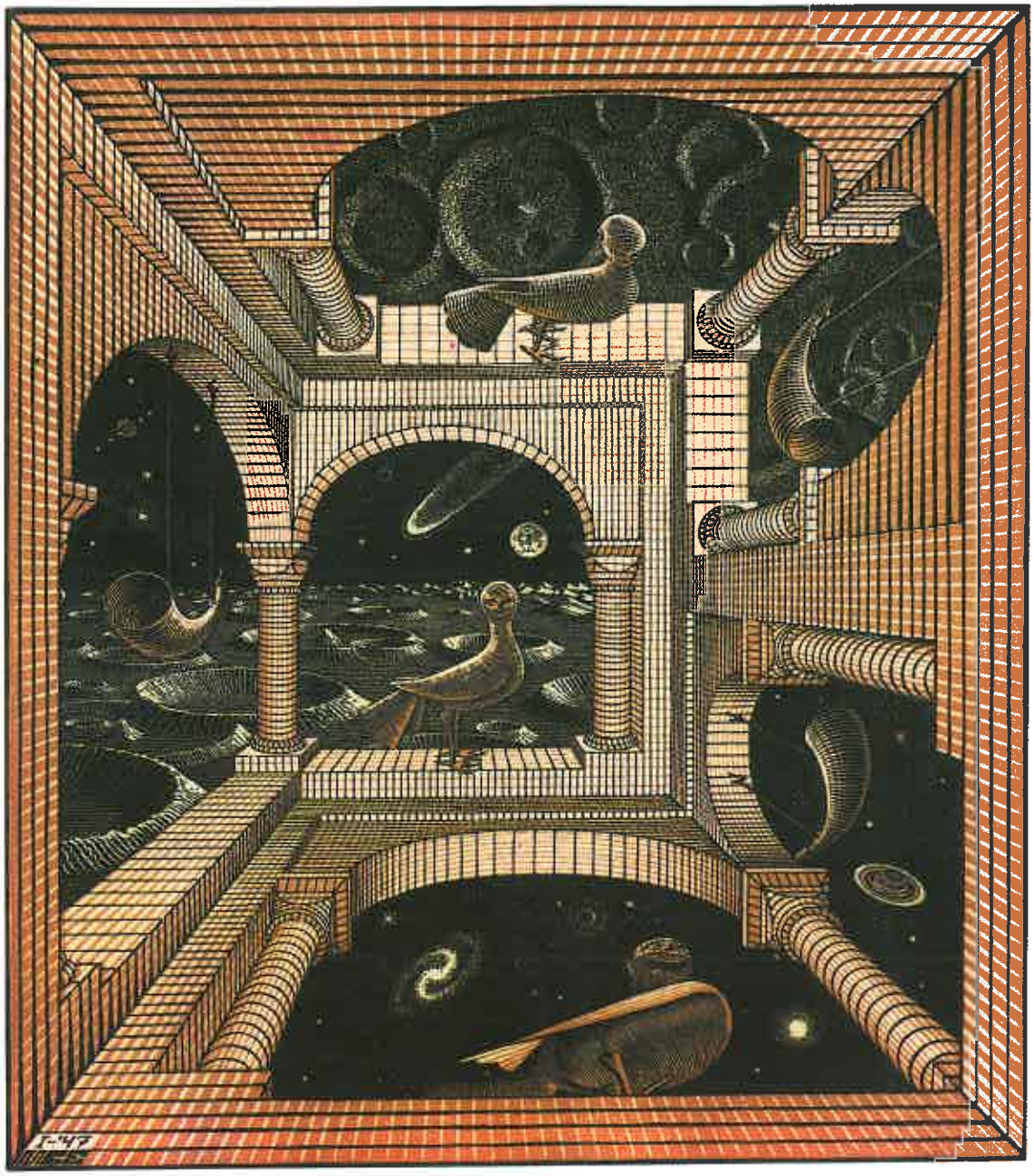
گزارش کامپیوتر

ماهنامهٔ انجمن انفورماتیک ایران
شمارهٔ ۳ سال شانزدهم ـ شمارهٔ پیاپی ۱۲۶ ، مرداد و شهریور، انتشار اسفند ۱۳۷۳، ۸۴ صفحه



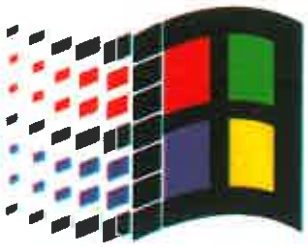
ویژه‌نامه شیءگرایی

- یک متدولوژی پیشنهادی برای تولید شیءگرا
- تاریخچه مختصر اسمال‌تاک
- معرفی شیءگرایی به غیرمتخصصان
- مصاحبه‌های اختصاصی با پلاگر و رندل و بوچ
- مدلی برای توسعه شیءگرا
- واژه‌نامه شیءگرایی

Microsoft®

WINDOWS™ فارسی

- نرم‌افزار اصل ارائه شده توسط شرکت تولید کننده (مایکروسافت)
- برخورداری از خدمات مایکروسافت همانند سایر کاربران در سراسر دنیا
- پیوستن به ۶۰،۰۰۰،۰۰۰ کاربر رسمی این محصول در سراسر دنیا
- امکان کار در محیط ۳۸۶ پیشرفته
- ارائه نرم‌افزارهای متعدد توسط شرکتهای نرم‌افزاری دیگر برای محیط استاندارد مایکروسافت
- ارائه برنامه‌های کاربردی جانبی
- پشتیبانی کاراکترستهای موجود فارسی
- Upgrade
- استفاده از فونتهای میزان‌پذیر
- قابلیت چند زبانه
- آموزش رایگان
- کتاب راهنما و آموزش فارسی
- راهنمای هوشمند کاملاً فارسی
- تقویم هجری شمسی
- وژه‌پرداز فارسی
- مدیر پرونده فارسی
- صفحه کنترل فارسی
- کاردکس فارسی
- پرپاتی سریع، یادگیری و استفاده آسان
- یادداشت فارسی
- پشتیبانی کلیه چاپگرهای موجود
- چاپ تا ۶۰۰ کاراکتر در اینچ
- و صدها قابلیت دیگر که فقط در این نسخه می‌توانید آنها را بیابید ...



Microsoft® WORD فارسی نسخه ۶

- امکان منحصر بفرد Intellisence™ (فهم هوشمندانه کاری که می‌خواهید بکنید و ارائه نتیجه آن با سریع‌ترین و ساده‌ترین شکل ممکن)
- ارائه بیش از ۳۰ فونت فارسی مختلف و متنوع برای این محصول و ویندوز مایکروسافت توسط شرکتهای دیگر
- ارتباط با پروندهای بانکهای اطلاعاتی مختلف برای استفاده و چاپ اطلاعات آنها با فرمت تعیین شده
- امکان تعریف فریم متن یا گرافیک با امکان تغییر مکان فریم و صفحه‌بندی و ستون‌بندی اتوماتیک
- تعریف فرم برای اطلاعات ثابت (مانند فرم فکس)
- استیل‌های مختلف نوشتاری بصورت از پیش تعریف شده
- چاپ نامه برای لیست دلخواه یا مشخصات جداگانه
- سایه‌گذاری
- جدول‌بندی
- سرصفحه و پاصفحه زوج و فرد
- حروف مخصوص و سمبل‌ها
- انجام عملیات روتین بصورت خودکار
- عدد گذاری
- نشان گذاری
- نمودار
- ستون‌بندی
- چاپ پاکت نامه و برچسب
- قالب‌بندی حروف و پاراگراف
- اندنویس
- استفاده از لایه‌های مختلف برای چاپ شکل یا آرم در زمینه متن
- ایجاد فهرست بصورت خودکار
- ایجاد ایندکس بصورت خودکار
- استفاده از منوها و امکانات Excel بدون نیاز به تغییر محیط
- ابزار طراحی
- کادر بندی
- خط کشی
- شکست خودکار جملات
- راهنما
- نصب دلخواه بصورت قسمت‌بندی شد
- بازگشت عمل چند مرحله‌ای
- شمارش کلمات
- خط کش افقی و عمودی
- و صدها قابلیت دیگر که فقط در ای نسخه می‌توانید آنها را بیابید ...

نرم‌افزار کامپیوتر نماینده رسمی مایکروسافت در ایران تلفن : ۹۳۱۵ ۲۲۲ ۰۶۹۱ ۲۲۵ فکس : ۲۹۸۹ ۲۲۷

گزارش کامپیوتر

ماهنامهٔ انجمن انفورماتیک ایران

(فعلا هر دو ماه یکبار منتشر می‌شود.)

صاحب امتیاز: انجمن انفورماتیک ایران

مدیر مسئول و سردبیر: ابراهیم نقیب‌زاده مشایخ

مقاله‌ها و گزارشهای تالیفی یا ترجمه خود را برای گزارش کامپیوتر بفرستید. مطالب ارسالی را با خط خوانا (یا ماشین‌شده) بر یک روی کاغذ بنویسید. فاصله کافی برای افزودن اصلاحات ویرایشی در بین خطها و حاشیه در نظر بگیرید. در صورت ارسال مطلب ترجمه شده، یک نسخه از اصل مطلب را نیز ارسال دارید. شکلها باید با روان‌نویس مشکی ترسیم و به‌صورت آماده چاپ ارسال شوند. مطالب ارسالی پس‌فرستاده نمی‌شوند.

مسئولیت مطالب گزارش کامپیوتر با نویسندگان است و لزوماً نشان‌دهنده نظر انجمن انفورماتیک ایران نیست. ذکر نام شرکتها و محصولات در مطالب گزارش کامپیوتر برای ارائه اطلاع به خوانندگان است و نباید به معنی تائید انجمن از آنها تلقی شود.

تمام حقوق به انجمن انفورماتیک ایران متعلق است. نقل مطالب گزارش کامپیوتر با ذکر منبع بلامانع است.

مطالب این شماره از گزارش کامپیوتر با استفاده از برنامهٔ LaTeX بارگ حروفچینی و صفحه‌بندی شده‌است.

حروفچینی و صفحه‌بندی: دادمکاوی ایران

لیتوگرافی: پگاه

چاپ: صنوبر

دفتر مجله: تهران، دکتر فاطمی، بروین اعتصامی، کوچه دوم، شماره ۲۱، طبقهٔ سوم.

ساعات کار: هرروز بجز پنج‌شنبه‌ها از ۹ تا ۱۶.

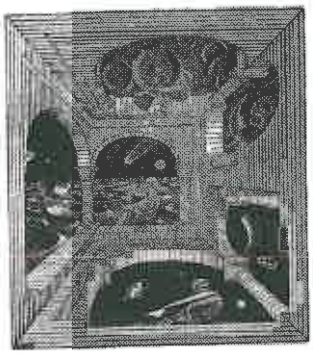
شمارهٔ تلفن: ۶۵۶۶۸۷

شمارهٔ حساب بانکی: ۱۱۱۷ بانک صادرات شعبه ۱۶۵۲، دانشگاه صنعتی شریف، تهران.

نشانی پستی: گزارش کامپیوتر، صندوق پستی ۱۹۶-۱۴۱۵۵، تهران.

پست الکترونیک: isl@irearn.bitnet

برای مکاتبه با گزارش کامپیوتر فقط از نشانی پستی استفاده کنید.



ویژه‌نامه

شیءگرایی

در این شماره

شیءگرایی، یک جهان‌بینی نرم‌افزاری

در کنار تمام ویژگیهایی که ممکن است به شیءگرایی نسبت دهیم، درک شیءگرایی به‌عنوان یک «جهان‌بینی نرم‌افزاری» از اهمیت اساسی برخوردار است. شیءگرایی روند جدایی‌ناپذیر زمان ما است. شیءگرایی شیوه نگرش طراح به یک سیستم، تشخیص مفاهیم مجرد، و یافتن مدلی برای توصیف مسئلهٔ واقعی در «فضای مفهومی مسئله» است. سپس راه‌حلهایی را خلق می‌کند که برخی از آنها را در «فضای راه‌حل» به‌مرور به‌صورت سیستماتیک حذف یا پالایش می‌نماید.

تناظر میان قلمروی مسئله و اجزای قلمروی راه‌حل، ساختار ذهنی مناسبی است که در تجزیه پیچیدگیهای مسئله ما را یاری می‌کند. ولی مدیریت پیچیدگی تنها در تفکیک مسئله نمود پیدا نمی‌کند، بلکه در «چگونگی» تفکیک مسئله و در نهایت روش نمایش حاصل تفکیک متجلی می‌شود و نیل به راه‌حل را آسانتر می‌نماید. این راه‌حل به‌نوبه خود می‌تواند در طیف پیچیدگی نسبی که تمام عوالم مفهومی علم کامپیوتر را در برمی‌گیرد به‌عنوان راه‌حلی ساده‌تر یا پیچیده‌تر طبقه‌بندی گردد.

ما در علم و مهندسی کامپیوتر درصدد حل مسائل و مشکلات دنیای واقعی هستیم. این کار را با خلق مدلی از مسئله در آن چه «قلمروی مسئله» می‌نامیم، انجام می‌دهیم. سپس با استفاده از سازه‌های ذهنی، مفاهیم و تکنیکهایی که ابزار کارمان هستند، در «قلمروی راه‌حل» مبادرت به خلق مدلی از مسئله می‌نماییم. برای این که راه‌حل، قابل استفاده، بجا و قابل فهم باشد، می‌بایست سعی شود ارتباط و تناظر میان قلمروی مسئله و قلمروی راه‌حل برقرار شود و تا حد امکان به یکدیگر نزدیک باشند.

شیءگرایی با گذر نرم و روان از مراحل آنالیز، طراحی، پیاده‌سازی، و آزمون خود را از دیگر نگرشهای نرم‌افزاری کاملاً متمایز می‌سازد. گذر از یک مرحله به مرحله دیگر هیچ تغییری را در ابزار مفهومی ایجاب نمی‌کند. همان ساختارهای مفهومی که در یک مرحله خلق می‌شوند، در فاز جدید صرفاً پالایش و بسط داده می‌شوند. این گذر از فاز به فاز با چنان سهولت و زیبایی صورت می‌گیرد که فاصله مفهومی (Conceptual Gap) ناشی از نگاشت فضای مسئله به درون (Into Mapping) فضای راه‌حل و مجدداً بر روی (Onto Mapping) فضای مسئله (فرارونده دنیای واقعی) را تا حد بسیار زیادی تقلیل می‌دهد.

طبیعت تکرارپذیر (Iterative) و افزایش‌پذیر (Incremental) فرارونده توسعه نرم‌افزار گرچه همواره به‌شدت مورد تایید و تاکید قرار گرفته، هنوز از لحاظ نظری به‌خوبی درک و تعریف نشده است. هر چه شناخت سرشت این فرارونده دقیقتر و عمیقتر گردد، پیشرفتهای سریع‌تری در کیفیت همه‌جانبه نرم‌افزارهای تولید شده و نیز تحقق اهداف مهندسی نرم‌افزار حاصل خواهد شد. از سوی دیگر، مشتریان و کاربران نهایی سیستمها رضایت بیشتری خواهند داشت. عوامل این رضایت، راه‌حلهای تعبیه شده در برنامه‌هایشان است که روند کاری، تجاری و علمی آنها را به‌نحو کاملتری پشتیبانی می‌نماید و نه تنها برای بشریت معضلی ایجاد نمی‌کند بلکه برای انسانها مفید فایده نیز خواهد بود.

نگرش شیءگرا گامی تکاملی در جهت فوق به‌شمار می‌رود. با این وجود، در سکوهای سخت‌افزاری و نرم‌افزاری ناهمگون که دارای شفافیت شیئی (Object Transparency) هستند، برای رسیدن به انسجام و یکپارچگی گسترده‌تر اجزایی با قابلیت استفاده مجدد، باید گام‌های فراتری در مهندسی نرم‌افزار برداشت. منظور از «شفافیت شیئی» امکان دستیابی سریع، هوشمندانه و بی‌مانع به اشیایی است که در سیستمهای گوناگون یا توزیع شده پراکنده‌اند. قدم بعدی در تکامل مفاهیم مهندسی نرم‌افزار، این هدف را برآورده خواهد ساخت. یکی از ارکان بنای جدید مهندسی نرم‌افزار به‌طور قطع حاصل پژوهشها و محصولات مبتنی بر نگرش شیءگرا و بر مبنای تکنولوژی شیءگرا خواهد بود.

* * *

خلاصه‌ای از مقالات این ویژه‌نامه

رشد فزایندهٔ جذابیت و توجه به شیءگرایی در دهه اخیر، به این دلیل بوده است که این روش حیطهٔ وسیعی از امکانات نوین را در اختیار مهندسین نرم‌افزار قرار داده و مزایای قابل توجه آن، مرزهای محدود روشها و نگرشهای پیشین را

پشت سرگذاشته است. همراه با این رشد، معضلات و مسائل عدیده‌ای پدید آمده‌اند که برای حل آنها، متخصصین شیءگرایی در سراسر جهان مبادرت به تشکیل گروههای تخصصی، جهت استانداردسازی و ایجاد انسجام بیشتر میان محیطها، زبانها، سیستمهای عامل و نرم‌افزارهای تجاری/علمی که با نظر به این تکنولوژی پدید آمده بودند، نموده و رسمیت عمیقتری به مفاهیم و محصولات شیءگرا بخشیده‌اند. نگرش شیءگرا بر این اساس استوار است که دنیای نرم‌افزار را می‌توان به‌عنوان سیستمهایی در نظر گرفت که برهمکنش هوشمندانه‌ای میان اشیای آن صورت پذیرفته و این اشیاء علاوه بر دارا بودن مجموعه صفات و افعال مربوط به خود، دارای هویتی مستقل از یکدیگر و حتی کل سیستم هستند و مدلسازی روابط و رفتار متقابل آنهاست که امکان حل مسائل کامپیوتری دنیای واقعی را میسر می‌سازد. ویژه‌نامهٔ حاضر به همین نگرش شیءگرا اختصاص یافته است.

در این ویژه‌نامه، طیف وسیعی از موضوعات، در مقالات درج شده است که بر روی هم مجموعه، ارزشمندی را تشکیل می‌دهد. ذیلاً سعی کرده‌ام تا اهم مطالب هر مقاله را به اختصار بازگو نمایم. مقالات را به‌صورت زیر دسته‌بندی نموده‌ایم:

۱ـ معرفی شیءگرایی به غیرمتخصصین

۲ـ نبرد شیءها

۳ـ مدلی برای توسعهٔ شیءگرا

۴ـ تاریخچهٔ اجمالی زبان اسمال‌تاک

۵ـ یک متدولوژی پیشنهادی برای تولید شیءگرا

۶ـ آیا ایده‌های قدیمی هنوز اعتبار دارند؟

۷ـ وراثت و گوشه‌ای از آن در C++

۸ـ واژه‌نامه پیشنهادی

۹ـ گزارش گروه شیءگرایی

لازم به ذکر است که موضوعات فوق، سطوح مختلفی از خوانندگان را پوشش می‌دهد؛ از خوانندگانی که می‌خواهند قدم به عرصهٔ شیءگرایی بگذارند ـ که مایلند با مفاهیم شیءگرایی آشنا شده و یا می‌خواهند تجربیات و معلومات خود را دقیقتر و ژرفتر نمایند ـ تا متخصصین شیءگرایی که در حال استفاده روزمره از این تکنولوژی نرم‌افزاری هستند و در پروژه‌های خود به دنبال راه‌حلی جدید برای استفاده از یک مدل توسعه نرم‌افزار کاملاً شیءگرا می‌باشند.

در مقالهٔ اول، به‌نام «معرفی شیءگرایی به غیرمتخصصین» (رامین عرفانیان)، مفاهیم اولیهٔ شیءگرایی به زبانی ساده و با مثالهایی روزمره ارائه شده و مسائل بنیادی این مبحث مطرح گشته است. در طی این مقاله روشهای حل مسائل از دید نرم‌افزار مطرح شده و نواقص کلی این روشها گوشزد شده است. سپس، دید شیءگرا به‌عنوان یک راه‌حل اساسی برای رفع نواقص روشهای دارای قدمت بیشتر مطرح می‌شود. در پایان، برخی معیارهای کیفیت نرم‌افزار توضیح داده می‌شود.

مقالهٔ دوم، ترجمهٔ خوبی است از «نبرد شیءها» که در ماه می ۱۹۹۴ در مجله بایت به چاپ رسید. این مقاله، روندهای گاهی غیر همگرا در دنیای استانداردها و محصولات (به‌خصوص زبانهای) شیءگرا را بررسی نموده و معیارها و موضع هر کدام از طرفین صاحبنظر و محصول را تبیین می‌نماید. در این مقاله به‌خوبی روشن می‌شود که چگونه رقبای صنعت نرم‌افزار مشغول تبلیغ استانداردهای صنعتی خود بوده وگاهی این جدالها صرفاً بر سر فروش محصولات بروزکرده و شدت می‌گیرد.

