این روش هم مشخص است. این رویکرد، رویکرد کاملی نیست. یعنی ممکن است تمام موقعیت‌های توصیف شما را کاملا پوشش ندهد. به عبارت دیگر روش سیستماتیکی نیست و فقط تعدادی قوانین موردی دارد. بنابراین این روش نمی‌تواند به شما الگوریتمی بدهد که این الگوریتم تضمین کند هر توصیفی را به برنامه نهایی ببرد. زیرا قوانینی که دارد ممکن است یک توصیف را کاملا پوشش بدهد و یک توصیف را نتواند پوشش دهد (بخشی از توصیف را به برنامه نهایی ببرد و بخشی از توصیف را نبرد). یعنی مجبور خواهید بود به صورت دستی یا با روش‌های دیگر آن قسمت را به برنامه نهایی تبدیل کنید.

بنابراین روش تقلید شامل یک سری قوانین موردی برای تبدیل عبارات توصیفی[7]به دستورالعمل‌های برنامه‌نویسی[8]است. این مورد یک موضوع خوب برای تحقیق خصوصاً پایان نامه‌های کارشناسی‌ارشد است.

۲-۲-۲- روش پالایش

1 Animation
2 Refinement
3 Constrictive
4 ad-Hoc
5 Guide line
6 Statement
7 Specification statements
8 Program Statements

این روش بر خلاف روش تقلید، یک ضرب کار را انجام نمی‌دهد، بلکه در چند مرحله انجام می‌دهد. در روشهای پالایش شما یک حساب پالایش[1]دارید که شامل قوانین پالایش[2]است. این قوانین بر خلاف قوانین موردی که در روش تقلید استفاده می‌شد، تمامی موقعیت‌ها را پوشش می‌دهد و قطعی است. در این روش شما می‌توانید یکی از قوانین پالایش را به قسمتی از توصیف apply کنید (به عبارت دیگر آن قسمت را پالایش می‌کنیم). نکته قابل توجه این است که لزوما این قسمت به یک برنامه قابل اجرا تبدیل نمی‌شود. به عبارت دیگر فرض کنید یک توصیف دارید و همچنین لیستی از قوانین نیز دارید. قوانین را بررسی می‌کنید تا ببینید کدام یک از این قوانین به بخشی از توصیف قابل اعمال شدن هستند.

با استفاده از این روش یک عبارت توصیفی به یک عبارت توصیفی دیگر که پالایشی از عبارت اول است، تبدیل می‌شود. اتفاقی که در اینجا می‌افتد این است که اگر توصیف اولیه غیر قطعی بوده است، توصیف پالایش شده می‌تواند کاملا قطعی باشد و یا اینکه نسبت به توصیف اول قطعی‌تر شده باشد.

فرض کنید نام توصیف اولیه ما B باشد. توصیف به دست آمده را B' می نامیم.B' توصیفی است که به برنامه قابل اجرا نزدیک‌تر است. عمل پالایش آنقدر تکرار می‌شود که نهایتا به یک توصیفی برسیم که کاملا قابل اجرا است. در روش پالایش می‌گویند که برنامه‌ها زیر مجموعه‌ای از توصیف‌ها هستند.

معروف‌ترین روش پالایش، روش پالایش مورگان است که متناظر با آن حساب پالایش مورگان[3]نیز وجود دارد. حساب پالایش مورگان یک روش ساده است و روش بسیار خوبی است. این روش بر خلاف روش‌های تقلید، توصیف را محدود نمی‌کند. روش‌های تقلید فرض می‌کنند که توصیف شما ساختار خاصی را ندارد. به عبارت دیگر روش تقلید وقتی جواب می‌دهد که توصیف یک یا چند ساختار خاص را نداشته باشد. اما در حساب پالایش مورگان این مشکل را نداریم.