در «مدلی برای توسعه شیءگرا» (علی ارسنجانی)، یک مدل نرم‌افزاری کاملاً شیءگرا ارائه شده است. در ابتدا، اهمیت ایجاد یک فرایند توسعه نرم‌افزار که به‌صورت بنیادی، شیءگرا باشد، توضیح داده می‌شود. علیرغم وجود تعداد بسیار زیادی متدولوژی شیءگرا، فقدان یک مدل توسعه شیءگرا کاملاً مشهود می‌شود. متدولوژیهای متداول و اساسی برشمرده شده و این مسئله مطرح می‌گردد که: تا زمانی که این متدولوژیها بر مدلهایی مانند مدل آبشاری، تکاملی و یا نمونه‌سازی، استوار باشند و از طریق آنها مورد استفاده قرارگیرند، اهم توان و قدرتهای خاص شیءگرایی مستور و عقیم خواهد ماند. در حالیکه، مزایای شیءگرایی زمانی کاملاً ظاهر می‌شوند و می‌توانند اهداف مهندسی نرم‌افزار را تأمین کنند که در قالب یک مدل شیءگرا بهره‌برداری و بکارگرفته شوند. متدولوژی مانند راننده است و مدل مانند ماشین؛ از یک راننده، هر چند که ماهر باشد، نمی‌توان توقع داشت که با یک ماشین نامناسب و ضعیف در شرایط نامساعد جاده بتواند در مسابقه‌ای سخت برنده شود. اصول مهندسی نرم‌افزار به‌صورت اجمالی بررسی می‌شود و سپس مفاهیم اساسی شیءگرایی به‌همراه یک طبقه‌بندی از متدولوژیهای موجود ارائه می‌گردد. بکارگیری مدلهای سنتی مانند مدلهای آبشاری، تکاملی و یا نمونه‌سازی امکانپذیر است (اکثریت از همین‌ها استفاده می‌کنند) ولی مزایا و توانهای ویژهٔ شیءگرایی، بنا به طبیعت این مدلها، در چنین زمینه‌ای قادر به رشد و نمو و بهره‌دهی نمی‌باشند. عدم استفاده از یک مدل ماهیتاً شیءگرا آنتروپی (Entropy یا بی‌نظمی اطلاعاتی/معرفتی) را وارد پروژه می‌کند. این آنتروپی موجب می‌گردد تا مشکلات زمانبندی، کیفیتی، عدم تناسب طراحی با پیاده‌سازی و خواسته‌های کاربران از طرف دیگر بروز نموده و کلاً پروژه‌ها به‌سوی عدم موفقیت سوق داده شود؛ همانگونه که اخبار آن از گوشه و کنار جهان می‌رسد. در انتها، مزایا و مسائل کاربرد شیءگرایی در پروژه‌های نرم‌افزاری مورد بحث قرار می‌گیرد.

در «تاریخچهٔ زبان اسمال‌تاک»، روند تاریخی تکامل یکی از اولین زبانهای شیءگرا شرح داده می‌شود و وجوه تمایز آن با دیگر زبانها، به‌همراه دیدگاههای خاص بوجود آورندهٔ آن مورد بررسی قرار می‌گیرد.

در مقاله بعدی تحت عنوان «یک متدولوژی پیشنهادی برای تولید شیءگرا»

(محسن شریفی)، توضیح داده می‌شود که نگرش شیءگرا امید بیشتری برای حل فعالیتهای تولید نرم‌افزار را دارا می‌باشد. به این نکته اشاره می‌گردد که اکثر متدولوژیها به یک فاز از مدل نرم‌افزار پوشش می‌دهند. ((البته این امر در مورد نسل اول متدولوژیهای شیءگرا صادق است؛ مانند اولین متدولوژیهای Shlaer and Mellor, Coad and Yourdon. در حالیکه پس از آن، متدولوژیهای متعدد و غنی‌ای که در نسل بعدی متدولوژیهای شیءگرا به‌صورت متکاملتری ظاهر شدند، سعی در پوشش تمام چرخه زندگی نموده‌اند.))[1] متدولوژی ارائه شده در این مقاله ترکیبی از روشهای نمایش‌سازی چند متدولوژی برده و در قالب آنها، یک مثال شبیه‌سازی (کنترل ترافیک هوایی) برای توضیح دو بخش آنالیز و طراحی مورد استفاده قرار گرفته است.

در «آیا ایده‌های قدیمی هنوز اعتبار دارند؟» رامین عرفانیان تجربیات خود را از یک میزگرد بازگو نموده و نظرات خود را پیرامون مباحث مطرح شده به رشتهٔ تحریر درآورده است. این میزگرد در سال ۱۹۹۲ در شهر لندن برگزار شد. شرکت‌کنندگان عبارت بودند از رامین عرفانیان، Tim Wolf، Michael Simpton و Richard Keith. در این جلسه Tim Wolf و رامین عرفانیان به‌عنوان مخالفین ایده‌های قدیمی سخنرانی نمودند که گوشه‌ای از آن در مقالهٔ حاضر آورده شده است.

محمودرضا زارع مفهوم وراثت را به‌همراه مثالهایی به زبان C++ در «وراثت و گوشه‌ای از کاربرد آن در C++» ارائه می‌نماید. این مثالها از ساده به مشکلتر انتخاب شده است تا کاربرد وراثت را در بخشی از یک سیستم فرضی طراحی به‌کمک کامپیوتر توضیح دهد.

واژه‌نامهٔ پیشنهادی ارائه شده توسط دکتر محسن شریفی حاصل ارزشمندی است از تلاش برای یکپارچه نمودن تعابیر و واژه‌های شیءگرایی که به فارسی ترجمه می‌شوند. این مجموعه بدون هیچگونه ویرایش محتوی آن، و با تشکر از دکتر شریفی به‌عنوان یک پیشنهاد چاپ شده است. نظر به اینکه گروه شیءگرایی درصدد تدوین یک واژه‌نامهٔ شیءگرایی می‌باشد، مجموعهٔ حاضر به‌عنوان اولین پیشنهاد برای بحث و تحقیق پیرامون ترجمهٔ صحیح واژه‌های شیءگرایی تلقی گردیده و از علاقمندان دعوت می‌شود در جلسات گروه شیءگرایی، درگروه ویژه‌ای که برای تدوین واژه‌نامه فعالیت خواهد نمود شرکت نمایند.

در «گزارش گروه شیءگرایی» ایجاد گروه شیءگرایی تحت نظر علی ارسنجانی و رامین عرفانیان، به‌عنوان یک زیرگروه تخصصی انجمن انفورماتیک ایران اعلام گشته و اهداف این گروه تخصصی (در پی تشکیل دو جلسه عمومی) معرفی می‌گردد. عمده هدف تشکیل این زیرگروه ارتقاء آگاهی تخصصی دست‌اندرکاران نرم‌افزار از امکانات و مسائل این تکنولوژی بوده و نیز ایجاد زمینه‌ای مناسب برای تبادل تجربیات و نظرات استفاده‌کنندگان این تکنولوژی می‌باشد.

در پایان، مصاحبه‌هایی با چند تن از متخصصین شیءگرا، که از طریق پست الکترونیکی انجام گرفته، چاپ شده است. افراد مصاحبه‌شونده عبارتند از: Grady Booch و C.J. Plauger و Scott Randall. اولین شخص، از پیشکسوتان تکنولوژی شیءگرایی در سطح بین‌المللی محسوب می‌شود. دومین شخص، سالها در شرکت بل (Bell Laboratories) مشغول به‌کار بوده است و با Brian Kernighan یک کتاب معروف به‌نام Elements of Programming Style به رشتهٔ تحریر درآورده است و اکنون سردبیر ارشد مجلهٔ C Users Journal می‌باشد. فرد آخر، یا اسکات رندال در شرکت مایکروسافت مسئول بخش پشتیبانی از Microsoft Foundation Classes و مدیر گروه AFX می‌باشد.

علی ارسنجانی

۱) سردبیر فنی

ایجاد گروه تخصّصی

شیءگرایی

بدین وسیله به‌اطلاع می‌رساند که گروه تخصّصی «شیءگرایی» زیر نظر کمیتهٔ فعالیتهای علمی و فنّی انجمن تشکیل یافته است. چارچوب فعالیتها و اهداف این گروه تخصّصی به قرار زیر است:

۱. ارتقاء آگاهی تخصصی کارشناسان وعلاقمندان به:

۱ـ۱. به‌کارگیری (عملی): مزایا و مشکلات به‌کارگیری تکنولوژی شیءگرا؛ استفاده صحیح از این تکنولوژی و اجتناب از به‌کارگیری ناصحیح آن.

۱ـ۱ـ۱. فراهم نمودن تسهیلاتی برای معرفی مفاهیم، تکنیکها، ابزار، متدولوژیهای شیءگرا و سطوح مبتدی، متوسط، پیشرفته، چنانکه توسط کاربران، کارشناسان و پژوهشگران شیءگرایی استفاده گشته و توصیه می‌گردد.

۱ـ۱ـ۲. ترویج به‌کارگیری متدولوژیها، تکنیکها، محیطها و زبانهای شیءگرا به‌عنوان وسیله‌ای برای ارتقاء سطح کیفی نرم‌افزارهای تولید شده در ایران.

۱ـ۲. شناخت مفاهیم (نظری): ترویج مفاهیم و معانی رهیافت شیءگرایی.

۱ـ۳. ایجاد روند کانالیزه شده و هدفمند برای ترویج و اشاعه فرهنگ شناخت و به‌کارگیری صحیح شیءگرایی از دید مهندسی نرم‌افزار.

۲. فراهم نمودن زمینه مناسبی برای تبادل نظر و تجربیات کارشناسان و علاقمندان:

۲ـ۱. ایجاد انگیزه‌ای برای ارتباط علمی میان کارشناسان و علاقمندان در جهت افزایش میزان کاربرد این تکنولوژی نرم‌افزاری در سطحی وسیعتر.

۲ـ۲. ایجاد انگیزه و زمینه برای خلاقیت و نوآوری در به‌کارگیری مفاهیم و ابزار، توسعه سیستمهای نرم‌افزاری براساس الگوی مفهومی (Paradigm) شیءگرا، تشویق پژوهش در مفاهیم اساسی روشهای مبنایی شیءگرایی، تشویق تدریس مفاهیم و کاربردهای عملی.

۳. ایجاد رابطه با سازمانهای بین‌المللی ذیصلاح و دست‌اندرکار مسائل شیءگرایی.

این گروه هم‌اکنون زیر نظر آقای رامین عرفانیان و آقای علی ارسنجانی تشکیل گردیده و سه جلسه از گردهمائیهای آن برگزار شده است. علاقه‌مندان به عضویت در این گروه تخصّصی می‌توانند با آقای عرفانیان (۷۵۰۹۲۶۴-۷۵۰۵۱۶۹ و یا آقای ارسنجانی (۸۷۳۱۷۳۹) تماس حاصل نمایند.

معرفی شیءگرایی به غیرمتخصصان

رامین عرفانیان*

مقدمه

کامپیوتر چنان در زندگی ما ریشه دوانده است که این روزها تقریباً همه از کامپیوتر به‌شیوه‌های مختلف استفاده می‌کنند. یا شغل تمام وقت شما برنامه‌نویسی یا تدریس است که در این صورت آگاهی از تکنولوژیهای جدید برایتان اساسی است، یا در بخش بزرگی از مسئولیت خود با کامپیوتر سروکار دارید مانند فروشندگان کامپیوتر، مسئولان سیستم، مدیران، یا از کامپیوتر به‌عنوان سرگرمی استفاده می‌کنید و یا اینکه فقط یک کاربر صرف هستید و تنها چیزی که می‌خواهید این است که کارتان انجام شود ولی نمی‌توانید از موج پیش‌روندهٔ اختراعات و پیشرفتها که رسانه‌های همگانی به‌آنها توجه فراوان دارند مصون بمانید. وضعیت شما در دنیای کامپیوتر هرچه که باشد، مجبورید مطالب بسیاری را دربارهٔ تکنولوژیهای جدید و غیرمتعارف بشنوید و بخوانید. یادگیری و فهم بعضی از آنها مانند Video Graphic Adapter ساده و بعضی دیگر مانند شیءگرایی بسیار دشوار و پیچیده است.

این مقاله برای کسانی است که دربارهٔ تکنولوژی شیءگرایی بسیار شنیده‌اند ولی ایدهٔ روشنی از آن ندارند. ممکن است چندین بار از خود پرسیده باشید که شیءگرایی چیست؟ از کجا آمده است؟ آیا چیز جدیدی است؟ و یا بسیاری سئوالات دیگر که من سعی می‌کنم در این مقاله به‌شما پاسخ دهم.

بسیاری از مردم نمی‌دانند که کامپیوترها جعبه‌های بی‌صدایی از مدارهای الکترونیکی هستند. آنچه که واقعاً کامپیوتر را وادار به‌کار می‌کند نرم‌افزار است که بر روی سخت‌افزار اجرا می‌شود. مهندسی نرم‌افزار هنر خلق این عناصر بسیار حیاتی در دنیای محاسبات است. بروس باند می‌گوید: «در دههٔ ۱۹۹۰ دو کالا حائز اهمیت هستند یکی نفت است و دیگری نرم‌افزار، و در حال حاضر جایگزینهایی برای نفت وجود دارد.» ژنرالهای ارتش تاکتیکهای جنگی متفاوتی را برای موقعیتهای مختلف تنظیم می‌کنند. تاکتیکهای مورد نیاز در صحرا با تاکتیکهایی که در جنگل مورد نیاز است، به‌طور کلی تفاوت دارند. سلاحها و سربازها از موقعیتی به‌موقعیت دیگر متفاوت هستند ولی نتیجهٔ مطلوب نهایی همواره یکی است و آن پیروزی در نبرد است.

مانند ژنرالهای ارتش، مهندسین نرم‌افزار برای رسیدن به‌اهدافشان روشهای متفاوتی را به‌کار می‌برند، رهیافت شیءگرا یکی از این روشهاست.

اکثر ما نه آموزش لازم و نه وقت آن را داریم که واقعاً بفهمیم روشهای شیءگرا دقیقاً به چه معنایی هستند. ممکن است مطالبی اینجا و آنجا خوانده باشید اما تا وقتی که عمیقاً به آموختن برنامه‌نویسی نپردازید، تقریباً غیرممکن است که استلزامهای شیءگرایی را درک کنید. به‌هر حال، روشهای شیءگرا چه هستند؟ چرا چنین نام غیرطبیعی برای آن انتخاب شده است؟ اول از همه باید همیشه به‌یاد داشته باشید که اسامی در دنیای علم کامپیوتر؛ از شخصی گرفته می‌شود که فکر اولیه از او بوده است و یا از جایگزینیهای خوب دیگر انتخاب می‌شود که جای نام مخترع را می‌گیرد.

اجازه بدهید کلمهٔ شیءرا بررسی کنیم. تعریف واژه نامهٔ Oxford Advanced Learner این است: «چیزی که می‌توان آن را دید و یا لمس کرد، شخص یا چیزی که اعمال، احساسات و یا افکار متوجه آن است.» آخرین توصیف را در نظر بگیرید. شیءچیزی است که عمل متوجه آن است. اما دربارهٔ کلمهٔ oriented، این واژه کوتاه شدهٔ orientated است و در واژه نامهٔ Chambers maxi به‌این صورت تعریف شده است: «به‌طور طبیعی از موقعیت کسی آگاه بودن، هدایت شده (به‌سوی).» مثالی از این مورد جامعهٔ مرد ـگرا می‌باشد. بنابراین شیءگرا یعنی هدایت شده به‌سوی چیزهایی که هدف قرار گرفته شده‌اند. اگر چه توصیف خیلی واضحی نیست اما در مجموع قابل فهمتر از عبارت شیءگرا می‌باشد. برای درک کامل این تعریف باید تا اواسط دههٔ ۱۹۵۰ به‌عقب بازگردیم، زمانی که برنامه‌نویسان می‌بایست در سطح ماشین یا زبان ماشین[1] برنامه بنویسند. ما انسانها مجبور بودیم به‌زبان کامپیوترها فکرکنیم و سیستمهای جدید و یا لااقل متفاوت عددنویسی را مانند مبنای دو و یا مبنای شانزده به‌کار بریم. این مبناهای مختلف عددی بسادگی به‌کار می‌رفتند زیرا این شیوه، بهترین روشی است که کامپیوتر چیزها را می‌فهمد. برنامه‌نویسان مجبور بودند فکر کرده و مسائلی را در قالب موضوعات مختلف که از منابع روزمره و معمولی می‌آیند حل کنند. می‌توانید تصور کنید که نه تنها برنامه‌نویس باید مسأله را به‌کامپیوتر بفهماند، بلکه باید فکر کند و بر اساس معماری سخت‌افزاری که معمولاً به‌معنای صفرها و یکها است، برنامه بنویسند. این کار علاوه بر توانایی درک مسأله، به‌ترجمهٔ راه حلها به‌زبانی که برای ما بسیار غیرطبیعی است نیز نیاز دارد. بنابراین آنچه اتفاق افتاد این است که شخص باهوشی

*) اصل این مقاله به انگلیسی نوشته شده و توسط آقای سعید زین‌العابدین به فارسی ترجمه شده است.

۱) زبان ماشین ابتدایی‌ترین زبان برنامه‌سازی است که بر پایهٔ دستورات ابتدایی که برای پردازنده مرکزی قابل فهم هستند، قرار دارد.

زبان اسمبلی[۲] را اختراع کرد (زبان یادمانها). سپس شخص دیگری که باهوشتر بود زبان سطح بالا را اختراع کرد و دوباره کسی که واقعاً باهوش بود الگوریتمها (مجموعه‌ای از دستورالعملها) را بسته‌بندی کرد و آن را رویه، تابع، زیرروال، روال و یا هر چیز دیگری که شما دوست دارید نامید و دوباره شخصی که واقعاً باهوشتر بود انواع مختلف داده‌ها را کنار هم قرار داد و آن را رکورد نامید و سرانجام شخص باهوشی داده‌ها و الگوریتمها را با هم بسته‌بندی کرد و آن را شیءنامید.

می‌توانید ببینید که شیءگرایی، علیرغم اینکه بسیاری از مردم می‌پندارند باید چنین باشد، یک اختراع انقلابی نیست، شیءگرایی فرآروندی تکاملی است که در مراحل اولیه علم کامپیوتر شروع شده و هنوز در حال پیشرفت است ...

نکته حاشیه‌ای

علم کامپیوتر با اطلاع اندکی از احتمالات شروع شد. تا زمان اخیر کوششی برای پیش‌بینی آینده و سعی برای وفق یافتن با آن صورت نگرفته است. در مراحل اولیه دانشمندان فقط به‌قسمت فکری مسائل می‌پرداختند و هیچگاه سعی نکردند که راه‌حلها را ساده‌تر کنند تا کسانی نیز که از دانش کمتری برخوردارند آنها را بفهمند. البته اگر علم کامپیوتر مانند فیزیک هسته‌ای بود افراد معمولی هیچگاه نمی‌خواستند که دربارهٔ آن چیزی بدانند، اما امروزه بچه‌های هشت ساله سعی می‌کنند که از کامپیوتر استفاده کرده و حتی آن را برنامه‌ریزی کنند. بنابراین نسخه‌ای ساده‌شده از تمام راه‌حلها مورد نیاز است. در حقیقت اسمال تاک که نمادی واقعی از محیط برنامه‌نویسی شیءگرا است، در ابتدا به‌عنوان ابزار برنامه‌نویسی برای بچه‌ها به‌کار برده شد اما آنچنان وسیع شد که اکنون شامل بیش از ششصد رده است و به‌صورت یکی از توسعه‌یافته‌ترین سیستمها درآمده است.

دربارهٔ دروغی که کسی گفته است فکر کنید. برای تأیید آن دروغ، دروغهای بسیار دیگری نیاز است و ممکن است تمام موجودیت شخص را به‌خطر اندازد. مسلماً اکثر دروغهایی که گفته می‌شود اینقدر بد نیستند، ولی اصل همان است. علم کامپیوتر کار خود را با تصمیمات بد و ناآگاهانه‌ای شروع کرد، جدیدترین این تصمیمهای بد به‌وسیلهٔ آی‌بی‌ام گرفته شد. اگر چه فراوانی کامپیوترهای شخصی پیشرفت علم کامپیوتر را سرعت بخشید به‌طوریکه قابل مقایسه با هیچ مورد قبل از آن نیست، ولی مانع انجام هزاران اندیشه خوب شد.

هنگامی که این تصمیمات در دنیای نرم‌افزار انباشته شدند، سرانجام سلسله مراتبی متزلزل را به‌وجود آوردند که همواره در معرض فروریختن قرار داشت، سپس آنرا *بحران نرم‌افزار* نامیدند.؟؟

۲) نسل بعد از زبان ماشین. هر خط از دستورالعمل در این زبان تقریباً مطابق با یک عمل در زبان ماشین است. اختراع زبانهایی مانند C یا توسعهٔ زبانهایی مانند turbo Pascal، Pascal، این زبان را به‌صورتی زائد درآورد. ولی به‌هرحال هرگاه کارایی بالاترین اولویت را داشته باشد جایگزینی برای آن وجود ندارد.

زمینه

بسیاری از مردم درک متفاوتی از شیءگرایی دارند و غیرمعمول نیست که شاهد بحثهای داغ زیادی بین خبره‌ها در کنفرانسها و بولتنها باشیم. این بحثها بر سر این موضوع هستند که شیءگرایی دقیقاً چه معنایی می‌دهد و یا از چه چیزی تشکیل شده است.

گردی بوچ[۳] می‌گوید[۴]: «آنچه ما درباره‌اش اتفاق نظر داریم این است که مفهوم شیءدر مورد هر چیز شیءگرا، مفهومی اساسی و با اهمیت است.»

بدون اینکه خودمان را درگیر اصطلاحات فنی کنیم و طرفدار جدی استعمال لغات درست باشیم، آنچه که من شیءگرا می‌گویم، این حقیقت است که شیءگرا در واقع انسان‌گرا و یا مسأله‌گرا است. این عقیده بر پایه این حقیقت است که قبلاً انسانها مانند کامپیوتر فکر می‌کردند، اما هم‌اکنون کامپیوترهایی ساخته می‌شوند که در قالب فکری انسان قرار گیرند.

بنابراین چرا ما نیازمند تغییر روشهایی هستیم که در طی چندین دهه کارهای خود را براساس آنها انجام داده‌ایم، آیا یکی از همین روشها که رهیافت کامپیوترگرا نامیده می‌شود امکان قدم گذاشتن بر روی ماه را به‌انسان نداد. مشکل چیست؟ مشکل برمی‌گردد به‌توانایی انسان در غلبه بر پیچیدگی. گردی بوچ در کتابش می‌گوید: «در جهان افراد نابغه کمی وجود دارند. دلیلی وجود ندارد که معتقد باشیم جامعه مهندسی نرم‌افزار دارای سهم غیر معمول زیادی از آنهاست. اگر چه روزنه‌هایی از نبوغ در همهٔ ما وجود دارد، ولی در قلمروی صنعت نرم‌افزار همیشه نمی‌توانیم مطمئن باشیم که معجزه ما را نجات می‌دهد. بنابراین باید روشهای دقیقتر و نظام‌یافته‌تری را برای غلبه بر پیچیدگی در نظر بگیریم.» در حمایت از نظر بوچ آقای بیارنه استروستروپ[۵] دریکی از سخنرانیهای خود در دفاع از زبان C++ می‌گوید: «کن تامپسون[۶] هیچگاه مرتکب خطاهای تایپی نمی‌شود ولی من کن تامپسون نیستم لذا به‌کامپایلری نیاز دارم که خطاهایم را پیدا کند.» صرفنظر از ۵٪ انسانها که از موهبت خداداد درک مسائل پیچیده برخوردارند، بقیهٔ ما باید مسائل را به‌قسمتهای کوچکتری تقسیم کرده و مؤلفه‌های کوچکتر را بفهمیم و تا حد امکان نیز از دیگران کمک بگیریم. مسائل دیگری نیز مانند اصلاح‌پذیری، توسعه‌پذیری، قابلیت تطابق، قابلیت اندازه‌گیری و قابلیت نگهداری وجود دارد. هر قطعهٔ باارزش نرم‌افزار در لحظه‌ای از حیاتش نیاز به‌تغییر و اصلاح دارد. هر چند انتظار داریم که تغییر کوچک نیاز به‌کار کم و تغییر زیاد نیاز به‌کار زیاد داشته باشد ولی خیلی پیش می‌آید که تغییری کوچک نیاز به‌کار زیادی داشته باشد و این مسأله با استفاده از روشهای سنتی حادتر جلوه می‌کند.

3) Grady Booch

۴) نویسنده کتاب *حماسه‌ای* "Object Oriented Design with Applica-tion" است. اگر شما به‌این موضوع علاقمند هستید و اطلاعات زمینه‌ای را دارید من قویاً مطالعهٔ این کتاب را به‌شما توصیه می‌کنم.

۵) خالق C با رده‌ها (class) که بعداً C++ نامیده شد.

۶) همکار در اختراع سیستم عامل Unix و شناخته شده در دنیای شطرنج کامپیوتری

جیمز کوپلین[۷] توضیح می‌دهد که چرا رهیافت شیءگرا رهیافت بهتری است: «الگوی شیءگرایی مانند هر الگوی دیگر به مجموعه‌ای از قواعد ساختمانی و دستورالعملها صورت خارجی می‌دهد. این قواعد فعالیتهای طراحی و برنامه‌سازی را محدود می‌کند اما این محدودیتهاست که طراحی و نگهداری را در درازمدت از خطا مصون می‌دارد. خوشبختانه این الگو یک تکنولوژی پیاده‌سازی به‌نام برنامه‌سازی شیءگرا بوجود آورده است که مستقیماً می‌تواند ساختار تجریدها را در مرحلهٔ پیاده‌سازی، تحقّق بخشد.»

روشهای بسیاری برای طراحی، تحلیل و پیاده‌سازی ابداع شده است که همه دارای یک هدف مشترک هستند و آن ساده‌کردن مرحلهٔ درک مسئله، دستیابی به بهترین راه حل ممکن و زیباترین راه حل است. طبیعتاً مانند بسیاری موارد دیگر انسانها متفاوت هستند و درک آنها از اینکه مسأله چه هست و راه حل چیست، مختلف است اما قدر مسلم این است که باید خیلی چیزها را آموخت و برای اینکار باید کمک گرفت.

روشهای شیءگرا سعی دارند که با رهیافت متفاوتی با مسأله روبرو شوند. وقتی از یک قلم استفاده می‌کنید می‌توانید با نگاه کردن به‌آن به‌تمام سئوالات دربارهٔ قلم پاسخ دهید. سئوالاتی مانند چه رنگی است، چقدر تیز است، نوک آن چقدر کند است؟ متأسفانه در دنیای نرم‌افزار اکثر مواقع اینگونه نیست و جدا از پیچیدگی طبیعی مسأله، یک نوع پیچیدگی اضافه نیز توسط تحلیل‌گران، طراحان و مجریان بوجود می‌آید.

اجازه بدهید ببینیم روشهای ممکن کدام هستند. از زمان بوجود آمدن اولین زبان سطح بالا و پیدایش توابع و رویه‌ها، قابل قبولترین روش طراحی روش از بالا به‌پائین بود، یعنی مسأله را به‌مسائل کوچکتری تقسیم کرده و به‌این کار آنقدر ادامه می‌دهیم تا در پایان به‌ساده‌ترین مسائل برسیم. برای مثال، برای طراحی یک تیم فوتبال با

[یک تیم فوتبال]

شروع کرده و سپس می‌گوئیم یک تیم فوتبال تشکیل شده است از

[یک خط دفاعی
یک خط میانی
یک خط حمله]

سپس خط دفاعی را به‌صورت زیر تقسیم می‌کنیم

[یک دروازه‌بان
مدافعان]

مدافعان خود می‌توانند به‌شکل زیر تقسیم شوند

[مدافع چپ
مدافع راست
مدافعان میانی]

و به‌همین ترتیب ادامه می‌یابد.

روشن است که در هر بار تقسیم جزئیات بیشتری وارد مسأله می‌شود و به‌هدف خود نزدیکتر می‌شویم. شخصی با آگاهی از یک زبان برنامه‌سازی قطعاً متوجه می‌شود که هر تجزیه مطابق با یک رویه و یا یک تابع است. بنابراین مسألهٔ بزرگتر نیازمند تجزیه‌های بیشتر است و این امر خود مستلزم تعداد بیشتر رویه‌هاست.

شیوهٔ شناخته شدهٔ دیگری که به‌وسیلهٔ طراحان پایگاه داده‌ها به‌وجود آمد، طراحی از روی داده‌ها نامیده می‌شود. در این روش تأکید اصلی بر نگاشت داده‌های ورودی و خروجی است.

مشکل روش از بالا به‌پائین این است که همیشه نمی‌توان با این دید با مسأله برخورد کرد و بعضی اوقات روشهای از پائین به‌بالا و یا از وسط لازم هستند. مشکل دیگر روش از بالا به‌پائین این است که مسأله ممکن است به‌طور غیرضروری بزرگ شود و در نتیجه بسیار پیچیده می‌شود.

مشکل روش طراحی از روی داده‌ها این است که در مورد بسیاری از مسائل مناسب نیست. برای مثال می‌توان از مسائلی یاد کرد که زمان در آنها نقش حیاتی دارد. رالف هاجسون در مقاله‌اش برای کنفرانس اروپایی C++ و C جزئیات بیشتری را ذکر کرده است.

عناصر شیءگرایی

خوب، ما دربارهٔ این حقیقت که روشهایی که بطورگسترده به‌کار می‌روند برای کاربردهای امروزی کافی نیستند، توافق کردیم. اما راه حل چیست؟

در دنیای شیءگرایی دو نقطهٔ مهم برای شروع وجود دارد. اول اینکه ما نیازی به‌ترجمهٔ راه حلهایمان برای اینکه درکامپیوتر اجرا شوند، نداریم ـ نکتهٔ مهم اینجاست که راه حلها بر پایهٔ ایده‌های جهان واقعی هستند و طبیعتاً قابل درکترند. همچنین به‌دلیل اینکه نیازی به‌ترجمه نداریم، برخی از پیچیدگیهای مصنوعی از بین می‌رود. دوم اینکه همه چیز بر پایه شیءقرار دارد و کوچکترین واحد، در مقابل تابع یا رویه، شیءاست.

فهمیدن این نکته نیاز به نبوغ خاصی ندارد که شیءیک واحد خود کفاست و به‌سادگی قابل فهم است و می‌توان از آن بارها استفاده کرد.

پنج مفهوم حیاتی در دنیای شیءگرایی وجود دارد که بدون آنها بهتر است بگوئیم دنیای شیءگرایی وجود خارجی ندارد.

- مدل‌سازی با اشیاء
- بسته‌بندی و این همانی شیء
- پیامها یا فراخوانی عملیات
- رده‌ها و وراثت
- چندریختی

این مفاهیم پایه و اساس پنج عنصر هستند: شیء، تجرید، بسته‌بندی، سلسله مراتب، واحدمندی.

۷) محقق آزمایشگاههای AT & T و نویسندهٔ کتاب Advanced C++ with Idioms

شیءچیست؟[8]

اکنون زمانی است که می‌توانیم با بعضی از ایده‌ها از دید علم کامپیوتر آشنا شویم. هر چیز در این جهان در هر زمانی دارای حالتی است و کارهایی را می‌توان روی آن حالت انجام داد که ممکن است آن حالت را تغییر دهد. برای مثال یک تلویزیون ممکن است روشن یا خاموش و بدون صدا یا بدون رنگ باشد. مجموعهٔ این صفات یک حالت منحصر بفرد را تشکیل می‌دهد و عملیاتی وجود دارد که می‌تواند هر کدام و یا تمام این صفات را تغییر دهد، مانند زیاد کردن صدا که طبیعتاً روی صفت صدا تأثیر می‌گذارد، عوض کردن کانال، خاموش کردن تلویزیون که می‌تواند تمام صفات را تغییر دهد و بالاخره اینکه فشردن دکمهٔ ساعت، زمان فعلی را بدون تغییر هیچ صفتی نشان می‌دهد[9]. این مثال را می‌توان به‌انسان، حیوانات، ماشینها، یک صفحهٔ کاغذ و غیره تعمیم داد. دانشمندان کامپیوتر این حالات را مشاهده کرده‌اند و ایده‌ای تقریباً مشابه زا مطرح کرده‌اند. بنابراین، شیءمجموعه‌ای از صفات و عملیات قابل اعمال است. این ایدهٔ به‌ظاهر ساده تمام علم کامپیوتر، مخصوصاً مهندسی نرم‌افزار را متحول کرده است. تفکر دربارهٔ شیءها در عوض الگوریتم، ساختمان داده‌ها یا پردازنده‌ها چنان جذاب است که در چند سال گذشته مجموعه‌ای از چیزهای شیءگرا مانند کاربردهای شیءگرا، زبان، طراحی و تجزیه و تحلیل شیءگرا پدیدار شده‌اند.

خوب، ممکن است بگوئید این که خیلی ساده است، پس هر چیزی یک شیءاست. هیچ چیز اینقدر ساده نیست، در مهندسی نرم‌افزار بسیاری از عناصر و محدودیتها بر نتیجهٔ یک تصمیم تأثیر می‌گذارند و برای من یا شما غیرممکن است که بدون داشتن اطلاعات لازم، آنها را به‌دو گروه صحیح و اشتباه تقسیم کنیم. یک مثال خوب رنگ است، اگر شما به‌عنوان شیءماشینی داشته باشید آیا رنگ آن ماشین یک صفت است یا خودش شیءدیگری است که جزئی از ماشین است[10]. یا برای اینکه واقعاً گیج شوید دربارهٔ عبارت «می‌تواند پرواز کند» چه فکر می‌کنید، آیا یک صفت است یا یک عمل؟ بله، اگر من تعدادی صفت دیگر مانند «بال دارد» داشته باشم، آنگاه «می‌تواند پرواز کند» می‌تواند یک عمل برای آزمایش این صفت و نوع شیءباشد. اگر بالی پیدا نشود و یا اگر شیءیک شترمرغ باشد، نتیجه می‌تواند اشتباه باشد.

تجرید چیست؟

تعریف تجرید، به‌وجود آوردن مفهومی شخصی از یک ایدهٔ اساسی است. ما در طی سالها، توانایی مجرد کردن «چیزها» را به‌کار برده‌ایم. برای مثال بازیکن شطرنج صفحهٔ بازی را به‌صورت میدان نبرد تصور می‌کند، شخصی معمولی امکان دارد صفحهٔ شطرنج را یک قطعهٔ چوبی شطرنجی تصور کند و یک برنامه‌ساز کامپیوتر آنرا به‌صورت مجموعه‌ای از ساختمان داده‌ها می‌بیند. بله، افراد مختلف تجریدهای متفاوتی به‌کار می‌برند. نکتهٔ دیگر در به‌کار بردن تجرید رهانیدن خودمان از جزئیات خاص و عملی یک ایده و درنظر گرفتن آن به‌عنوان یک کل است. طراح ماشین، موتور را به‌عنوان تجریدی درنظر می‌گیرد که قرار است کار خاصی را انجام دهد. اینکه موتور چگونه این‌کار را انجام می‌دهد، به‌طراح ارتباطی ندارد. تکنیک تجرید در طی سالها، در نمونه‌های دیگری نیز به‌کار رفته است اما هیچگاه یک عنصر کامل و مجتمع نبوده است. الگوی شیءگرایی نه تنها به‌کار بردن این تکنیک را آسان می‌کند بلکه آنرا به‌سختی و با شدت به‌کار می‌برد و اجرا می‌کند. شیءها تجریدهایی هستند از موجودیتهای قلمرو مسأله.

بسته‌بندی چیست؟

بسته‌بندی یعنی به‌کلّی احاطه کردن، تماماً شامل بودن، در درون نگه داشتن. گردی بوچ می‌گوید: «تجرید یک شیءباید به‌تصمیمات مربوط به‌اجرای آن مقدّم باشد، وقتی روش اجرا انتخاب شد باید به‌عنوان راز تجرید درنظر گرفته شود و از اکثر کاربران پنهان باشد.» در دید شیءگرایی، می‌توانیم آشکال بسیاری از شیءرا ببینیم، هر کدام با رفتاری خاص، اما آنچه که نمی‌دانیم ساختار داخلی آن شیءهاست. برای مثال شبکه تلفن به‌همراه خدماتی مثل شماره گرفتن، زنگ زدن، صدور صورتحساب، واحدبندی مکالمه و... فراهم می‌شود. آنچه که مورد علاقه ما نیست و حتی نباید بدانیم، روشی است که شرکتهای مخابراتی این خدمات را انجام می‌دهند. ما نمی‌دانیم که آنها از ماهواره استفاده می‌کنند و یا سیمهای زیرزمین را به‌کار می‌برند. بسته‌بندی به‌این شرکتها اجازه می‌دهد بدون اینکه ما متوجه شویم، ساختار درونی‌اش را تغییر دهد. حالا اگر استفادهٔ ما از خدمات وابسته به‌چگونگی اجرای آن توسط شرکت مخابراتی بود، هنگامی که سیستمهای جدید نصب می‌شد قادر به‌استفاده از خدمات نبودیم.

فرستادن پیام چیست؟

اگر بخواهم برای تعطیلات بلیط بخرم به‌آژانس مسافرتی محل خود رفته و آن را تهیه می‌کنم. در این مرحله من به‌آنچه که بعداً اتفاق می‌افتد توجه ندارم، در حقیقت عمل من باعث ارسال پیامهایی از آژانس مسافرتی به‌مسئول تور، از مسئول تور به‌خط هوایی و هتل می‌شود. خط هوایی خود باید پیامهایی برای خلبانان و مهمانداران بفرستد و به‌همین ترتیب ... تعجب‌آور است که عمل بلیط خریدن چگونه باعث بروز عملیات زیاد دیگری شد. نکتهٔ مهم دیگری که در اینمورد مطرح است این است که هیچکس نیازی ندارد که بداند کارها در قسمتهای دیگر شبکه چگونه انجام می‌شود. آژانس مسافرتی نمی‌داند کدام خط هوایی برای شما به‌وسیلهٔ مسئول تور درنظر گرفته شده

---

۸) برای توضیح تکنیکی بیشتر به‌کتاب Object Oriented Design with Application صفحهٔ ۳۴ مراجعه شود.

۹) عموماً اگر عملیاتی یک یا بیشتر از صفات را تغییر دهد به آن modifier می‌گویند و اگر فقط دربارهٔ صفت یا صفاتی سئوال کند ـ بدون تغییر ـ آن را interrogator می‌نامند.

۱۰) این یک رابطهٔ شناخته شده بین شیءهاست که HAS A نام دارد. مانند رابطهٔ اتومبیل و موتورآن.

است و مسئول تور نیز نمی‌داند چه کسی مسئول هتل است. این مثال ساده ولی روشن، اصل ماهیت رهیافت طبیعی و یا رهیافت شیءگرا را نشان می‌دهد.

داستان فوق تقریباً شامل تمام ایده‌های سحرآمیزی است که به‌تکنیکهای شیءگرایی مربوط می‌کنیم ولی ممکن است بگوئید این مثال یک موضوع روزمره است و هیچ‌چیز واقعاً علمی در آن ذکر نشده است. دقیقاً همینطور است!!!

بزرگترین بدعت در روشهای شیءگرایی این است که هیچ بدعتی در این روش وجود ندارد! این روش فقط شیوه‌ای را نشان می‌دهد که انسان در طی هزاران سال به‌وسیلهٔ آن فکر کرده و کارهایش را انجام داده است.

مدل‌سازی شیء

در داستان من، می‌توانید شیءهای زیادی را انتخاب کنید مثل خودم، آژانس مسافرتی، هواپیما و هتل. از تجربه‌ام با برنامه‌سازی ساختیافته به‌روش جکسون[11] فهمیدم که شروع کار بسیار مشکل است. برای اولین بار که ایده‌ای داشتم همه‌چیز به‌هم ریخت. این تجربه دوباره وقتی که مشغول طراحی یک برنامهٔ شطرنج شیءگرا بودم اتفاق افتاد. یافتن شیءها کار دقیق و ظریفی است اما وقتی فهمیدید کدام شیءهست و کدام نیست، دیگر مسأله‌ای ندارید.

وراثت چیست؟

حتماً زیاد شنیده‌اید که بچه‌ای شبیه والدینش است و یا عبارتی مثل «عضوی از خانواده» یا «مثل پدر و پسر». ما همیشه دربارهٔ نسل قدیمی، نسل جدید و نسل متفاوت صحبت می‌کنیم. آنچه واقعاً می‌خواهیم بگوئیم این حقیقت است که بعضی از رفتارهای ما شبیه بعضی از نسلهای قبلی و بعضی تغییر کرده است. پدرم بلندقد و تاس بود پس من بلندقد و بی‌مو هستم. پدرم خیلی چای می‌خورد و من آب میوه می‌خورم. وراثت بعضی از صفات و رفتار را از اجدادمان به‌ما انتقال می‌دهد که ما بعضی از آنها را تغییر داده و بعضی دیگر را بدون تغییر به‌فرزندانمان منتقل می‌کنیم.

در این الگو، شما می‌توانید ببینید که تمام عناصری که در برگهای درخت قرار دارند بعضی صفات و رفتارهای مشترک با گره‌های درخت دارند[12].

چندریختی چیست؟

poly یعنی چندتا و morphe یعنی ریخت و شکل. توانایی سروکار داشتن با اشکال زیاد، چندریختی است. یک عمل خاص بسته به‌اینکه روی چه شیئی انجام شود نتایج متفاوتی دارد.

شیءگرایی در عمل

اکنون موقع آن است که از آموخته‌های خود استفاده کرده و به‌سراغ شیءگرایی برویم. اول، باید وارسی کنیم که طرح شیءگرا باشد. یعنی، باید مطابق شرایط ذکر شدهٔ ما باشد.

مدل‌سازی با شیء. تمام موجودیتها در این طرح شیءهستند مانند ویراستار، مجله و...

بسته‌بندی و این همانی شیء. نمی‌دانیم نویسنده مطالبش را چگونه می‌نویسد، از چه کامپیوتری استفاده می‌شود، و یا اینکه آیا نویسنده مرد یا زن است، آسیایی است یا اروپایی.

پیام یا فراخوانی عملیات. اعمالی مانند ارسال، خواندن و منتشرکردن همه پیام هستند.

رده‌ها و وراثت. با مراجعه به‌طرح وراثت خودمان، مجله خود نسلی از ادبیات است و نویسنده و ویراستار نسلی از انسان هستند. مجله و نویسنده رده‌ها هستند و شیءها مواردی از این رده‌ها هستند برای مثال من موردی از ردهٔ نویسنده هستم.

چندریختی. نویسنده ممکن است با فرستادن مطالبش به‌ویراستار به «ارسال» پیام پاسخ دهد. ویراستار ممکن است با فرستادن مطالب به‌اتاق چاپ به «ارسال» پیام پاسخ دهد. یک پیام نتایج متفاوت بر روی شیءهای متفاوت دارد.

تکرار نکات مفید

گفتیم که شیءگرایی خوب است زیرا همانطور که به‌ما در مقابل پیچیدگی کمک می‌کند، مواردی نظیر تغییرپذیری، توسعه‌پذیری، تطابق‌پذیری، قابلیت اندازه‌گیری و قابلیت نگهداری را تسهیل می‌کند.

سروکار داشتن با پیچیدگی، مهمترین جنبهٔ مهندسی نرم‌افزار است. مدل‌سازی بوسیلهٔ شیءها، برای انسان مستقیمترین روش کار کردن با چیزهاست. همانطور که ممکن است تا به‌حال متوجه شده باشید، مطلب غیرطبیعی در این مورد وجود ندارد. در مقابل ما می‌توانستیم روش متفاوتی مانند طراحی از بالا به‌پائین به‌کار بریم.

---

۱۱) روشی برای طراحی که به‌وسیلهٔ مایکل جکسون به‌وجود آمد (نه آن خوانندهٔ معروف!) این روش بر پایهٔ سه نوع عملیاتِ انتخاب، تکرار و دنباله است.

۱۲) در علم کامپیوتر درختها همیشه وارونه هستند. پیوندهای داخلی، گره (node) نامیده می‌شود در حالیکه گره‌هایی که انشعابی ندارند موسوم به‌برگها (leaves) هستند.

[مقاله]
[نوشتن
تصحیح
انتشار
خواندن
]
[
نوشتن
تحقیق
تایپ کردن جملات در کامپیوتر
کنترل املائی
فرستادن آن برای ویراستار
تصحیح
انتشار
خواندن]

اشتباه نکنید، این فعالیتها در مراحلی از توسعه و پیشرفت مورد نیاز هستند ولی بخاطر طولانی بودن باید آنها را تا مراحل بعدی به‌تأخیر انداخت.