اما این روش یک مشکل عملی دارد و آن هم این است که وقتی که در یک حالت هستیم و به قوانین نگاه می‌کنیم، به عنوان مثال مشاهده می‌کنیم که از ۱۰۰ قانونی که حساب پالایش به ما داده است، ۵ مورد قابل اعمال است. در اینجا ابزار نمی‌تواند کمک کند. ابزار نیاز به انسان دارد و این انسان است که باید به ابزار بگوید کدام مورد باید انتخاب شود. اگر قرار باشد ابزار انتخاب کند، ممکن است به حالتی برسد که در آنجا دیگر هیچ قانونی قابل اعمال نباشد.

بنابراین باید بازگشت به عقب[4]انجام شود و اولین جایی که در درخت این تصمیم را اتخاذ کرده‌ایم، برسیم و آن تصمیم را کنار بگذاریم و تصمیم دیگری بگیریم. یعنی رفت و برگشت[5]در این روش زیاد است در عمل برای توصیف‌های بزرگ، این کار زمان زیادی می‌برد. به عنوان مثال ممکن است آنقدر درخت ما

1 Refinement Calculus
2 Refinement Rules
3 Morgan Refinement Calculus
4 Back Track
5 Back Track

بزرگ شود که حافظه کم بیاوریم. بنابراین اینگونه ابزارها، ابزارهای محاوره‌ای[1]هستند. به عبارت دیگر ابزار یک جاهایی نیاز به کمک انسان دارد. یعنی انسان با توجه به دانشی که دارد به ابزار کمک می کند. در مواردی حتی ممکن است انسان نیز نتواند کمک کند و به صورت تصادفی انتخاب کند.

این مهم‌ترین چالشی است که در پالایش داریم، یعنی نسبت به سایر روش‌ها، روش بسیار خوبی است و بیشترین استفاده را دارد. چون هم ساده‌تر است و هم تمام موقعیت‌ها را پوشش می‌دهد. فقط همین چالش را دارد که برای رفع آن از روش‌های هیورستیک استفاده می‌شود. با به کارگیری هیورستیک می‌توان حجم بازگشت به عقب را کم کرد.

۳-۲-۲- روش ساختی

روش آخر روش ساختی است. گاهی اوقات به روش ساختی، روش استخراج برنامه هم می‌گویند. ریاضیاتی که ما روی آن تکیه کرده‌ایم، ریاضیات کلاسیک است. یعنی مبتنی بر تئوری مجموعه‌ها است.

در تئوری مجموعه دو تا term ساده داریم که عبارتند از عضویت و تساوی. همچنین تعدادی اصول داریم که عبارتند از زیر مجموعه بودن، اجتماع، اشتراک و غیره.

این اصول توسط دو نفر به نام های Zernel و Franckel که به اختصار ZF نامیده می‌شوند، ابداع شده‌اند. این اصول اواسط صده پیش مورد نقد جدی قرار گرفت. به عنوان مثال در منطق کلاسیک PVT همیشه درست است. یعنی یا باران می‌بارد یا باران نمی‌بارد. اما به عنوان مثال در ریاضیات غیر کلاسیک باید دلیل بیاوریم. به عنوان مثال بگوییم پنجره را باز کن و ببین باران می‌بارد یا نه. به ریاضیات غیر کلاسیک، شهودگرا هم می‌گویند. یعنی علاوه بر نمادهایی که در ریاضی وجود دارد، باید ساختار ذهنی هم برای شهود در نظر بگیریم و به آن عینیت بدهیم.

بنابراین برخی از مسائل به صورت ساختی حل می‌شوند. به عبارت دیگر در این روش، در حین اثبات، روش را هم می‌بینیم. در روش‌های شهودی، اثبات کردن معادل با ساختن است. برای اینکه بگوییم یک موضوع درست است، باید آن را بسازیم. یعنی به ازای هر اثبات باید موضوع مورد نظر ساخته شود و برای آن نمونه آورده شود به این حوزه ریاضیات ساختی نیز می‌گویند. ریاضیات ساختنی دقیقا معادل با برنامه‌نویسی تابعی[2]است. برنامه نویسی تابعی نوعی پارادایم برنامه‌نویسی است که در آن "محاسبات" به صورت ارزش‌یابی توابع ریاضی در نظر گرفته می‌شوند.