تغییرپذیری. من می‌توانم ویراستار را بردارم و به‌جای آن کمک‌ویراستار را قرار دهم، بدون اینکه طراحی خراب شود. در حقیقت برنامه‌سازی شیءگرا حیات خود را بصورت یک زبان نمونه‌سازی[۱۳] شروع کرد.

۱۳) نمونه‌سازی، فعالیت ساختن سریع سیستمی است که بتواند احساس و رفتار سیستم پایانی را تقلید کند.

توسعه‌پذیری. من می‌توانم مسئولیتهای بیشتری را به‌نویسنده محول کنم. برای مثال مطلب نوشته شده باید جزوه شود.

تطابق‌پذیری. من می‌توانم ویراستار را در پروژهٔ دیگری که انتشار یک کتاب است به‌کار برم، زیرا ویراستار من می‌داند چگونه قسمتی از ادبیات را بخواند و آن را نقد کند. نباید مشکلی برای دوباره به‌کار بردن او وجود داشته باشد.

قابلیت نگهداری. این ویژگی، دومین هدف عمدهٔ مهندسی نرم‌افزار است. حالا، اگر نویسنده دوست دارد در عوض استفاده از کامپیوتر و واژه‌پرداز، قلم و کاغذ بردارد یا اگر نویسنده موفق بوده و یک منشی استخدام کرده، می‌تواند از منشی خود بخواهد که مطالب را برایش تایپ کند. تمام این تغییرات با توجه به‌طبیعت تکنولوژی شیءگرا با حداقل کوشش انجام می‌شود.

در این مرحله، شما ایدهٔ خوب و روشنی از مفهوم بعضی مطالب نامفهوم و مرموز دارید. برای مثال تجزیه و تحلیل شیءگرا روشی است که سعی می‌کند شیءها و ارتباطات آنها را در دامنهٔ مسأله تشخیص دهد، طراحی شیءگرا روشی است که راه‌حل را روی شیءها مدل‌سازی می‌کند، سلسله مراتبی از رده‌ها ایجاد کرده و مسئولیت آنها را در طرح کلی تعریف می‌کند. برنامه‌سازی شیءگرا مجموعه‌ای از زبانهایی است که قرار است پیاده‌سازی مدل‌سازی، بسته‌بندی، وراثت، چندریختی و تبادل پیام شیءرا تسهیل کند. برنامه کاربردی شیءگرا، برنامه‌ای است که به‌وسیلهٔ عناصر شیءگرا طراحی و نوشته شده است.

امیدوارم از این مقاله لذت برده باشید و مهمتر اینکه دید بهتری از دنیای شیءگرا به‌دست آورده باشید.

●

نبرد شیءها*

زبان C++ نوع خاصی از تفکر دوگانه را در صنعت نرم‌افزار پدید آورد. ویژگیهای شیءگرای C++ نشان‌دهندهٔ اینْ مطلب هستند که چرا این زبان تدریجاً اما با اطمینان زبان C را تحت‌الشعاع خود قرار می‌دهد. با وجود این روشن است که C++ یکی از اصول طراحی شیءگرا را برآورده نمی‌کند و آن باز به‌کارگیری نرم‌افزار[1] است.

در مورد مفاهیم بسته‌بندی[2] و توزیع ماجولهای دودوئی، زبان C++ در واقع گام بزرگی به عقب برداشته است. کتابخانه‌هایی که به‌طور ایستا بخشی از زبان C به‌شمار می‌روند همواره راه مؤثری برای معاوضهٔ کدهای قابل استفادهٔ مجدد بوده‌اند. کتابخانه‌های پویایی که به زبان C پیوند شده‌اند حتی بهتر کار می‌کنند. جدایی از وجود پیوند ایستا، سیستم‌عاملهایی که بر مبنای DLL هستند، مانند ویندوز و یا اُاِس۲ به‌صورت مجموعه‌ای از قسمتهای قابل بهنگام‌سازی درآمده‌اند.

هنگامی که کتابخانه رده‌های C++ دچار تغییر می‌شوند، کاربران نیز باید دوباره آنها را کامپایل کرده و با تغییرات جدید تطابق دهند. تام پنلو نایب رئیس شرکت متاور می‌گوید: «تولیدکنندگان سیستم‌عاملهایی که با C++ کار می‌کنند باید برای مدیران خود توضیح دهند که چرا نمی‌توانند کتابخانه‌های شیءگرا عرضه کنند. البته پاسخ تولیدکنندگان همیشه چنین است: «منظور شما چیست؟ ما در طی سالها همواره با کتابخانه‌های رویه‌ای[3] کار کرده‌ایم.»

در این قسمت یک مسأله مهم دیگر نیز مطرح است و آن سازگاری پیوندی در بین کامپایلرها می‌باشد. به گفتهٔ یکی از اولین فروشندگان کتابخانه‌های C++: «این جایی است که امنیت نوع[4] در زبان C++ مشکل ایجاد می‌کند.»

اگر چه ما با انعطاف‌پذیری برنامه‌سازی شیءگرا را معادل با استفادهٔ مجدد در نظر می‌گیریم، جیم بونین معاون سابق بخش مهندسی شرکت استپ استون که اکنون یکی از مشاوران آن شرکت است می‌گوید: «زبان C++ هیچگاه به‌عنوان راه‌حل مسألهٔ تعویض مؤلفه‌های نرم‌افزاری در مقیاسهای بزرگ در نظر گرفته نشده است. او می‌گوید: «وقتی ما بیارنه را در این موضوع تحت فشار قرار دادیم، (اشاره به بحثی است که در سال ۱۹۸۷ بین بیارنه استروستروپ از شرکت تلفن و تلگراف امریکا و مبدع زبان C++ و برَد کاکس از شرکت استپ استون و ابداع‌کننده زبان objective-C درگرفت.) او پذیرفت که موضوع استفادهٔ مجدد از ماجولهای C++ احتمالاً در سطح پروژه و یا اداره است.»

objective-C از شرکت استپ استون و SOM[5] متعلق به شرکت آی‌بی‌ام به‌عنوان راه‌حلهایی برای رفع این مشکل ارائه شدند. هر یک از این تکنولوژیها از یک هستهٔ در حال اجرا برای مقیدسازی پویای شیءها استفاده کردند و روند وراثت را نیز در محدودهٔ شیءها حفظ کردند. پنلو می‌گوید: «با SOM می‌توانید توابع مجازی را اضافه کنید و یا حتی سلسله مراتب گروه را دوباره دسته‌بندی نمائید.»

در حال حاضر ویژگی Direct to SOM که پنلو به high/C++ محصول شرکت متاور اضافه می‌کند الگوی شیءگرایی داخلی کامپایلر را با SOM محصول آی‌بی‌ام همسو می‌کند و در نتیجه مسأله تعویض مؤلفه‌ها و سازگاری پیوندها حل خواهد شد. پنلو می‌گوید مزایای این شیوه بیش از جبران هزینه، درکارایی است. کلیف ریوز مسئول تولیدات با تکنولوژی شیءگرا در شرکت آی‌بی‌ام می‌گوید: «هنگامی که همان ویژگی به کامپایلر C++ متعلق به آی‌بی‌ام افزوده شود، برای نخستین بار شاهد تبادل برنامه‌های دودویی بین کامپایلرهای مختلف C++ خواهید بود.»

بونین عقیده دارد که SOM برای به‌کار بردن مؤلفه‌های نوشته شده به زبانهای مختلف می‌تواند به‌کار رود اما سؤالاتی دربارهٔ سازگاری با گونه‌های قبلی C++ در این روش وجود دارد. او می‌گوید مشخص نیست که C++ تا چه حدی می‌تواند اضافات را تحمل کند. مایک پوتل معاون توسعهٔ تکنولوژی در شرکت تلیجنت می‌گوید: «کامپایلر Direct to SOM از عهدهٔ پیچیدگی زیاد C++ بر نخواهد آمد.» تلیجنت قسمتهایی را به کامپایلر C++ تولید خودش به منظور تأمین توانایی مقیدسازی پویا که برای سیستم‌عاملهای شیءگرا لازم است، افزوده است. (این سیستم‌عاملها در دست تولید هستند)

تعجبی ندارد اگر تندترین انتقاد از SOM و DSOM (Distributed SOM) (محصولات IBM) از طرف شرکت مایکروسافت باشد. مارک ریلاند می‌گوید: «ما به سختی پایبند این عقیده هستیم که واسطها[6] ویژگیهای نرم‌افزار هستند و جدا از اجراها می‌باشند. هیچیک از طرحهای CORBA[7] شامل SOM با این حقیقت مواجه نشده‌اند که داشتن میلیونها شیءدودویی واقعاً چه مفهومی دارد.»

او می‌گوید SOM در مواردی که فروشندگان اجراهای مختلف از یک واسط واحد را ارائه کنند، شکست می‌خورد. اجراهایی که در ابتدا معادل هستند ولی در طول زمان، از هم دور می‌شوند. او می‌گوید COM[8] متعلق به

(* Object Wars, BYTE, May 1994

1) Software Reuse 2) Packaging 3) Procedural 4) Type 5) System Oriented Model 6) interfaces 7) Common Object Request Broker Architecture 8) Common Object Model

مایکروسافت از این مسائل با استفاده از واسطهای زاینده اجتناب می‌کند به این صورت که یک شیءجداگانه می‌تواند به‌طور همزمان چندین نسخه از خود را تعریف کرده و مجموعهٔ تواناییهای هر نسخه را تغییر دهد.

جد هریس مدیر اجرایی آزمایشگاههای Cilabs با حرارت پاسخ می‌دهد: «این یک مثال ناقص است. امکان ندارد هر دو اجرا از یک واسط معتبر باقی بمانند. حتماً در یکی از آنها خطایی وجود دارد که می‌توانید آنرا به‌طور مکانیکی مشخص کنید.» نه SOM و نه CORBA نیاز به درخت وراثت تک‌ریشه‌ای ندارند و کاربران می‌توانند ترکیبی از وراثت چندگانه را از میان مجموعهٔ مؤلفه‌ها به‌کار برند.

مارک برم هال مدیر فنی خدمات محاسباتی توزیع شده در شرکت دیجیتال می‌گوید: «اگر چه این عمل با SOM امکانپذیر است ولی در مقایسه با COM از کارایی کمتری برخوردار است.» (DEC محصول COM را تأئید کرده است با آ این هدف که مجموعه ابزار تولید خود موسوم به object broker را به‌عنوان رابطی بین تکنولوژیهای OLE و CORBA معرفی کند.)

او می‌گوید: «در حالت توزیع شده و با به‌کارگیری DSOM هزاران شیءاز راه دور به معنای هزاران نماینده هستند ولی با COM می‌توانید این شیءها را به تعداد قابل شمارشی کاهش دهید. این کاهش به‌وسیلهٔ تبدیل آنها به مجموعهٔ کوچکتری از واسطها امکانپذیر است. در این تبدیل اندازه‌گیریها غیرخطی است. بنابراین می‌توانید فقط با دهها و یا صدها نماینده کار کنید و از غیر قابل کنترل شدن حجم محیط جلوگیری کنید.

اگر چه بحثها و صحبتها ادامه دارد بدون آنکه نشانی از پایان آنها باشد، یک اتفاق نظر ضمنی نیز به‌وجود آمده است: مؤلفه‌ها موضوع مهمی هستند و C++نمی‌تواند به تنهایی از عهدهٔ کارکردن با آنها برآید و مکانیسمهای جدیدتری برای پیشرفت نیاز است.

ترجمهٔ سعید زین‌العابدین

لیست قیمت TEX-پارسی در Unix/Xenix

تعداد کاربر	قیمت
یک	۱,۷۵۰,۰۰۰ ریال
دو	۲,۲۵۰,۰۰۰ ریال
چهار	۲,۷۵۰,۰۰۰ ریال
هشت	۳,۵۰۰,۰۰۰ ریال
شانزده	۴,۵۰۰,۰۰۰ ریال
نامحدود	۶,۰۰۰,۰۰۰ ریال

TEX پارسی

قویترین نشرافزار فارسی

با واسط کاربر گزینشی (Menu-driven)

با توانایی به‌کارگیری

- انواع صفحه‌های نمایش
- انواع پایانه‌های متن و گرافیک
- انواع چاپگرهای لیزری و سوزنی

قابل اجرا در سیستم‌های عامل

- UNIX
- XENIX
- MSDOS

قیمت ویژه MSDOS
۶۴۰,۰۰۰ ریال

حداقل شرایط مورد نیاز کامپیوتر ۳۸۶ با ۴ مگابایت حافظهٔ اصلی و ۸۰ مگابایت دیسک سخت.

شرکت داده‌گاوی ایران

شماره ۴۱۷، باغ صبا، خیابان شریعتی، تهران

تلفن: ۷۶۰۰۱۴، ۷۵۰۱۴۵۶ فاکس: ۷۶۶۳۵۶

مدلی برای توسعهٔ شیءگرا

علی ارسنجانی*

email: arsanjani@irearn.bitnet

مقدمه

یکی از اولین پیشگامان و پیشکسوتان روش شیءگرا، آقای رِنش در سال ۱۹۸۲ چنین می‌نویسد:

«در دههٔ هشتاد (و نود) برنامه‌نویسی شیءگرا همان اهمیتی را کسب خواهد کرد که برنامه‌نویسی ساختیافته در دههٔ هفتاد داشته است. همه طرفدار آن خواهند بود. تولیدکنندگان نرم‌افزار محصولات خود را به‌عنوان محصولاتی که از شیءگرائی پشتیبانی می‌کند و بر پایهٔ آن طراحی شده است، تبلیغ خواهند نمود. مدیران، همگی دم از رعایت آن خواهند زد و به آن بها خواهند داد. برنامه‌نویسان همگی ادعای به‌کارگیری آنرا خواهند داشت (هر یک به طریق متفاوت خودش). حال آنکه، آنها واقعاً نمی‌دانند که شیءگرائی چیست.» [Rentsch 82].

امروزه، تب شیءگرائی، اذهان دنیای انفورماتیک را چنان به‌خود مشغول ساخته است که گویی کلمه شیءگرائی، در فرهنگ اصطلاحات مهندسی نرم‌افزار با عبارت «طراحی و پیاده‌سازی خوب» مترادف شده است. حداقل ده سال است [Booch 84] که این فلسفه و نگرش، به‌طور جدی مطرح بوده است، ولی چرا اکنون چنین محبوبیت زیادی پیدا کرده است؟

بیشک، یکی از دلایل عمده این محبوبیت، ورود کامپایلرها و محیطهای پرقدرت زبانهای شبه شیءگرا مانند C++ به بازار و نتیجتاً قابل دسترس بودن امکانات توسعهٔ شیءگرا برای تعداد کثیری از برنامه‌نویسان بوده است. علت دوم، این است که سیستمهای عامل (از آن جمله ویندوز اِن‌تی، ویندوز ۳/۱ و تلی‌جنت)، ابزار، محیطها و محصولات جدّی برنامه‌سازی نسبتاً جدید، دارای طراحی و پیاده‌سازی مبتنی بر متدولوژی شیءگرائی می‌باشند. سوم این‌که شرکتهای بزرگی مانند آی‌بی‌ام، بورلند، مایکروسافت و هیولت پاکارد پروژه‌های وسیعی در جهت توسعه و پشتیبانی از نگرش[1] شیءگرا در دست اجرا دارند [Poulin 93]. این شرکتها، همچنین از طریق تبلیغات و ارائه محصولاتی مبتنی بر نگرش شیءگرا، موضع خود را در قبال اهمیت و توانایی بالقوهٔ آن مشخص نموده و با اعلام انجام پروژه‌های طویل‌المدت با بودجه‌های سنگین در این زمینه [Griss 93]، اذهان دنیای کامپیوتر را بیشتر و بیشتر به این نگرش معطوف داشته‌اند.

یکی از اهداف اصلی این نوشتار، روشن نمودن این نکته است که فرارونده توسعه نرم‌افزار به روش شیءگرا، مزایای قابل توجهی دارد و برای بهره‌برداری از این منافع بالقوه، باید از یک مدل توسعه نرم‌افزار استفاده کرد که با این نگرش همسویی و انطباق کامل داشته باشد و اجازه بهره‌برداری از تمام امکانات این روش را در اختیار مهندسین نرم‌افزار قرار دهد. در حال حاضر، پروژه‌هایی که از متدولوژی شیءگرا استفاده می‌کنند، اغلب از مدلهای سنتی، مانند مدل آبشاری برای توسعه سیستم نرم‌افزاری استفاده می‌کنند. توسعه‌گران سیستمها به ظاهر به این امر کم توجه هستند که برای یک متدولوژی مانند شیءگرائی که دارای ویژگیها و امکانات خاصی می‌باشد، نمی‌توان از چهارچوب یک مدل توسعهٔ غیر متجانس با اسلوبها و مفاهیم اولیهٔ آن متدولوژی استفاده نمود و در عین حال، انتظار بهره‌برداری کامل و موفقی از تمام مزایای آن روش داشت. ویژگی و مزیتهای شیءگرائی، زمانی تجلی و جلوه خواهد نمود که مدل توسعه نیز شیءگرا باشد. به دلیل عدم تأکید تئوریسینها و مهندسین نرم‌افزار بر این نکته، دنیایی از امکانات، بلااستفاده باقی مانده است. ذیلاً مفاهیم و نکاتی در جهت برداشتن گامهایی برای نیل به یک مدل شیءگرا پیشنهاد خواهیم نمود. برای ورود به بحث پیرامون مدل، ابتدا، مفاهیم مدل و متدولوژی را بررسی کرده، و پس از گذری اجمالی بر مبادی، اصول و متدولوژیهای شیءگرائی، در انتها مزایا و معایب این نگرش را به‌طوری که در مهندسی نرم‌افزار مطرح است، مرور خواهیم کرد.

مدلها و متدولوژیهای توسعه نرم‌افزار

ابتدا باید تفاوت میان مدل و متدولوژی توسعهٔ نرم‌افزار را بررسی نمود [Boehm 87]. یک مدل توسعه نرم‌افزار[2] باید شامل اجزای زیر باشد:

*) آقای ارسنجانی دارای مدرک کارشناسی ارشد مهندسی نرم‌افزار، گرایش سیستمهای اطلاعاتی مدیریت می‌باشند. ایشان حدود پنج سال و نیم است که در زمینه آنالیز، طراحی و برنامه‌نویسی شیءگرائی مشغول فعالیت بوده و در سال گذشته در کشور انگلستان مشغول انجام پروژه‌های شیءگرا (با زبان و بانک اطلاعاتی شیءگرا در محیط SUN) بوده‌اند و مدیریت فنی تیم خود را برعهده داشته‌اند. علایق کاری ایشان شامل زبانها و بانکهای اطلاعاتی شیءگرا و مدلها و متدولوژیهای توسعه سیستمهای نرم‌افزاری بزرگ می‌باشد.

1) Paradigm

2) Software Development Model

۱ـ **ترتیب مراحل:** قدمها یا مراحل متوالی که باید برای نیل به محصول یکی پس از دیگری پشت سر گذاشت. این بخش از مدل را می‌توان به‌عنوان پاسخی به این سؤال در نظر گرفت که: «پس از اتمام این مرحله، مرحلهٔ بعدی کدام است؟»

۲ـ **معیارهای ورود به مرحلهٔ بعد**[3]: «با تحقق چه شرایطی باید به مرحلهٔ بعدی برویم؟» متدولوژی، با مدل همان تفاوتی را دارد که جزء نسبت به کل دارد. یک متدولوژی در داخل هر یک از مراحل یک مدل نقش خود را ایفا می‌کند.

متدولوژی یا متد نرم‌افزاری، به فراوندی که باید در درون هر مرحلهٔ مدل توسعه صورت پذیرد دلالت می‌کند. حیطهٔ تمرکز و موضوع اصلی یک متدولوژی حول محورهای زیر می‌چرخد:

۱ـ **داخل هر مرحله را چگونه می‌پیماییم**[4] ؛ تعیین داده‌ها، کنترلها، سلسله مراتب، تمایز وظایف هر بخش[5] و زیر فعالیتها، تعیین و تأمین نیازهای بخش مربوطه

۲ـ **چگونه محصولات مربوط به این مرحله را نمایش دهیم**[6]. هر محرله دارای محصولاتی است که به‌عنوان شالوده و ثمرهٔ آن مرحله و به‌عنوان محکی برای مراحل بعدی، مدیریت و تولید می‌گردد. مدلسازی ذهنی به‌صورت نمودارهای استانداردی مانند نمودار جهان داده‌ها[7]، نمودار حالت[8]، نمودارهای ای‌ـآر[9] و ... نمایش داده می‌شود. یک روش منسجم برای جمع‌آوری حاصل طراحی، احتیاج به نمودارها و جداولی دارد که حاصل کار فکری و تجربیات طراح می‌باشد. متدولوژی، باید در خصوص چگونگی به‌کارگیری روشها و ابزار پیشنهادی برای درک حوزهٔ مورد بررسی، یافتن عوامل کلیدی طراحی، یافتن راههایی برای مدلسازی صحیح حوزهٔ مسأله به‌صورت یک راه‌حل و نهایتاً نمایش این راه‌حل، راهنماییهای لازم را در اختیار طراح قرار دهد. بنابراین، اولین نکتهٔ مهم این است که مدل و متدولوژی در عین دارا بودن حیطهٔ عملیاتی خود، در حد سطوح کلیّت از یکدیگر مجزا می‌باشند به‌طوری که متدولوژی باید زیرمجموعه‌ای از مدل توسعهٔ مناسب خود باشد. یک مثال از جهان خارج می‌زنیم. متدولوژی مانند لوکوموتیو و واگنهای یک قطار است که توسعه را به جلو می‌برد؛ در حالی‌که مدل همچون ریل و خطوط راه‌آهن می‌باشد که نه تنها جهت و سرعت متدولوژی متکی به آن است، بلکه فقدان مدل مناسبی برای یک متدولوژی، تواناییهای متدولوژی را تلف نموده و امکان نیل به مقصد را تقریباً غیرممکن می‌سازد. اگر مدل متناسب و متجانس با متدولوژی نباشد، سرعت و قدرت لوکوموتیو را هدر داده و فقط انتظار حرکتی ضعیف به سمت جلو میسر خواهد بود. اکنون به اصول اولیه نگرش شیءگرا نظری گذار می‌افکنیم.

3) Transition criteria 4) phase navigation 5) Module
6) Phase product representation 7) Data Flow Diagrams
8) State Charts 9) E-R Diagrams

نگرش شیءگرا

مهمترین موضوعی که از نگرش شیءگرا باید مد نظر قرار گیرد، این است که شیءگرائی یک نگرش و یک دیدگاه برای فراروند توسعهٔ نرم‌افزار است. شیءگرائی یک جهان‌بینی نرم‌افزاری است. اساس این نگرش بر سه اصل استوار است:

- تجرید داده‌ها[10]
- وراثت یا اشتقاق[11]
- چندریختی[12]

اکنون اشاره‌ای مختصر به هر یک می‌کنیم:

تجرید داده‌ها به مفهوم پنهانسازی[13] و بسته‌بندی[14] ساختار داده‌ای است که آنرا از تیررس تغییراتی که توسط اشیاء دیگر بر آن اعمال می‌شود مصون می‌دارد و در عین حال انسجام و وحدتی میان عملیاتی که یک شیءقادر به انجام آنها است به‌وجود می‌آورد. یعنی عملیات به‌همراه ساختار داده‌ای که در شیءپنهان شده است.

وراثت به امکانی اشاره می‌کند که به‌واسطهٔ آن می‌توان اشیاء پایه[15] ایجاد نمود که دارای رفتار (عملیات) و «سیرت» (وضعیت ساختار داده‌ای داخلی آنها) خودکفایی بوده و پایه و اساسی برای تشکیل اشیایی شبیه به خود، ولی با ویژگیهای اضافی فراهم می‌سازد. به‌عبارت دیگر، اگر بخواهیم در آینده اشیایی بسازیم که مشابه آن اشیاء پایه می‌باشند، نیازی به نوشتن کد تکراری برای آنها نمی‌باشد. می‌توان با اندک تغییراتی شیءجدیدی، با امکانات مطلوب و متفاوت، ولی مشابه شیئی که وضعیت و رفتارش را به ارث برده است، خلق کنیم. اغلب، این اشیاء دارای وجوه تشابه زیادی با یکدیگر بوده و هم‌خانواده می‌باشند. هر یک، علاوه بر ویژگیهای پایه‌ای دارای خصایص ویژه خود نیز می‌باشند که هر یک را از دیگر اشیاء متمایز می‌سازد؛ مانند خصایص «پستانداران» که میان حیوانات اهلی مختلف مشترکند ولی هر کدام موجودیت مستقلی محسوب می‌شوند. برای شبیه‌سازی این وضعیت، احتیاجی به خلق مجدد رفتار یا سیرت هر شیءمشابه نیست. صرفاً، خصایص هر کدام را از یک یا چند شیءدیگر به وراثت گرفته و وجوه تمایز و ویژگیهای خاص شیءجدید را به شیءفوق می‌افزاییم.

چندریختی به آن قابلیت اشیاء اشاره می‌کند که از طریق آن بتوانیم با (ظاهراً) دستورالعمل واحدی که به‌صورت پیام[16] به موضوعات مختلف ارسال

10) Data Abstraction 11) Inheritance or Derivation
12) Polymorphism 13) Information hiding 14) Encapsulation
15) Base Objects 16) message

می‌گردد، در هر یک، رفتار و عملکرد مناسب و ویژهٔ خویش را برانگیزیم و وادار به انجام عملیات ویژه‌ای بنماییم. مثال کلاسیک این ویژگی، دستور رسم یا draw می‌باشد که به اشیاء مختلف هندسی مانند دایره، ذوزنقه، مثلث و ... فرستاده می‌شود، تا هر کدام شکل خاص خود را رسم نمایند. بنابراین، یک پیام واحد موجب بروز رفتار متفاوتی در اشیاء متفاوت می‌گردد.

متدولوژیهای شیءگرا

برای به‌کارگیری اصول نگرش شیءگرا، باید روشها و متدولوژیهایی پیشنهاد گردد تا بر اساس آنها، فرایند خلق نرم‌افزارهای شیءگرا امکانپذیر گردد. از سال ۱۹۸۴ تاکنون، متدولوژیهای بسیاری برای آنالیز، طراحی و پیاده‌سازی به روش شیءگرا ارائه شده است. این متدولوژیها شامل موارد زیر می‌باشند:

Category 1

Process only (No representation)

Bulman

Henderson - Sellers and Constantine

Johnson and Foote

Scharenberg and Dunsmore

Category 2

Representation Only (No technique)

Ackroyd and Daum-a graphical notation

Beck and Cunningham - Class - Responsibility Collaboration

Page-Jones et al. - Uniform Object Notation (UON)

Wasserman et al. - OO Structured Design Notation (OOSD)

Wilson-Class Diagrams

Category 3

Process and Representation

Alabiso - Transformation from Analysis to Design

Bailin - Object - Oriented Requirements Specification Method

Booch - Object - oriented Design

Coad and Yourdon - Object - Oriented Analysis and Design (OOA and OOD)

Gorman and Choobineh - Object - Oriented Entity Relationship Model (OOERM)

Livari - A Framework for Object Identification

Kappel - Object/Behaviour Model

Liebherr et al. - The Law of Demeter

Meyer - Object - oriented Software Construction

Rumbaugh et al. - Object Modelling Technique (OMT)

Shlaer and Mellor - Object - Oriented Systems Analysis

Wirfs - Brock et al. - Designing Object -Oriented Software

Source: [Monarchi 92]

این گروه به سه دسته متدولوژیهای ۱) دارای فرایند توسعه و ۲) دارای روش نمایش و ۳) روشهایی که هم برای روش توسعه تکنیک پیشنهاد می‌کنند و هم مکانیزم نمایش آنرا دارا می‌باشند، تقسیم می‌شود.

من حیث‌المجموع، سوابق تاریخی این تعدّد زیاد متدولوژیهای شیءگرا، از سه نگرش فکری منتج می‌شود. اولین گروه، طرفداران روشهای ساختیافته هستند (Coad and Yourdon, Shlaer, Mellor) که به متدولوژی خود رنگ و بوی شیءگرائی داده‌اند. دوم، افراد از دنیای مدلسازی داده‌ها و دنیای بانکهای اطلاعاتی، مانند (Chen, Gorman) هستند که روشهای خود را با روش شیءگرا تلفیق کرده و به نوعی به یک روش دورگه رسیده‌اند. سومین گروه، آنهایی هستند که از بنیان شیءگرا بوده و اصول اولیه جهانبینی خود را بر این نگرش استوار داشته‌اند، مانند Meyer، Rumbaugh، Borgida، Booch. گروه سوم، قویترین و کارآمدترین روشها و تکنیکهای شیءگرائی را پیشنهاد نموده‌اند [Davis 93].

ولی جالب اینجاست که علیرغم انبوه متدولوژیهای پیشنهادی، و اختلاف بر سر تکنیکها و اصول طراحی و پیاده‌سازی، تعداد مدلهای پیشنهاد شده برای بهره‌گیری از مجموعهٔ قابلیتهای شیءگرائی، در مجموعه فوق، نزدیک به صفر است. یعنی، از میان صدها مقاله [Palvia 93]، اکثر قریب به اتفاق مؤلفان، در کنار بحث پیرامون روشهای جزئی خود مثلاً چگونگی تشخیص اشیاء، کیفیت تشکیل گرافها و سلسله مراتب وراثت و نحوه‌های ارسال پیامها بین اشیاء و ... که همگی معطوف به امر متدولوژی شیءگرا می‌باشد، اشاره‌ای حتی کمرنگ به فرایند توسعه یا مدل شیءگرا نکرده‌اند. خلاصهٔ بحث آنها را می‌توان در نقطه‌نظر Grady Booch خلاصه نمود که مراحل را بدین قرار توصیف می‌کند:

تعریف مسأله

یافتن نیازهای کاربران به‌صورت غیردقیق و غیررسمی

رسمیت و دقت بخشیدن به نیازها و مشخصات

تشخیص اشیاء و رفتار

تشخیص روابط میان اشیاء

پیاده‌سازی اجزای تشخیص داده شده در یک زبان شیءگرا

اما می‌توان به‌راحتی تشخیص داد که مراحل فوق مربوط به یک مدل نبوده و در واقع حیطهٔ بحث فقط و فقط پیرامون متدولوژی می‌باشد. یعنی بر فرایندهای کنترل پروژه و کنترل کیفیت محصول نرم‌افزاری دلالت می‌کند و نه فرایند به‌دست آوردن نرم‌افزار. باید به دنبال مدلهایی مانند مدل چرخشی[17]

17) Spiral Model

بود که ریسک را در نطفه تشخیص می‌دهد و تدابیری برای رفع و حلاجی آن برنامه‌ریزی و پیاده‌سازی می‌نماید، و به فراروند کنترل پروژه و کنترل و تضمین کیفیت توجه خود را معطوف نموده، به آنها یاری می‌بخشد. ایجاد یک محصول موفق که مطابق زمانبندی پیش‌بینی شده تحویل داده شود، به ترکیبی استادانه از تکنیکهای مدیریت پروژه، تضمین کیفیت، و در نهایت خودِ امر برنامه‌نویسی متکی می‌باشد. بنابراین باید اهمیت شایانی را به شناخت و به‌کارگیری صحیح یک مدل توسعه مناسب برای هدایت متدولوژی در نظر گرفت و به انتخاب یک متدولوژی صرف، بسنده نکرد. انتخاب و به‌کارگیری مدل مناسبی برای توسعه نرم‌افزار، موفقیت همه‌جانبه‌ای را به ارمغان می‌آورد؛ در حالی‌که صرف پیروی از یک متدولوژی، فقط جنبهٔ سطحیِ تکنیکی-مکانیکی مهندسی نرم‌افزار را برآورده ساخته و ارضاء می‌نماید و دستاوردهای چشمگیری را به‌دنبال نخواهد داشت و چه بسا موجب گردد تا پروژه شکست خورده و از زمانبندی و بودجهٔ از پیش تعیین شده خود نیز عدول کند.

پیشنهاداتی برای مدل شیءگرا

به سبب اهمیتی که انتخاب و به‌کارگیری یک مدل مناسب برای یک متدولوژی مانند شیءگرائی به همراه دارد، شایسته است تا ویژگیهای بنیادی و ضروری چنین مدلی را بررسی نموده و مشخص نماییم. از اهم مواردی که علیرغم بحثهای بسیار زیاد پیرامون آن، نتیجهٔ کافی از آن حاصل نشده است، راه‌حل جامعی برای مسألهٔ باز به‌کارگیری نرم‌افزار[18] می‌باشد. باز به‌کارگیری باید در چرخهٔ زندگی نرم‌افزار شیءگرا ملحوظ گشته و مدیریت شود. برای روشن شدن بهتر فراگیری باز به‌کارگیری، سطوح باز به‌کارگیری را به‌ترتیب از سطح عینی تا تجریدی به‌شرح زیر تعریف می‌کنیم:

سطوح باز به‌کارگیری

۱ـ سطح رده ـ روالهای رده‌ها
۲ـ ساختارهای منطقی میان‌رده‌ای ـ روابط میان مجموعه‌هایی از رده‌ها
۳ـ معماری و نمای برون‌رده‌ای
۴ـ مجموعه اجزای کاربردی
۵ـ دانش حیطهٔ کاربردی
۶ـ دانش توسعهٔ نرم‌افزار
۷ـ اطلاعات محیطی

در اینجا، منظور از باز به‌کارگیری، صرفاً استفاده مجدد از «کد برنامه» نیست؛ بلکه استفاده مجدد از جمیع محصولات و اطلاعات شکلبندی شده در تمام مراحل مدل توسعه نرم‌افزار می‌باشد.

18) Software reuse 19) Reuse Management 20) Risk Management

برای این منظور، مدیریت باز به‌کارگیری[19] باید به مدل توسعهٔ نرم‌افزار افزوده شود، و باید درکنار مدیریت ریسک[20]، مدیریت پیکربندی[21] و تضمین وکنترل کیفیت[22] جزء لاینفکی از فراروند و فعالیتهای کنترل پروژه به‌شمار آمده و به‌کار گرفته شود. محصول هر مرحله نرم‌افزار، اطلاعاتی را شکلبندی می‌نماید. نحوه این شکلبندی باید به‌گونه‌ای برنامه‌ریزی شده باشد که شکلبندی آن با استفاده مجدد از اطلاعات درون آن، سازگار و هماهنگ بوده و به سهولت قابل استفاده مجدد در همان پروژه و یا پروژه‌های دیگر باشد.

ذیلاً، به‌طور اخص، نحوه تشخیص و مدیریت تجدید استفاده، چنانکه در مدل شیءگرا مد نظر است، شرح داده شده است. قابلیت باز به‌کارگیری با استفاده از هفت اصلی مطروحهٔ زیر، که از ارکان مدل شیءگرا هستند، قابل برنامه‌ریزی و پیاده‌سازی خواهد بود. موارد «آ» تا «چ» که ذیلاً ارائه می‌گردد، حیطه‌هایی از مراحل توسعه را درکنار محدوده‌هایی از محصولات برای باز به‌کارگیری در نظر می‌گیرد. یعنی درکلیهٔ مراحل آنالیز، طراحی مفهومی، طراحی تفصیلی، پیاده‌سازی و نگهداری تدابیر زیر محصولاتی ایجاد خواهند نمود که قابل استفاده مجدد باشند. درکنار سطوح مختلف باز به‌کارگیری، ذیلاً اصولی را برای تحقق و خلق آنها، براساس تجربیات ایجاد سیستمهای متوسط تا بزرگ نرم‌افزاری براساس روش شیءگرا، ارائه می‌نمائیم.

آ. **سطح اول، رفتارو وضعیت شیء:** به‌واسطهٔ اشتقاق می‌توان از رفتار (روشها[23]) و وضعیت یک شیء (ساختار داده‌ای درونی[24]) به‌طور خودبه‌خود، باز به‌کارگیری نمود. ولی برای ایجاد شبکه‌های اشتقاق، که در سطح بسیارگسترده‌تری امکان باز به‌کارگیری را فراهم می‌سازد، باید برنامه‌ریزی دقیق و مشخصی را از همان ابتدای طراحی انجام داد. صرف تکیه بر توان یک زبان برنامه‌نویسی برای فراهم سازی امکان اشتقاق کفایت نخواهد کرد. صرف تکیه بر توان یک زبان کفایت نمی‌کند.

> اصل اول. شبکه‌های اشتقاق و وراثت باید در مراحل ابتدائی مدل توسعه، به‌طور رسمی و مشخص برنامه‌ریزی گردد.

هر رده را می‌توان به‌عنوان بخش قابل استفاده مجدد، در سطح اول در نظر گرفت

ب. **سطح دوم، ساختارهای منطقی میان‌رده‌ای:** شامل ساختارهای داده‌ای و روالی نرم‌افزار و شامل تشخیص اجزاء و روابط میان آنها بوده بدون آنکه این ساختارها را به نوع و شیوهٔ خاصی از نمایش و پیاده‌سازی وابسته کنیم. این ساختارهای منطقی عبارتند از روابط میان مجموعه‌ای از رده‌ها.

21) Configuration Management 22) Quality Assurance and Control 23) Methods 24) State

> اصل دوم. مفهوم منطقی ساختارهای درون‌رده‌ای را از لحاظ اجزاء و روابط تشخیص داده، برای بازبه‌کارگیری به‌شکلی قابل دسترس برای همکاران خود در پروژه قرار دهیم.

به‌عنوان مثال اگر بخواهیم اصل فوق را توسط یک زبان شبه‌شیءگرا مانند C++ پیاده‌سازی کنیم، ابتدا واسطهای رده‌ای[25] را معین نموده و به اعضای دیگر تیم از طریق پست الکترونیکی، یادداشت داخلی و یا اعلام آن در جلسات باز به‌کارگیری مطرح نمائیم تا آنها بتوانند محصولات طراحی مفهومی جدیدی را که به آنها اعلام کرده‌ایم، به لیست مفروضات خود اضافه کرده و در طراحی خود نیز بتوانند این اجزای جدید را ملزوم نمایند. دومین عنصر مهم در اصل دوم از باز به‌کارگیری، تشخیص، ثبت و اعلام روابط میان رده‌ها[26] که به طریق فوق در شبکهٔ طراحان قرارگیرد و در مقاطع آتی پروژه نیز قابل دسترس و تجدید استفاده باشد.

پ. **سطح سوم، معماری و نمای برون‌رده‌ای:** کتابخانهٔ رده‌ها[27] در اینجا جایگاه خود را پیدا می‌کند. این رده‌ها نباید پس از اتمام سیستم ایجاد شود، بلکه باید خلق آنها در ابتدای چرخهٔ طراحی در نظر گرفته شده و برنامه‌ریزی گردد.

> اصل سوم. معماری و نمای ساختار عملیاتی سیستم، آنچنانکه کاربران و استفاده‌کنندگان آنرا رؤیت خواهند نمود باید برنامه‌ریزی، مستند و برای استفاده همگان قرار گیرد. این امر هم مربوط به کاربران طراح و برنامه‌نویس و هم به کاربران مدیر می‌شود.

مثال مناسبی برای این اصل کتابخانهٔ OWL[28] مربوط به شرکت Borland International می‌باشد، که کتابخانهٔ کاملی برای ایجاد برنامه‌های کاربردی تحت MS-Windows است. در عین حال باید توجه داشت که غیر از کتابخانه‌های کاربردی، کتابخانه‌های دیگری مانند کتابخانهٔ NIH Container Class وجود دارد که نسبت به OWL در سطح بالاتری از تجرید قرار دارد.

ت. **سطح چهارم، مجموعهٔ اجزاء کاربردی شیءگرا:** برای ایجاد و بهره‌گیری از معماریهای ترکیبی و زیرسیستمهای مستقل که بطور کلی رفتار هماهنگی را از خود نشان می‌دهند، مانند یک صفحه گسترده یا ویراستار صفحه‌ای، باید برنامه‌ریزی و طراحی لازم را به‌عمل آورد. به این نوع اطلاعاتِ قابل استفادهٔ مجدّد، کتابخانه اجزاء[29] اطلاق می‌گردد. این محصولات، یک سطح بالاتر از سطح کتابخانه‌های رده‌ها بوده* و سرویسها و رفتار ترکیبی و پیچیده‌تری را، همچون یک مجموعه مدار مجتمع که برای یک تخته مدار خاص طراحی شده‌اند، از خود نشان می‌دهد. نمونه‌ای از این نوع تجدید استفاده، کنترلهای ویژهٔ Visual Basic در Custom Controls (.VBX) می‌باشد.

> اصل چهارم. ترکیب هوشمندانه یک مجموعه اجزای شیءگرا یا کتابخانهٔ اجزای کاربردی، به‌طوری‌که مجموعاً سرویسهای مرکّبی را در حیطهٔ کاربردی معین (مانند حسابداری)، ارائه دهند از تشکیل کتابخانه رده‌ها مهمتر و در درازمدت، مقرون به‌صرفه‌تر می‌باشد.

ث. **سطح پنجم، دانش خارج از نرم‌افزار (دانش حیطهٔ کاربردی):** این دانش خارج از حیطه نرم‌افزار است و به جنبه‌های رفتاری بخشی از دنیای خارج مربوط می‌گردد. این دانش بر دو نوع است:

- دانش حیطه کاربرد[30]
- دانش توسعه[31]

دانش حیطه کاربرد قابلیت به‌کارگیری مجدد این دانش نسبی است. برخی از حیطه‌های کاربردی مانند کاربردهای علمی و مهندسی و یا مثلاً حسابداری، مدون و مشخص و قابل استفاده مجدد هستند. اما موضوعاتی مانند هتل‌داری یا علوم اجتماعی ممکن است نادقیقتر، غیر مدون و در نتیجه، بسیار دشوار برای استفاده مجدد باشند. در این نوع دانش، عناصری که می‌توان در سطح آنالیز مورد تجدید استفاده قرار داد، عبارتند از روابط مفهومی میان اجزای حیطهٔ کاربرد (مانند ارتباط سند با دفتر روزنامه و دفتر کل در حسابداری). اکثر اوقات، معرفت کسب شده توسط تحلیل‌گر به‌صورت ایستا در زونکنهایی در قفسه‌های خود مشغول خاک خوردن هستند. برای اینکه این تجارب ارزنده باز به‌کارگیری شوند، نیاز به بازنمایی این معرفت کسب شده توسط مکانیزمی است که قادر به نمایش ارتباطات پویای مفهومی باشند. برای این منظور توصیه می‌گردد که برای باز به‌کارگیری معرفت مفهومی این حیطه از دنیای واقعی، این دانش توسط ابزاری که مجهز به آبرمتن[32] بوده ذخیره و بازنمایی شود. از این‌رو، حاصل تحلیل، که اغلب به‌صورت متون فاقد ساختار و یا دارای ساختارهایی

25) Class Interfaces 26) Class relationships and dependencies
27) Class Libraries 28) Object Windows Library

29) Component Libraryor or Generic system *) Class Library
30) Application-domain knowledge 31) Development knowledge 32) Hypertext

هستند که به‌سرعت تغییر می‌کنند، را بتوان بازنمایی نموده نکات حائز اهمیت را تأکید و روابط میان اجزاء را به‌سهولت برقرار نموده و نمایش داد.

> اصل پنجم. **بازنمایی معرفت حیطه کاربردی برای باز به‌کارگیری.** باید تمام داده‌های جمع‌آوری شده و معرفت کسب شده در هر یک از زمینه‌های کاربردی دنیای واقعی، اعم از دقیق و نادقیق بودن این اطلاعات، ثبت گردیده و طوری مستند شود که دیگران نیز بتوانند از این اطلاعات استفاده نمایند.

ج. **سطح ششم، دانش توسعه نرم‌افزار.** این دانش شامل مراحل و فازهای مدل توسعه، نتایج مراحل قبل، محصولات حاصل از کاربرد متدولوژی نرم‌افزاری، لیست ملاحظات برای کنترل کیفیت در هر مرحله، نتایج حاصله از جلسات تحلیل، طراحی، پیاده‌سازی و آزمون نرم‌افزار و کلیه خط‌مشیهای متخذه توسط همکاران دخیل در پروژه می‌باشد.