ایده روش ساختی این است که شما یک توصیف می‌نویسید و سپس به جای اینکه آنرا با ریاضیات کلاسیک منطبق کنید، با ریاضیات ساختی تطبیق می‌دهد و در نهایت آن را اثبات می کنید. اما چون کار شما مبتنی بر ریاضیات ساختی بوده است، نتیجه اثبات شما یک برنامه خواهد بود. به عبارت دیگر به

---

1 interactive
2 Functional Programming

محض اینکه به وسیله ریاضیات ساختی ثابت می‌کنید یک چیزی درست است، خود به خود برنامه آن استخراج می‌شود.

برای اثبات از یک درخت اثبات استفاده می‌شود. در ریشه درخت برنامه ساخته می‌شود. ریاضیات ساختی مشکلات روش تقلید را ندارد. یعنی قائم به توصیفات و روش‌های خاص نیست و همه جا می‌توان از آن استفاده کرد. مانند روش پالایش هم نیست که بازگشت به عقب داشته باشد. اگر هم بازگشت به عقب لازم باشد، بسیار محدود است. فقط مشکلی که دارد این است که پایه ریاضی قوی می‌خواهد که مهندسین نرم‌افزار آنرا نمی‌دانند. در واقع باید ریاضی ساختی بدانند (نظیر تئوری انواع مارتینف و تئوری مجموعه‌های ساختی). بنابراین این روش نسبت به دو روش قبل کامل‌تر است. اما مشکلی که دارد آشنا نبودن مهندسین نرم‌افزار با مباحث تئوری مورد نیاز است. نکته دیگری هم که وجود دارد این است که این روش جوان‌تر است و مسائل حل نشده[1]آن بیشتر است و همچنین ابزارهای کمتری برای آن وجود دارد. به هر حال پتانسیل زیادتری نسبت به دو روش قبلی دارد و در آینده می‌تواند مهمترین روش صوری شود.

به عنوان تفاوتی از ریاضیاتی کلاسیک و ریاضیات ساختی به مثال زیر توجه کنید.

در ریاضیات کلاسیک می گوییم p^q وقتی درست است که هم p و هم q درست باشد. اما در ریاضیات ساختی می‌گوییم p^q، وقتی درست است که شما یک شیء از p و یک شیء از q بسازید. به عنوان مثال دیگر، در ریاضیات کلاسیک برای اینکه ثابت کنیم p درست است، می‌توانیم ثابت کنیم ~p غلط است. اما در ریاضیات ساختی این کار غلط است. برای اینکه ثابت کنیم p درست است باید یک نمونه از آن بسازیم.

نکته دیگر هم این که قوانین به کار رفته در ریاضیات ساختی از قوانین ریاضیات کلاسیک کمتر است و این موضوع یک مزیت است. به عبارت دیگر در ریاضیات کلاسیک می‌توان برنامه‌هایی که وجود خارجی ندارد (نظیر halting program) را توصیف کرد.

در درس‌هایی مانند نظریه زبان‌ها و ماشین‌ها و تئوری پیچیدگی مثال‌هایی از برنامه‌هایی که وجود خارجی ندارند و فقط از نظر تئوری می‌توانند وجود داشته باشند، ذکر شده است. اما در ریاضیات ساختی، قوانینی که برای توصیف چنین برنامه‌هایی هستند، حذف شده‌اند. بنابراین با ریاضیات ساختی نمی‌توان توصیف‌های غیر واقعی نوشت. جدول ۲-۱ سه روش تقلید، پالایش و ساختی را مقایسه نموده است.

---

1 Open Problem