> اصل ششم. **دانش و تجربیات کلیدی حاصل از انجام هر فعالیت** از پروژه در خلال فرایند توسعه نرم‌افزار باید ثبت و مستند و در دسترس افراد تیم پروژه قرار گیرد. این اطلاعات، منبع غنی‌ای از تصمیمات و توجیه سیاستهای متخذه در برابر شرایط غیرعادی و معمولی در طول عمر پروژه می‌باشد.

برای این بخش از باز به‌کارگیری، باید در بانک اطلاعاتی باز به‌کارگیری مربوط به پروژه[33]، یک بانک اطلاعاتی ثبت تصمیمات متخذه ایجاد و نگهداری شود. نمونه‌ای از مندرجات این مخزن باز به‌کارگیری به‌شرح زیر می‌باشد:

- نوع جلسه • تاریخ • شرکت‌کنندگان
- مرحلهٔ پروژه (تحلیل، طراحی، ...)
- ریسکها
- موضوع/صورت مسئله
- تصمیمات متخذه و دلایل

این مخزن نگهدارندهٔ تجارب کسب شده در پروژه بوده و تصمیمات اتخاذ شده را مستندسازی می‌نماید. از این الگوها در پروژه‌های دیگر نیز می‌توان بهره‌برداری نمود.

33) Project Reuse Repository

ج. **سطح هفتم، اطلاعات محیطی.** شامل چگونگی ترکیب عوامل محیط مسأله با سیستم نرم‌افزاری توسعه‌یافته می‌باشد. برای مثال، چگونه سیستم نرم‌افزاری حاصله در یک سیستم حسابداری در یک مؤسسه دولتی باید جایگاه خود را پیدا کند. این اطلاعات ارتباط سیستم نرم‌افزاری با عوامل محیطی است و موفقیت نهایی پروژه به مقبولیت سیستم توسط کاربران و محیط دنیای واقعی بستگی دارد. اطلاعات تحلیلگر در این خصوص به‌گونه‌ای قابل استفاده و دستیابی برای همه باید قرار گیرد. برای مثال فهرست اشیاء و عوامل دنیای خارجی مسأله در بانک اطلاعاتی یا حتی دفترچه‌ای نمایه‌دار[34] ثبت می‌گردد.

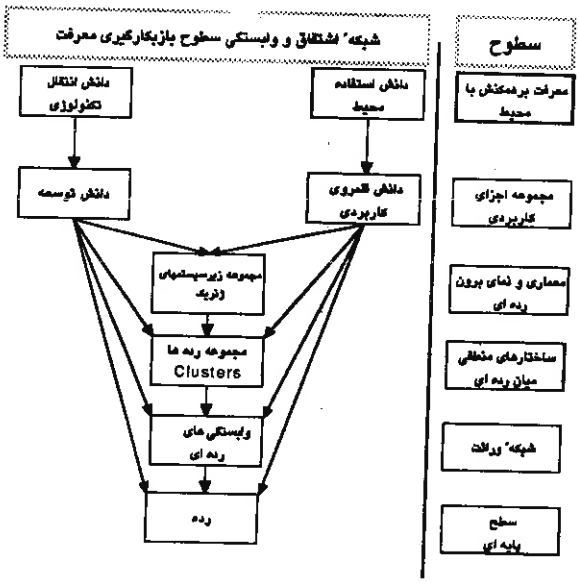
دومین بُعد اطلاعات محیطی، چگونگی انتقال تکنولوژی جدید به محیط مسأله از طرفی و طراحان سیستم از طرف دیگر، می‌باشد. دسته‌بندی و متمرکزسازی این اطلاعات در بانک اطلاعاتی تحلیل پروژه، گامی در جهت تجدید استفاده و جلوگیری از، از دست رفتن دانش کسب شده در مراحل کلی و ابتدائی مدل نرم‌افزاری می‌باشد.

> اصل هفتم. اطلاعات تحلیل سیستم، به‌خصوص اطلاعات مربوط به ارتباط سیستم نرم‌افزاری و دنیای واقعی که در آن قرار خواهد گرفت، و نیز مکانیزمها و تجربیاتی در خصوص انتقال تکنولوژی به حیطهٔ مسأله و نیز طراحان، باید از مراحل اولیه پروژه، در بانک اطلاعاتی بازبکارگیری مربوطه به نحو قابل استفاده‌ای ثبت گردد.

اگر به محصولات قابل استفاده مجددی که از طریق اصول فوق تولید و نگهداری می‌شوند، نگاهی شیءگرا بیاندازیم، ارتباط وراثتی و اشتقاقی میان آنها را مشاهده خواهیم نمود.

حاصل تجزیه و تحلیل سیستم به‌صورت متون آبَرمتن در رده‌های پایه‌ای[35] یک پروژه کاربردی قرار خواهد گرفت. دانش مدون شده در مراحل دیگر مدل نرم‌افزاری را می‌توان به‌عنوان وارثین سطوح بعدی تلقی نمود. این شبکهٔ وراثت تا رسیدن به مرحلهٔ کد برنامه که از مرحلهٔ طراحی تفصیلی مشتق شده است ادامه خواهد داشت. این شبکهٔ اشتقاق[36] در پایینترین سطح دارای «برگهایی»[37] می‌باشد که عبارتند از برنامه‌های قابل اجرا. مسیر اشتقاق این برنامه‌ها از طراحی تفصیلی تا تجزیه و تحلیل بدین طریق کاملاً روشن خواهد بود. بنابراین در این مدل توسعه امکان ردیابی نیازمندیها[38]، صحت‌یابی[39] برنامه به‌طور طبیعی و خودبه‌خود از درون مدل توسعه به‌وجود می‌آید.

34) Indexed 35) Base Class 36) Inheritance Network
37) Leaf nodes 38) Requirements traceability 39) Verification and validation



شکل۱: نمودار وراثت اجزای باز به‌کارگیری

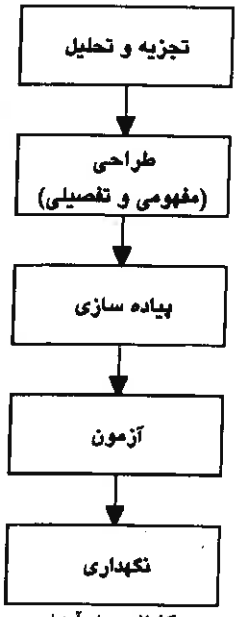
اهمیت تطابق مدل با متدولوژی توسعه نرم‌افزار

اگر مدل براساس نگرش فراروندی[40] و متدولوژی بر مبنای دید داده‌ای[41] باشد، عدم تطابق نگرشها موجب اتلاف منابع و پایین آوردن کیفیت محصول خواهد گشت. عدم همسویی و انسجام نگرشها، موجب دوباره‌کاری و تبدیلات غیر ضروری می‌گردد. بنابراین برای متدولوژی شیءگرا، بهتر است از یک مدل شیءگرا پیروی کنیم.

باید از یاد نبرد که وقتی صحبت از مدل شیءگرا به میان می‌آید، منظور این است که کلیه مراحل چرخه زندگی نرم‌افزار[42] به‌صورت شیءگرا انجام پذیرد؛ نه صرفاً یک مرحله مانند تحلیل یا پیاده‌سازی. اما اکنون مدل واحدی برای توسعهٔ شیءگرا به چشم نمی‌خورد که جمیع مراحل چرخه زندگی را پوشش دهد. اغلب روشهایی که نزدیک به این امر می‌شوند، صرفاً یک یا دو مرحله از چرخه زندگی را در بر می‌گیرد. این دو مرحله معمولاً مراحل تحلیل و طراحی می‌باشند. اما اگر دقت خودمان را معطوف به کنه مطلب نماییم، دریافت خواهیم نمود که نگرش ماهیتاً آبشاری است و تسلسل و ترتیب مراحل تحلیل و طراحی و پیاده‌سازی در متدولوژیهای شیءگرا مستتر می‌باشد. خود چرخهٔ زندگی را می‌توان مانند یک شیء در نظر گرفت که دارای عملیات سنتی آبشاری نیز می‌باشد، مانند: تحلیل، طراحی، پیاده‌سازی، آزمون، نگهداری و مستندات. اما ترتیب به‌کارگیری هر مرحله لزوماً مطابق مدل آبشاری نیست و مانند مدل چرخشی می‌تواند شامل مراحل کاملتر زیر باشد:

۱ـ تعیین محدودیتها، عملکرد مطلوب، اهداف، ریسکها

40) Process-oriented 41) Data-oriented 42) Software Life-Cycle



شکل۲: مدل آبشاری

۲ـ تولید

۳ـ بررسی نتایج مرحله حاضر

۴ـ برنامه‌ریزی برای مرحله بعد

اما مهمترین عنصری که باید در یک مدل شیءگرا با اولویت بالا گنجانده شود، مسألهٔ باز به‌کارگیری نرم‌افزار است. این امر مربوط به کلیهٔ مراحل چرخه زندگی می‌باشد و ترجیحاً باید توسط یک محیط نرم‌افزاری پشتیبانی شود.

مدلهای سنتی، شیءگرائی را محدود می‌کنند

فرض مدلهای سنتی مانند آبشاری و تکاملی، استفاده از روشهای ساختیافته بوده است. با این‌که، شیءگرائی، حالت پیشرفته‌تری از ساختیافتگی به ارمغان می‌آورد، اما سعی در تزریق شیءگرائی در قالبی صرفاً ساختیافته بسیاری از ویژگیهای آنرا عقیم می‌سازد. کاربرد متدولوژی شیءگرا، در چهارچوب مدلهای سنتی همچون مدل آبشاری، تکاملی و نمونه‌سازی[43] محدودیتهای مصنوعی ناشی از جهان‌بینی صرفاً ساختیافته و پیمانه‌ای را بر آن متدولوژی تحمیل می‌نماید. برای بهره‌برداری از جمیع فواید و امکانات متدولوژی شیءگرا، شایسته است که آنرا بر پایهٔ یک مدل خاص خود، استوار نموده و مورد بهره‌برداری قرار دهیم. سنخیت، تجانس، انطباق و همنگرش بودن مدل با متدولوژی، تعیین‌کنندهٔ میزان موفقیت و حاصل به‌کارگیری تکنیکهای آن متدولوژی به‌شمار می‌رود [Agrestil 86].

مدل شیءگرای توسعه نرم‌افزار

برای استفاده از کلیه مزایا و امکاناتی که رهیافت شیءگرا در اختیار طراحان و برنامه‌سازان قرار می‌دهد، باید از مدلی استفاده کنیم که ماهیتاً شیءگرا باشد. زیرا هر یک از مدلهای سنتی مانند مدل آبشاری، بدون توجه به امکانات

43) Prototyping

جدیدی که شیءگرائی در اختیار ما قرار می‌دهد، توسعه نرم‌افزار را به بخشهای زیر تقسیم می‌نماید:

تحلیل، طراحی مفهومی، طراحی تفصیلی، پیاده‌سازی و برنامه‌نویسی، اشکالزدایی و آزمون برنامه، مستندسازی و نهایتاً، نگهداری

برخورد خطی فوق، با فرارونـد ماهیتاً چندبعدی و گاهی همزمان توسعه سیستمهای نرم‌افزاری شیءگرا، به‌وضوح بر پایه این نگرش استوار شده است که فاصله منطقی میان دامنه مسأله[44] و دامنه راه‌حل[45] آنقدر زیاد است که طراح، تجریدی از دنیای مسأله را خلق نموده، سپس مدلی از آنرا پرورش می‌دهد و در آن دنیای فرضی خود که بسیار دور از دنیای واقعی است، سعی در حل مسأله دارد. آنگاه، سعی دارد راه‌حل خود را در قالب یک زبان با محیط برنامه‌سازی ریخته و اطمینان نسبی حاصل نماید که برنامه با طراحی راه‌حل تجریدی‌اش تناقض نداشته و همسو باشد. (شکل ۱)

هر چقدر فاصله میان این دو فضا یا دامنه زیادتر باشد، فرارونـد طراحی به دشواری بیشتری قابل پیاده‌سازی خواهد بود و وارسی و اعتبارسنجی[46] آن، امری پیچیده‌تر و با احتمال بالاتری از خطا مواجه خواهد شد. هر چقدر بتوانیم فضای مسأله را به مدل راه‌حل نزدیکتر سازیم، امکان افزایش آنتروپی اطلاعاتی و لذا خطای پیاده‌سازی کمتر و کمتر خواهد شد. بحث این است

Problem Space فضای مسأله (دنیای واقعی)	فاصلهٔ مفهومی	Solution Space فضای راه‌حل (مدل ذهنی)

که مطلوب ما، به‌عنوان مهندس نرم‌افزار، این است که با سرعت بیشتر و هزینه پیشبینی شده، سیستمهای قابل اطمینان و با کیفیتی را پیاده‌سازی کنیم که احتیاجات روزمره بخشی از دنیای واقعی را برای انسانها به نحو مطلوبتری مرتفع سازد.

در حالی که، فرارونـد توسعه، به‌علت تغییرات زیاد نیازمندیهای اولیهٔ کاربران و برخورد با مشکلات و معضلات طراحی، به شکلی تکرارپذیر[47] و افزایش‌پذیر[48] مبدل می‌گردد. همهٔ مؤلفین شیءگرا به این حقیقت اذعان دارند. تفاوت یک مدل ماهیتاً شیءگرا این است که اولاً، در هفت سطح بر شمرده شده از باز به‌کارگیری، اقدامات لازم برای خلق هوشمندانهٔ قطعات نرم‌افزاری و معرفتی که قابلیت باز به‌کارگیری داشته باشند، از ابتدای چرخهٔ زندگی، برنامه‌ریزی، پیگیری و پشتیبانی می‌شود. به‌جای اینکه در هر گذر از دور تجدید نظر در تحلیل، طراحی و یا پیاده‌سازی، صرفاً نسبت به مشخصات سیستمی که در حال ساخت آن هستیم، نزدیکتر شده باشیم، بلکه در هر قدم از این گذرهای دوّار، محصولاتی خلق می‌شود که قابلیت استفادهٔ مجدد را خواهد داشت. این امر، تأثیر مهمی بر تحقق اهداف اقتصادی، زمان‌بندی، تحقق نیازهای کاربران گذاشته و معیارهای کیفیت نرم‌افزار (قابلیت اطمینان، انعطاف‌پذیری، صحت طراحی، قابلیت تکامل نیازها و بهبود زندگی انسانها و . . .) را تا حد بسیار زیادتری ارضاء خواهد نمود.

بنابراین، رکن اساسی و اولیه، یک مدل شیءگرا، اولویت بخشیدن به مدیریت باز به‌کارگیری می‌باشد. توسعه با دید شیءگرا باید پروژه را ابتدا از دیدگاه تولید و بهره‌برداری از بخشهای قابل استفادهٔ مجدد[49] مورد بررسی و برنامه‌ریزی قرار دهد. سپس فعالیتها و مکانیزم تخصیص کار را بر این اساس ترتیب دهد که هر سیکل از نمودار ۲، مجموعه محصولات قابل استفاده مجددی ایجاد می‌نماید. بنابراین اولین بخش یک مدل که بیانگر ترتیب مراحل است به‌قرار زیر می‌باشد. توضیح اینکه هر یک سیکل از دوایر نمودار ۲،

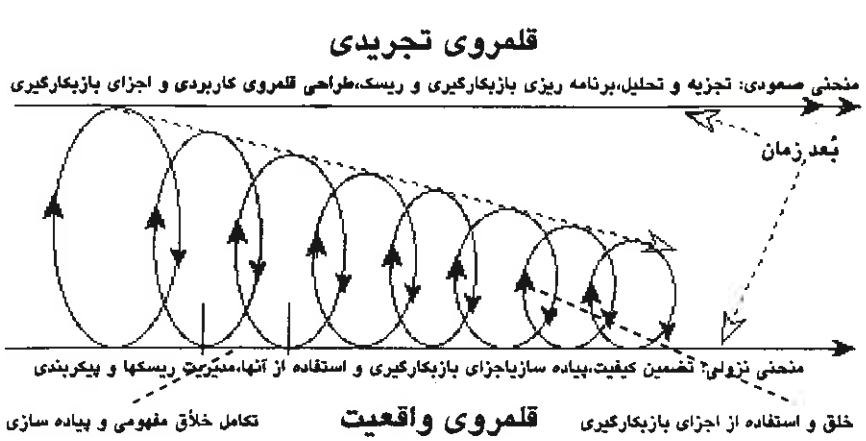
- در منحنی صعودی خود شامل گردآوری داده‌های مناسب در اولین دایره و یا ارزیابی عملکرد مرحلهٔ قبل و تعیین ریسکها، اهداف، قطعات قابل استفادهٔ مجدد، کنترل کیفیت و مدیریت پیکربندی، صحت‌یابی و مستندسازی و
- در بخش نزولی (از تجرید به‌سوی واقعیت در نمودار ۲) شامل به‌کارگیری اجزاء قابل استفادهٔ مجدد (تولید شده در همین پروژه یا پروژه‌های قبلی)، ایجاد اجزای جدید باز به‌کارگیری در سطوح هفت‌گانه درکنار بررسی، تجزیه و تحلیل، طراحی مفهومی، تفصیلی و پیاده‌سازی قطعات برنامه‌ریزی شده می‌باشد

لذا هر دور، موجب خلق محصولاتی می‌شود که قابلیت استفادهٔ مجدد خواهند داشت و هر دور نمونه‌ای از یک مقطع تکامل یافته‌تر سیستم می‌باشد. لازم به تأکید است که قطعات قابل استفادهٔ مجدد منحصر به «کُد» یا بخشهایی از برنامه یا حتی صرفاً رده‌های اشیاء نمی‌باشند و شامل تکه‌های قابل استفادهٔ مجدد از آنالیز و طراحی خواهند بود. هر چه زمان می‌گذرد و به‌سوی سمت راست نمودار ۲ حرکت می‌کنیم تجرید و واقعیت همگرا شده و سیستم در حال تولید، در مسیر یک تکامل خلاق قرار خواهد گرفت. توسعه‌گران، در هر دور مجموعه قطعاتی شامل قطعات تجزیه و تحلیل تا قطعات پیاده‌سازی به‌عنوان محصولات آن دور تکامل خلق می‌کنند.

پس از مقطع مشخص و برنامه‌ریزی شده‌ای از مرحله تحلیل، که به یک «وضعیت تشخیصی پایدار اولیه»[50] رسیدیم، طراحی در کنار ادامهٔ تحلیل آغاز شده و پس از رسیدن به یک SICS در مرحله طراحی، پیاده‌سازی را برای تحقق این بخش خودکفا که اغلب یک داده مجرد را تشکیل می‌دهد که بعداً مبنایی برای وراثت قرار خواهد گرفت، فعال خواهد ساخت. در این میان، آنالیز و طراحی احتمالاً فرارونـد خود را به پیش برده‌اند و تغذیهٔ اطلاعاتی را برای مراحل پایینتر فراهم خواهند آورد. به این مدل شیءگرا، مدل تکامل خلاق بازبکارگیری[51] گویند. در طی هر دور از این چرخه تکاملپذیر، جزیی

44) Problem dimain 45) Solution domain 46) Verification and validation 47) Iterative 48) Incremental

49) Reusable component-ware 50) Stable component identification state SCIS 51) Creative Reuse Evolution



توسعه بر اساس مدل تکامل خلاق

نوین و قابل استفادهٔ مجدد خلق می‌گردد. بنابراین مدل پیشنهادی براساس این دید استوار است که ما درصدد ایجاد و بهره‌گیری از قطعات «قابل استفادهٔ مجدد» نرم‌افزار که شامل هفت سطح بازیکارگیری می‌باشد، هستیم. در این راستا، در تمام مراحل مدل توسعهٔ نرم‌افزار می‌خواهیم اجزائی تشکیل دهیم که قابل استفاده و مصرف مجدد باشند.

مزایای رهیافت شیءگرا

ضمانت تحقق بیشتر اهداف مهندسی نرم‌افزار

بدیهی است که از تجرید داده‌ها، قابلیت توسعه و نگهداری منتج می‌گردد، از وراثت، توانایی تجدید استفاده و در نتیجه سرعت بیشتر توسعه نرم‌افزار حاصل می‌شود. از چندریختی و تجرید داده‌ها توأماً، توان دست و پنجه نرم کردن با مسائل و سیستمهای بسیار حجیم و بغرنج به‌دست می‌آید. بنابراین، قابلیت اطمینان و کیفیت نرم‌افزار افزایش چشمگیری می‌تواند پیدا کند.

مناسب بودن برای سیستمهای بزرگ و بغرنج

مزیت دیگر اینست که این روش، به‌دلیل این‌که قادر است به‌راحتی و به‌صورتی طبیعی، دامنه مسأله را به داده‌های مجردی تقسیم نماید که با پیغامهای متبادله میان خود، دنیای خارجی را شبیه‌سازی می‌کنند چسبندگی زیاد درونی[52] و وابستگی تزویجی اندک[53] از خود نشان می‌دهند، برای تحلیل، طراحی و پیاده‌سازی سیستمهای بزرگ و پیچیده بسیار مناسب و کارآمد می‌باشند.

شبیه‌سازی دامنهٔ دنیای خارجی توسط دامنهٔ راه‌حل

در واقع، با تقلیل فاصله منطقی نگاشت میان فضای مسأله و فضای راه‌حل، از طریق وجود تناظر یک به یک میان اجزای موجود در دنیای خارج و اشیاء مدلسازی شده، و تجمع داده‌ها و رفتار شیءدریک پیکر واحد، سیستم در واقع توسط طراح شبیه‌سازی می‌شود. لذا نیازی به تبدیل و ترجمه مفاهیم فضای مسأله به فضای راه‌حل و بالعکس نمی‌باشد. زیرا این تبدیلات مفهومی و نگاشتهای ادراکی، دست‌کم موجب افزایش دو برابر آنتروپی اطلاعاتی، به مفهوم کم شدن تعبیر اطلاعات و از دست دادن محتوای مفهومی و نیز دانش تجربی اطلاعات می‌گردد. در روشهای دیگر که فاصله بین فضای مسأله و فضای راه‌حل بیشتر است، نیازهای عملیاتی درخواست شده از جانب دنیای واقعی با سرویسهای پیاده‌سازی شده، دارای تناظر یک به یک نبوده و لذا وارسی و اعتبارسنجی دشوارتر و نادقیقتر می‌گردد.

محدودیتهای رهیافت شیءگرا

۱ـ ریزش خطاهای تحلیل و طراحی

علیرغم منافع و مزایای فراوان بالقوه شیءگرائی، اگر طراح از همان مرحله تحلیل، اشتباهاتی در خصوص انتخاب اشیاء مرتکب شود، این خطاها که شامل عدم تطابق دنیای واقعی با طرح ذهنی است، به مراحل پایینتر طراحی و پیاده‌سازی منتقل خواهد شد. یعنی انتخابات غلط به‌طور خود به‌خود به مراحل بعدی ریزش خواهند نمود. این امر استفاده از این روش را قدری بغرنجتر از روشهای دیگر می‌نماید. درست است که در هر متدولوژی، خطاهای مراحل بالاتر به مراحل پایینتر انتقال پیدا می‌کنند، ولی در مورد روش شیءگرا، این امر بسیار شدیدتر و مخرّبتر است. دلیل این امر از یکی از منافع شیءگرائی ناشی می‌شود که بدان «یکپارچگی مراحل مدل توسعه»[54] گویند. این مفهوم از آنجا نشأت می‌گیرد که موجودیتها، عملیات و داده‌های مجرد و روابط و تفکیک سیستم بر اساس آنها از زمان تحلیل عیناً به مرحلهٔ طراحی مفهومی، جزیی و پیاده‌سازی منتقل می‌شوند.

۲ـ کندی فراگیری و کسب تجربه و مناسب بودن برای کارهای تیمی

منحنی یادگیری شیءگرائی، در برنامه‌نویسان و طراحانی که به روشهای غیر شیءگرا عادت دارند، حاکی از کند بودن فرایند یادگیری و تثبیت شدن مطلب درکاربردهای شغلی روزمره می‌باشد. لذا زمان یادگیری طولانیتر و احتیاج به کسب تجربه‌های جدید در زمینهٔ استفاده از این روش، به‌طور عملی بیشتر می‌باشد. بنابراین، به‌طور کلی این روش، برای شرکتهایی که دارای تعدادی برنامه‌نویس می‌باشند که به‌صورت انفرادی کار می‌کنند، (تک برنامه‌نویس) و کارهایی که نیاز به تیمهای برنامه‌نویسی ندارند، مقرون به صرفه تشخیص داده نمی‌شود.

۳ـ عدم وجود قواعد ترتیبمندی

مسأله سوم این است که با وجود توسعه کتابخانه‌های رده‌ها که تجدید

52) High Cohesion 53) Low Coupling

54) Seamless Development Model Stages

استفاده را تشویق و حمایت می‌نماید، ولی ارتباطات میان متدها که تعداد آنها برای یک شیءپایه می‌تواند به چندین ده متد برسد، اصلاً بدیهی نبوده و ترتیب و وابستگی دو به دوی هر متد نسبت به دیگری و حالتی که موجب تغییر آن می‌گردد به هیچ نجوی در شیءثبت نشده و برای برنامه‌سازان و طراحان ایجاد اشکال و ابهام می‌نماید. یک شیءگرافیکی را در نظر بگیرید. برای آماده‌سازی سیستم گرافیکی می‌توان از ترکیب چندین متد به‌صورت پی در پی استفاده نمود. یک مبتدی هیچگاه نمی‌داند که اول کدام متد و سپس کدام متدها را می‌تواند مورد استفاده قرار دهد.

نتیجه‌گیری

برای بهره‌گیری از تمام قابلیتهای نگرش و متدولوژی شیءگرا، باید مدلی از توسعه نرم‌افزار به‌کارگرفته شود که قادر به متجلی کردن خواص و خصایص ویژه و شاخص نگرش شیءگرا باشد. هدف این نوشتار ارائه چنین مدلی بوده است. از طریق این مدل، می‌توان در پروژه‌های توسعه سیستمهای نرم‌افزاری که در آنها تیمهایی از توسعه‌گران مشغول کار هستند، نگرش شیءگرا به‌نحو صحیح و موفقی به‌کارگرفته شود تا به اهداف مهندسی نرم‌افزار نزدیک شویم و پروژه‌ها در زمان مناسبتر و با هزینه‌های معتدلتر و فشار عصبی کمتر به هدف اصلی خود، که رفع مشکلات انسانها، از طریق پیاده‌سازی سیستم کامپیوتری است، نایل شوند.

مراجع

1 [Agresti 86] Agresti, William, "What are the new Paradigms?", *New Paradigms For Software Development*, IEEE Computer Society Press, 1986.

2 [Boehm 87] Boehm, Barry, "A Spiral Model of Software Development and Enhancement", *IEEE Tutorial: Software Engineering Project Management*, IEEE Computer Society Press, 1990.

3 [Booch 84] Booch, Grady, "Object Oriented Design", *Software Design Techniques*, IEEE Computer Society Press, 1987.

4 [Davis 93] Davis, Alan M., *Software Requirements: Objects, Functions, and States*, Prentice-Hall International, 1993, pp. 59-61.

5 [Griss 93] Griss, M.L., "Software Reuse: from Library to Factory", *IBM Systems Journal*, Vol 32, No. 4, 1993, pp. 548-567.

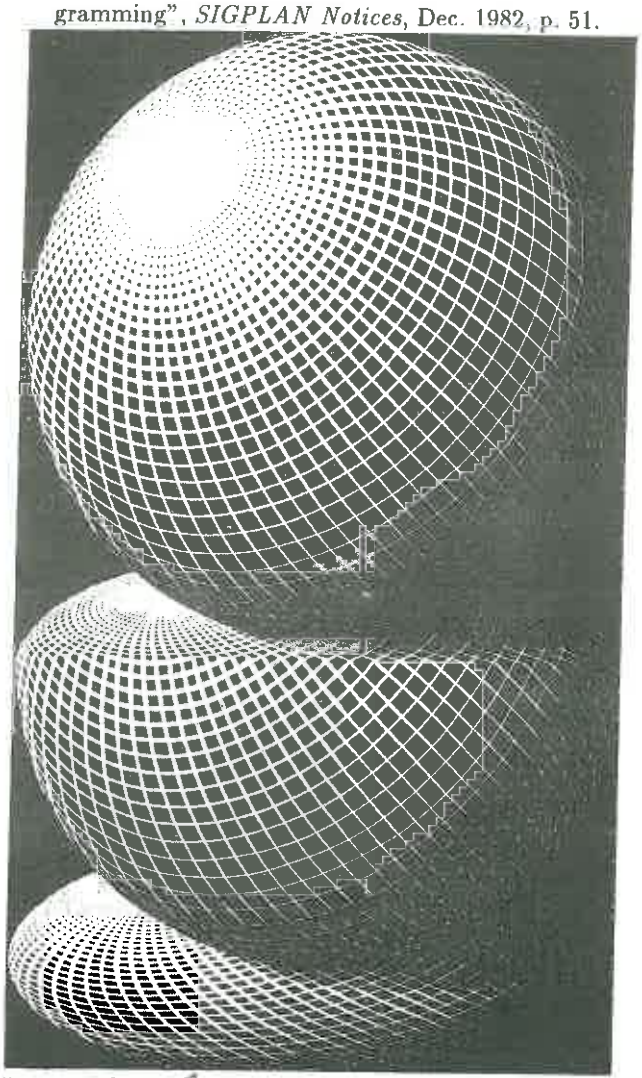
6 [Monarchi 92] Monarchi, David E., Puhr, Gretchen I., A Research Typology for Object-Oriented Analysis and Design, *Communications of the ACM*, September 1992, Vol. 35, No. 9, p. 36.

7 [Nerson 92] Nerson, Jean-Marc, "Applying Object-Oriented Analysis and Design", *Communications of the ACM*, Vol. 35, No. 9, September 1992.

8 [Palvia 93] Palvia, Prashant, Nosek, John, "A Field Examination of System Life Cycle Techniques and Methodologies", *Information and Management*, 25 (No. 2), 1993, pp. 73-84.

9 [Poulin 93] Poulin, J.S., et al., The Business Case for Software Reuse, *IBM Systems Journal*, Vol 32, No. 4, 1993, pp. 567-595.

10 [Rentsch 82] Rentsch, T., "Object-Oriented Programming", *SIGPLAN Notices*, Dec. 1982, p. 51.



تاریخچهٔ مختصر اِسمال‌تاک*

خلاصه‌شده توسط
رامین عرفانیان

مقدمه

برنامه‌سازی به زبانها را می‌توان به چند روش طبقه‌بندی کرد: امری[1]، کاربردی، براساس منطق، مسأله‌گرا[2] و غیره. اما به‌نظر می‌رسد که همگی آنها یا «ترکیبی از ویژگیهای مجزا» باشند و یا «تبلور شبک». کوبول، پی‌ال‌وان، اِیدا و غیره به‌نوع اول تعلق دارند و لیسپ و اِسمال‌تاک از نوع دوم هستند. شاید تصادفی نباشد که زبانهای نوع اول همگی توسط کمیته‌ها به‌وجود آمده‌اند و زبانهای نوع دوم به‌وسیلهٔ اشخاص منفرد.

اِسمال‌تاک اولین زبان شیءگراست که شیوهٔ برنامه‌سازی را یکبار و برای همیشه تغییر داد. این زبان شامل ایده‌هایی حیاتی مانند محیط مجتمع و سلسله مراتب ردهٔ جهانی[3] می‌باشد که این ایده در همین اواخر در زبانهای دیگر نیز ظاهر شده است. رهیافت انقلابی این زبان به جنبه‌های مختلف برنامه‌سازی ایجاب می‌کند که این واقعهٔ شگفت‌انگیز در دنیای کامپیوتر را بهتر درک کنیم.

طراحی ـ و وجود ـ زبان اِسمال‌تاک در نتیجهٔ این آگاهی است که هر چیزی را که بتوان توصیف کرد، می‌توان به‌وسیلهٔ ترکیب بازگشتی[4] یک نوع از بلوک ساختمانی ارائه کرد که ترکیب حالت و پردازشش را در خود پنهان می‌کند و فقط می‌توان از طریق مبادلهٔ پیامها با آن کار کرد. در اصطلاح کامپیوتری، اِسمال‌تاک بازگشتی روی خودِ مفهوم کامپیوتر است. به‌جای تقسیم کامپیوتر به مؤلفه‌هایی که هرکدام ضعیفتر از کل مجموعه هستند ـ مانند ساختمانهای داده‌ها، رویه‌ها و توابع که تقسیم‌بندی معمول زبانهای برنامه‌سازی هستند ـ هر شیء در اِسمال‌تاک بازگشتی است از تمام امکانات کامپیوتر. بنابراین مفهوم اِسمال‌تاک تقریباً شبیه داشتن هزاران کامپیوتر است که همگی به‌وسیلهٔ شبکه‌ای بسیار سریع به یکدیگر متصل شده‌اند.

آلن‌سی‌کی[5] متفکر و طراح اولیهٔ اِسمال‌تاک می‌گوید: «ایده‌های جدید برای پذیرفته شدن، مراحلی را هم از داخل و هم از خارج می‌گذرانند. از داخل، مراحل به این ترتیب است: از نگاهی گذرا و چند بار به یک الگو شروع می‌شود، سپس به ایده، بدون درک اهمیت وجودیش توجه می‌کنند، آنگاه آنرا در عمل و در چندین زمینه به‌کار می‌برند، سپس یک چرخش عمده در نظرشان پیش می‌آید و در نتیجه الگو مرکز شیوهٔ جدید تفکر می‌شود و سرانجام به همان نوع از قوانین غیر قابل انعطافی تبدیل می‌شود که در ابتدا از آن جدا شده بود. از خارج، همانطور که شوپنهاور ذکر کرد، ابتدا ایدهٔ جدید به‌عنوان یک کار معیوب مورد انتقاد قرار می‌گیرد، در طی چند سال آن روش، روشی روشن و عملی در نظر گرفته می‌شود و بالاخره منتقد اولیه ادعا خواهد کرد که خود آنرا اختراع کرده است.»

تاریخچه

تمام ماجرا از وقتی شروع شد که آلن کی برنامه‌سازی در نیروی هوایی بود و ایدهٔ موجود روی باروز ۲۲۰ را به‌صورت شیوه‌ای برای انتقال پرونده‌ها از یک ستاد فرماندهی آموزش هوایی به ستاد دیگر مشاهده کرد. هیچگونه سیستم‌عامل و یا قالبهای[6] پروندهٔ استانداردی وجود نداشت. بنابراین یک طراح (که تا امروز ناشناخته است) تصمیم گرفت با زیرکی این مشکل را برطرف‌کند. مشکل به این صورت حل شد که هر پرونده به سه قسمت تقسیم گردید. قسمت سوم کلاً رکوردهای داده‌ای واقعی بود که اندازه و قالب دلخواه داشتند.

قسمت دوم حاوی رویه‌های باروز ۲۲۰ بود که می‌دانست چگونه رکوردها و حوزه‌ها را برای کپی کردن بگیرد و قسمت سوم را بهنگام درآورد. و قسمت اول، آرایه‌ای از اشاره‌گرهای نسبی به نقاط ورودی رویه‌ها در قسمت دوم بود. دومین ایدهٔ گذرا درست کمی بعد به فکر آلن رسید، وقتی که در ستاد فرماندهی آموزش هوایی تصمیم گرفته شد باروز ۵۰۰۰ جایگزین ۲۲۰ گردد: سیستم حافظهٔ قطعه‌بندی شدهٔ آن، کارایی کامپایل کردن زبانهای سطح بالا در آن، مکانیزم خودکار آن برای فراخوانی زیر روال و راهگزینی روی چند برنامه، محض برای اشتراک و مکانیزم حفاظت آن. اما موفقیت بزرگ او روی این ماشین، در آن زمان، ایدهٔ OOP نبود، بلکه بعضی اطلاعات و آگاهیهایی بود دربارهٔ ترجمه و ارزیابی زبانهای سطح بالا. بعد از مدتی آلن در پائیز سال ۱۹۶۶ از دانشگاه یوتا فارغ‌التحصیل شد. در درس زیست‌شناسی تمرکز او بر متابولیسم سلول و نیز رشد جنینی ساختمان ارگانیسمهایی با اندازهٔ بزرگتر و علائم مکانیسم سادهٔ کنترل فرآیندهای پیچیده و نوعی از بلوک ساختمانی که قادر بود بین تمام بلوکهای ساختمانی مورد نیاز تفاوت قائل شود، متمرکز بود. سیستم پروندهٔ باروز ۲۲۰، باروز ۵۰۰۰، و سرانجام زبان سیمولا، همه از همین ایده برای اهداف متفاوتی استفاده کردند.

باب بارتون طراح اصلی باروز ۵۰۰۰ در یکی از سخنرانیهایش گفته بود که: «اصل اساسی در طراحی بازگشتی این است که مؤلفه‌ها همان قدرت مجموعهٔ کامل را داشته باشند.» این مطلب آلن را متعجب کرد که چرا

*) این مقاله خلاصه‌ای است از مقالهٔ آلن کی(Alan Kay) که در سال ۱۹۹۲ در مجلهٔ CACM به چاپ رسیده است.

1) Imperative 2) Problem-Oriented 3) Universal class hierarchy 4) recursive composition 5) Allen C. Kay

6) formats

همه می‌خواستند کامپیوتر را به‌عنوان یک کل، به مؤلفه‌های ضعیفتری که ساختمانهای داده‌ها و رویه‌ها نامیده می‌شدند، تقسیم کنند.

در سال ۱۹۶۷، آلن با اِد چیدل آشنا شد. یک نابغهٔ سخت‌افزار که در یک شرکت محلی بر روی یک «ماشین کوچک» کار می‌کرد. این ماشین اولین کامپیوتر شخصی نبود اما اِد آنرا برای متخصصان غیرکامپیوتری طراحی می‌کرد و مخصوصاً این که می‌خواست آنرا به زبان سطح بالاتری مانند بیسیک برنامه‌ریزی کند. اگر چه سرانجام به پیشنهاد آلن زبان جاس[7] انتخاب شد. آلن و اِد همکاری صمیمانه‌ای را شروع کردند و محصول آن را ماشین فلکس[8] نامیدند. آنها همانطورکه در طراحی پیش می‌رفتند، می‌خواستند طرحشان را به‌صورت پویا شبیه‌سازی کرده و بسط‌دهند ولی هیچکدام از زبانهای جاس و بیسیک (یا هر زبان موجودی که آنها می‌دانستند) مناسب این کار خاص نبود. آنچه که زبان فلکس نامیده شد لازم بود از نظر معنی، بیشتر تحت تأثیر سیمولا باشد تا زبانهایی مانند الگول یا اویلر[9]. ماشین فلکس برای سیستم NLS[10] که بخشی از ایده‌های جالب داگ انگلبرت بود، بسیار کوچک بود.

در سال ۱۹۶۸، آلن سیستم واقعاً قشنگی را دید. این سیستم گِریل[11] بود، نسخهٔ گرافیکی زبان جاس. اولین فهرست (فهرست مشهور RAND) به‌وسیلهٔ تام اِلیس به این منظور اختراع شده بود که اشارات و حرکات انسان را ثبت کند، وگیب گِرونر برنامه‌ای برای تشخیص کارآمد و پاسخ به آنها نوشت. گرچه همه چیز به‌شدت وصله‌پینه‌ای بود و سیستم اغلب خراب می‌شد ولی آلن هیچگاه اولین برخورد خود را با آن فراموش نکرد. در این هنگام آلن متوجه شد که واسط فلکس کاملاً اشتباه است. اما چگونه امکان داشت چیزی به بزرگی گِریل را به چنین ماشین کوچکی خوراند.

چند ماه بعد آلن با تیمی ملاقات کرد که لوگو را طراحی کرده بودند و آنرا به بچه‌های مدرسه آموزش می‌دادند. در آن زمان، سیمولا در OOP پیشقدم بود. اکنون آمیخته‌ای از ماشین فلکس، صفحه‌نمایش مسطح، گِریل، «گفتگوی ارتباطی»[12]، بارتون، ایده‌های مک‌لوهان و کار پاپرت با بچه‌ها، همگی برای شکل دادن به آنچه که کامپیوتر شخصی واقعاً باید باشد، در ذهن او جمع شده بودند.

در پائیز ۱۹۶۹، آلن سخنرانی جالبی را از باتلر لَمپسون دربارهٔ CAL-TSS شنید. CAL-TSS سیستم‌عامل پرتوانی بود که خیلی «شیءگرا» به‌نظر می‌رسید. اشاره‌گرها به‌همراه نقابهای بیتی[13]، دستیابی به عملیات داخلی «شیء» را محدود می‌کردند. این سیستم، نظر آلن را در مورد «شیءبه‌عنوان سرویس‌دهنده» تأئید کرد. همچنین رهیافت بسیار زیبایی در مورد مدیریت حالات استثنایی[14] نیز وجود داشت که روشی را که اغلب در سیستمهای تطابق الگو[15] برای پردازش خطاها به‌کار می‌رفت به یاد آلن آورد. تنها مسأله‌ای که طراحان CAL اصلاً به‌عنوان مسأله به آن نگاه نکرده بودند این بود که فقط چیزهای معینی (معمولاً بزرگ و کند) به‌عنوان شیء‌در نظر گرفته شده بودند. چیزهای سریع و کوچک و غیره، شیء‌نبودند. این مسأله نیاز به‌اصلاح داشت.

بعداً در آن سال آلن با زبان لیسپ به‌طورکامل آشنا شد. زیبایی این زبان در وهلهٔ اول برای او باورکردنی نبود. البته او دریافت که هرچند زبان محض قرار بود بر پایهٔ توابع باشد اما مهمترین مؤلفه‌های آن مانند عبارتهای لاندا[16]، نقل‌قولها و شرطها، اصلاً تابع نبودند و در عوض شکلهای خاص نامیده می‌شدند.

لندین[17] و دیگران توانسته بودند نقل‌قولها و شرطها را برحسب لاندا با شگردهای هوشمندانه و مفید به‌دست آورند، اما هنوز نقص وجود داشت. در زبانهای عملی وضع بهتر بود. در آن زبانها نه تنها EXPRها (که شناسه‌هایشان را ارزیابی می‌کردند) بلکه FEXPRها (که شناسه‌هایشان را ارزیابی نمی‌کردند) نیز وجود داشتند. سؤال بعدی آلن این بود، چرا اصلاً آنرا زبان تابعی می‌نامند؟ او هیچگاه پاسخ خوبی برای سؤالش نیافت. اما وقتی زمان اختراع اِسمال‌تاک فرا رسید این سؤال خیلی مفید بود، زیرا این سؤال یک خط فکری را آغازکرد که می‌گفت «سختترین و عمیقترین چیزی را که نیاز به انجام آن دارید در نظر بگیرید، سپس آن را تا می‌توانید بزرگ جلوه دهید و آنگاه چیز خلاصه‌تر و ساده‌تری را انجام دهید.»

زیراکس

در جولای سال ۱۹۷۰، زیراکس با اصرار رئیس دانشمندش جک گلدمن تصمیم گرفت مرکز پژوهشهای گسترده‌ای را در پالوآلتو در کالیفرنیا به‌وجود آورد. در ماه سپتامبر، جرج پیک، رئیس سابق دانشگاه واشنگتن (جایی که پروژهٔ آرپا متعلق به وس‌کلارک در آنجا اجرا شده بود) از باب تیلور (که پروژه آرپا را ترک کرده بود و در یوتا در تعطیلات به‌سر می‌برد) دعوت کرد آزمایشگاه علوم کامپیوتر آنجا را راه‌اندازی کند. باب تیلور از پالوآلتو دیدن کرد و تمام شب دربارهٔ آن صحبت می‌کردند. او آلن را به همکاری فراخواند و آلن از او تقاضای راهنمایی کرد. او به آلن گفت هر چه صلاح می‌دانی انجام بده. آلن بلافاصله شروع به کار روی نسخهٔ جدیدی از کی‌دی‌کامپ[18] کرد که در طراحی واسط کاربر در کامپیوترهای دفترچه‌ای به‌کار می‌رفت.

در اوایل سال ۱۹۷۱، باب تیلور با جذب اکثر اعضای فعال شرکت کامپیوتری برکلی گروه بسیار بزرگی را برای کار در پارک[19] به‌وجود آورد. این گروه شامل باتلر لمپسون، چاک تکر، پیتر دویچ و بسیاری دیگر بود. کمی بعد از آن بسیاری از افراد داگلاس انگلبرت نیز به این گروه پیوستند. بخشی از دلایل آنها این بود که می‌خواستند NLS را به‌عنوان یک سیستم توزیع شده دوباره پیاده‌سازی کنند و انگلبرت می‌خواست سیستم اشتراک زمانی را باقی

7) Joss 8) Flex 9) Euler 10) oNLine System
11) Grail 12) Communication talk 13) bit-masks
14) exception handling 15) pattern matching
16) lambda 17) Landin 18) KIddiKomp 19) PARC-Paolo ALto Research Center

نگهدارد. این گروه شامل بیل انگلیش (عضوی از گروه مخترعان موش[20])، جفری رالیفسون و بیل پیکستون بود.

در تابستان سال ۱۹۷۱ آلن دوباره ایدهٔ کی‌دی کامپ را با طراحی دقیقتری تعریف کرده و آن را مینی‌کام[21] نامید. او ایدهٔ برش بیتی را شبیه سیستم NOVA 1200 به‌کار برد، یک صفحه‌نمایش نگاشت بیتی داشت، یک وسیلهٔ اشاره‌گر داشت، انتخابی برای حافظهٔ جانبی (در حقیقت حافظهٔ سوم) ایجاد شده بود و زبانی که آلن آنرا اِسمال‌تاک نامید. سیستمها زئوس، اودین و ثور نامیده می‌شد و همه چیز به‌سختی انجام می‌گرفت. آلن تصور کرد که اِسمال‌تاک آنقدر نام حقیری است که اگر کار زیبایی انجام دهد مردم را شگفت‌زده خواهد کرد.

20) Mouse 21) MiniComp

منتشر شد

گزارش کامپیوتر

سال پانزدهم شماره‌های ۱ تا ۶

(شماره‌های پیاپی ۱۱۸ تا ۱۲۳)

۱۳۷۲

انتشارات انجمن انفورماتیک ایران

بهای سالنامه برای اعضای انجمن ۵۰۰۰ ریال و برای دیگران ۸۰۰۰ ریال است.

برای تهیهٔ سالنامه با دفتر انجمن تلفن ۶۵۶۶۸۷ و یا به نشانی انجمن صندوق پستی ۱۴۱۵۵-۱۱۹۶ تهران تماس بگیرید.

قابل توجه

شرکتهای کامپیوتری

انجمن انفورماتیک ایران، نشانی پستی اعضای خود را برای ارسال بروشورها و آگهیهای تبلیغاتی در اختیار می‌گذارد.

شرکتهای کامپیوتری می‌توانند برای تبلیغ محصولات خود، از لیست نشانیهای پستی اعضای انجمن استفاده کنند.

آگهیهای تبلیغاتی شما

در کوتاهترین زمان

به‌دست

مناسبترین مخاطبان

(مدیران، کارشناسان و متخصصان کامپیوتر در سراسر کشور)

خواهد رسید.

برای کسب اطلاعات بیشتر و آگاهی از چگونگی استفاده از لیست نشانیهای پستی انجمن در ساعات اداری با دفتر انجمن (تلفن ۶۵۶۶۸۷) تماس بگیرید.

خلاصه عملکرد گروه شیءگرایی

به اطلاع علاقه‌مندان شیءگرا می‌رساند که زیرگروه تخصصی شیءگرایی تحت نظر انجمن انفورماتیک ایران و به سرپرستی آقایان علی ارسنجانی و رامین عرفانیان تاکنون سه جلسه ماهانهٔ خود را با حضور متوسط ۳۸ نفر برگزار نموده است.

از اهداف این گروه بالا بردن سطح دانش و اشاعه علم شیءگرایی را می‌توان بیان کرد که به‌دنبال اجرای این اهداف تاکنون دو گردهمایی داشته است.

ریز اهداف این گروه به‌شرح زیر می‌باشد:

۱. ارتقاء آگاهی تخصصی کارشناسان وعلاقمندان به:

۱ـ۱. به‌کارگیری (عملی): مزایا و مشکلات به‌کارگیری تکنولوژی شیءگرا؛ استفاده صحیح از این تکنولوژی و اجتناب از به‌کارگیری ناصحیح آن.

۱ـ۱ـ۱. فراهم نمودن تسهیلاتی برای معرفی مفاهیم، تکنیکها، ابزار، متدولوژیهای شیءگرا و سطوح مبتدی، متوسط، پیشرفته، چنانکه توسط کاربران، کارشناسان و پژوهشگران شیءگرایی استفاده گشته و توصیه می‌گردد.

۱ـ۱ـ۱. ترویج به‌کارگیری متدولوژیها، تکنیکها، محیطها و زبانهای شیءگرا به‌عنوان وسیله‌ای برای ارتقاء سطح کیفی نرم‌افزارهای تولید شده در ایران.

۱ـ۲. شناخت مفاهیم (نظری): ترویج مفاهیم و معانی رهیافت شیءگرایی.

۱ـ۳. ایجاد روند کانالیزه شده و هدفمند برای ترویج و اشاعه فرهنگ شناخت و به‌کارگیری صحیح شیءگرایی از دید مهندسی نرم‌افزار.

۲. فراهم نمودن زمینه مناسبی برای تبادل نظر و تجربیات کارشناسان و علاقمندان:

۲ـ۱. ایجاد انگیزه‌ای برای ارتباط علمی میان کارشناسان و علاقمندان در جهت افزایش میزان کاربرد این تکنولوژی نرم‌افزاری در سطحی وسیعتر.

۲ـ۲. ایجاد انگیزه و زمینه برای خلاقیت و نوآوری در به‌کارگیری مفاهیم و ابزار، توسعه سیستمهای نرم‌افزاری براساس الگوی مفهومی (Paradigm) شیءگرا، تشویق پژوهش در مفاهیم اساسی روشهای مبنایی شیءگرایی، تشویق تدریس مفاهیم و کاربردهای عملی.

۳. ایجاد رابطه با سازمانهای بین‌المللی ذیصلاح و دست‌اندرکار مسائل شیءگرایی.

خلاصه‌ای از صورت جلسهٔ سه گردهمایی انجام‌شده به‌شرح زیر می‌باشد:

اولین گردهمایی مورخ ۷۳/۹/۲

۱ـ اهداف کلی گروه به‌طور مفصل و مشروح به‌اطلاع شرکت‌کنندگان رسید.

۲ـ مسئولیتهای موجود در گروه دسته‌بندی و تفکیک گردید.

۳ـ شرکت‌کنندگان نقطه‌نظرهای خود را در مورد چگونگی تشکیل جلسات آینده بیان کردند.

۴ـ سخنرانی کوتاهی پیرامون علم شیءگرایی و فواید آن ایراد گردید.

۵ـ پرسشنامه‌ای جهت پاسخگویی در اختیار حاضرین قرار گرفت.

دومین گردهمایی مورخ ۷۳/۱۰/۷

۱ـ نکات باقی‌مانده از جلسه اول به‌صورت اختصار بررسی گردید.

۲ـ پرسشنامه‌هایی که توسط شرکت‌کنندگان پاسخ داده شده بود مورد تحلیل قرار گرفته و به‌اطلاع ایشان رسانده شد.

۳ـ لیستی از تمامی نکات مهم در علم شیءگرایی به شرکت‌کنندگان داده شد.

۴ـ تعدادی از نکات لیست مذکور تحلیل و بررسی شد.

۵ـ یک بسته آموزشی شیءگرایی به‌صورت سخنرانی ارائه گشت.

سومین گردهمایی مورخ ۷۳/۱۱/۵

۱ـ یک سخنرانی پیرامون یکی از روشهای نمایش‌دهی طراحی در متدولوژی شیءگرای Object Modelling Technique ایراد شد.

۲ـ آئین‌نامهٔ زیرگروههای انجمن انفورماتیک خوانده و بررسی شد.

۳ـ تعدادی از موارد لیست نکات مهم در شیءگرایی تحلیل و بررسی شد.

۴ـ بحث و تبادل نظر پیرامون موارد فوق انجام شد.

۵ـ هیئت اجرائی و کمیتهٔ علمی زیرگروه انتخاب شدند.

امید آن داریم بر تعداد علاقمندان این گروه روز به روز افزوده گردد.

علی ارسنجانی / رامین عرفانیان

یک متدولوژی پیشنهادی برای تولید شیءگرا

دکتر محسن شریفی

آزمایشگاه مهندسی نرم‌افزار
دانشکده مهندسی کامپیوتر دانشگاه علم و صنعت ایران
email: iustech@earn.bitnet

چکیده

نگرش شیءگرایی در مهندسی نرم‌افزار به‌دوران بلوغ خود نزدیک شده و متدولوژیهای تولید در صنایع کامپیوتر در تلاش برای روشن نمودن و ارتقاء مبانی این نگرش برای تولید سیستمهای نرم‌افزاری هستند. در مقایسه با سایر روشهای تولید در مهندسی نرم‌افزار، نگرش شیءگرایی پتانسیل افزایش سازگاری فعّالیتهای فرارونـد تولید را دارا می‌باشد. البته اغلب تحقیقات و کارهای انجام شده در زمینهٔ این نگرش، فقط به‌فاز خاصّی از تولید توجّه نموده و گذار بین فازهای مختلف تولید را تحت این نگرش مورد توجّه قرار نداده‌اند. متدولوژی ارائه شده در این مقاله، کلّ چرخهٔ حیات تولید را در نظرگرفته است. سعی شده است تا تکنیکها و سیستمهای علامت گذاری متعدّد شیءگرایی در قالبی سازگار مطرح و ارائه شوند. از متدولوژی ارائه شده به‌عنوان چارچوبی برای استفاده از تکنیکهای شیءگرایی در تولید یک محیط شبیه‌سازی و نمونه‌سازی استفاده خواهد شد. مفاهیم پیچیدهٔ یک سیستم نمونه با استفاده از این متدولوژی نمونه‌سازی شده‌اند.

مقدّمه

تولید و نگهداری سیستمهای نرم‌افزاری همچنان مبارز می‌طلبد. برخلاف مهندسی سخت‌افزار که بعضاً تکنیکهای اثبات شدهٔ رسمی در آن با موفقیت به‌کارگرفته شده‌اند، مهندسی نرم‌افزار همچنان با تعدادی تکنیکهای پیشنهادی (وظیفه‌مندی[1]، ساختیافته[2]، شیءگرا ...) دست به گریبان است. هرکدام از این تکنیکها مدّعی یک متدولوژی سیستماتیک برای فرارونـد تولید نرم‌افزار هستند. ولی براساس کاربردهای عملی آنها، هر مجموعه از تکنیکها معایب و محاسنی در رابطه با فرارونـد تولید نرم‌افزار بروز داده‌اند. تکنیک شیءگرایی دارای این پتانسیل است که روشی سازگارتر برای فرارونـد تولید نرم‌افزار فراهم کند. این تکنیک از نقاط قوّت تکنیکهای سنّتی استفاده نموده و بر مفاهیمی چون **انتزاع داده‌ها**[3]، **دربرگیرندگی**[4]، **پنهان‌سازی اطّلاعات**[5]، **وراثت**[6]، **چندریختی**[7] و **باز به‌کارگیری**[8] تأکید دارد. با چنین پایه‌هایی، روش شیءگرایی‌می‌تواند مدلی سازگار برای تولید فراهم نماید که بتواند فرارونـد تولید نرم‌افزار رارسمیّت بخشد.

در تلاشی جدّی برای روشن کردن و ارتقاء تکنیکهای شیءگرایی، متدولوژیستهای صنعت کامپیوتر تکنیکها و سیستمهای نشانگذاری متعدّدی را برای انجام دادن فازهای تحلیل، طرّاحی و پیاده‌سازی فرارونـد تولید نرم‌افزار توصیه می‌کنند. این تعدّد سبب شده است که مهندسین نرم‌افزار در انتخاب و به‌کارگیری مجموعه‌ای سازگار از مراحل کاری برای چرخهٔ حیات تولید شیءگرا، با مشکل روبرو باشند. در یک بررسی به‌عمل آمده از تکنیکهای موجود شیءگرایی مشخّص شده است که هیچیک از این تکنیکها کلّ چرخهٔ حیات تولید نرم‌افزار را به‌طور سازگار پوشش نمی‌دهند ـ البته این بررسی [1]، سیستمهای نشانگذاری مفیدی (همچون تکنیکهای ارائه شده توسط [2]، [3]، [4] و [5]) را مشخّص نموده است. تکنیکهای به‌کار گرفته شده توسط McMenamin & Palmer [6] نیز، اگر چه کاملاً شیءگرا نیستند، ولی در متدولوژی ارائه شده در این مقاله تأثیر داشته‌اند. هدف این مقاله اینست که نشان دهد که این تکنیکها و سیستمهای نشانگذاری می‌توانند به‌طریقه‌ای جدید تلفیق شده و روشی سازگار برای فزارونـد تولید نرم‌افزار به‌دست دهند.

انگیزهٔ اصلی برای پیشنهاد این متدولوژی جدید تولید شیءگرا، نیاز مبرم به‌کاهش خطرات **تولید شیءگرای سیستمهای کلان**[9] **نرم‌افزاری** (بالاخص، برای محیطهای نمونه‌سازی و شبیه‌سازی) می‌باشد. این خطرات عبارتند از: (۱) فقدان راهنمائیهای واضح و سازگار برای اجرای هر یک از فازهای چرخهٔ حیات تولید شیءگرای نرم‌افزار، (۲) فقدان راهنمائیهایی برای گذار بین این فازها و نشان دادن ردّگذارهای بین فازها، (۳) فقدان ابزارهای خودکار[10] برای پشتیبانی از هر فاز، و (۴) فقدان تجربهٔ عملی ما در اعمال تکنیکهای شیءگرایی موجود بفرارونـد تولید نرم‌افزارهای کلان. متدولوژی شیءگرا بدین دلیل جهت تولید اینگونه نرم‌افزارها انتخاب و به‌کار گرفته شده است که در

1) Functional 2) Structured 3) Data Abstraction
4) Encapsulation 5) Information Hiding 6) Inheritance
7) Polymorphism 8) Reusability

۹) سیستمهای نرم‌افزاری کلان سیستمهایی هستند که از تمامی این ابعاد بزرگ باشند: (۱) اندازهٔ سیستم، (۲) گسترهٔ زمانی حیات سیستم، (۳) تعداد نفرات درگیر با سیستم در دورهٔ حیات سیستم، و (۴) زیربنای تکنیکی و آموزشی در دورهٔ حیات سیستم.

10) CASE Tools

- Coprocessor
- RAM Module
- DRAMs
- SRAMs
- CPU

Processor Upgrades

SX / Now!

SLC/Now!

شرکت مهندسی ایمان تک

نماینده انحصاری

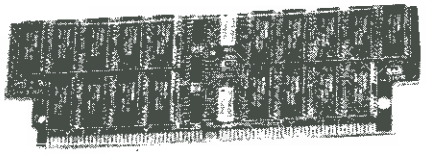
MANUFACTURED BY

Kingston TECHNOLOGY CORPORATION

در ایران

پرفروش‌ترین تولیدکنندهٔ CPU Upgrades

و حافظه در دنیا با پنج سال گارانتی



تهران - ایران - صندوق پستی ۶۶۵-۱۹۷۶۵

خیابان طالقانی، تقاطع ولیعصر، پلاک ۲، طبقه اول، شماره ٤

تلکس: ۲۱۳٤۳۲ فکس: ٦٤٠٧٥١٧

تلفن: ٦٤١٥١٥٦ - ٦٤١٥١٥٥ - ٦٤٠٠٨٧١ - ٦٤٠٩٥٦١

EPSON

LQ 2550 LQ 1060
LQ 1170 LQ 1070
LQ 150 LQ 100

یکسال گارانتی

کامپیوترریز مرکز توزیع چاپگرهای
EPSON

آدرس: تهران - خیابان خردمند شمالی - شماره ۱۳۵ - طبقه هفتم
تلفن: ۸۸۲۰۳۹۱ - ۸۸۲۴۹۰۶ - ۸۳۵۱۵۵
۸۸۳۵۸۹۸ - ۸۸۲۱۴۲۳
فاکس: ۸۸۳۷۰۴۳

سرویس و تعمیرات چاپگر

واحد تعمیرات شرکت کامپیوترریز از اول مهر ماه ۷۳ آماده سرویس دهی به کلیه شرکتهای کامپیوتری و دیگر موسسات می باشد.

نگارش ۶/۰

جدید

نرم‌افزار انتشاراتی

قوی‌ترین و آسان‌ترین!

- ویرایش همزمان چند پرونده
- رسم اشکال هندسی با استفاده از ماوس
- به‌صورت کاملا WYSIWYG
- محیط کاری شبه ویندوز
- سهولت بی‌سابقه یادگیری و کار
- متنوع‌ترین مجموعه قلمهای فارسی و لاتین
- مجموعه غنی از سمبلها و علائم مذهبی ملی و عمومی
- پیشرفته‌ترین امکانات واژه‌پردازی و چاپ با ساده‌ترین راه حل‌ها و مناسبترین هزینه

مؤسسهٔ انتشارات صابرین

خیابان دکتر بهشتی، خیابان میترا، کوچهٔ نکیسا، شمارهٔ ۵ کدپستی ۱۵۷۸۸ تلفن ۸۵۸۲۶۷

فراخواندن (احضارکردن) یک متد است. محدودیتهای نمایانی عمومی و خصوصی در اغلب زبانهای پیاده‌سازی وجود دارند.

نوع‌گذاری ضعیف **Weak Typing**

این ویژگی زبانهای پیاده‌سازی که نوع هر ارزش یا عبارت نمی‌بایستی صراحتاً اعلام شود. نوع‌گذاری ضعیف برنامه‌ساز یا سیستم زمان اجرا را ملزم می‌کند که خود نسبت به دستکاری صحیح ارزشها در کاربرد اطمینان حاصل کند.

نوع **Type (syn. Object Type)**

گزاره‌ای که صفتها و رفتار مشترک مجموعه‌ای از اشیاء را توضیح می‌دهد.

تعریف نوع **Type Definition**

معنی یک نوع ـ مشخصاتی از رفتار و صفتها که نمونه‌های یک نوع را به‌طور اختصاصی توصیف می‌کنند.

فاصل نوع **Type Interface**

مجموعه‌ای از درخواستها که نمونه‌های نوع می‌توانند به آنها پاسخ دهند.

شیءنوع **Type Object**

موجودیتی که هم یک شیءو هم یک نوع است.

محدودیت یگانگی **Uniqueness Constraint**

محدودیتی که خصیصه‌ای را به‌طور منحصر به‌فرد برای نمونه‌های یک نوع مشخص می‌کند. به‌عنوان مثال، نمونه‌های نوع «حساب بانکی» می‌توانند بدیتگونه محدود شوند که حتماً دارای یک ارزش منحصر به‌فرد برای صفت «شمارهٔ حساب» باشند.

استفاده **Use**

رابطه‌ای بین موجودیتها که نشان‌دهندهٔ این است که یک موجودیت (به‌نام مشتری) از سرویسهای موجودیت دیگری (به‌نام خدمتکار) استفاده می‌کند.

مقدار، ارزش **Value**

یک شیءیا داده. اشاره‌گرها و دستگیره‌ها می‌توانند به اشیاء، و نویسه‌های لفظی به داده‌ها، اشاره کنند. یک مقدار می‌تواند به‌عنوان پارامتر واقعی در یک درخواست به‌کار رود.

عمل وابسته به ارزش **Value-Dependent Operation**

عملی که واکنش آن به هویت اشیاء ردّ شده به‌عنوان پارامترها بستگی دارد.

تابع عضو مجازی **Virtual Member Function**

یک عمل در زبان C++ که تقیّد پویا به آن اعمال می‌شود.

نمایانی **Visibility**

دستیابی‌پذیری به صفتها و متدهای یک رده. قابل دستیابی به معنای توانایی بازگرداندن یک ارزش نگهداری شده توسط یک صفت یا

درخواست همگام **Synchronous Request**

یک درخواست مشتری که در آن مشتری در انتظار وصول جواب درخواست می‌ماند.

نحو **Syntax**

نحو یک رده در واقع ساختارهای داده‌ها و دستورالعملهایی از زبان هستند که رده را پیاده‌سازی می‌کنند.

شیءمقصد **Target Object**

یک شیءکه از آن تقاضای پاسخ به یک درخواست شده است. شیءمقصد معمولاً اولین نشانوند در فراخوانی (احضار) یک عمل است.

شیءگذرا **Transient Object**

یک شیءکه دورهٔ حیاتش به فراروندی که آن را خلق (ایجاد)کرده وابسته است.

Transition Post-condition (syn. Event Post-condition)

پس‌شرط گذرا

حالت (وضعیت) یک شیءپس از یک رویداد از یک نوع خاص.

Transition Pre-condition (syn. Event Pre-condition)

پیش‌شرط گذرا

حالت (وضعیت) یک شیءقبل از یک رویداد از یک نوع خاص.

قانون راه‌اندازی **Trigger Rule**

یک رابطهٔ علت و معلولی بین یک نوع رویداد و یک عمل. یک قانون راه‌اندازی می‌تواند یک شرط کنترلی داشته باشد که بررسی کند که آیا درست است که عملی را فرا بخواند (احضارکند) و اینکه چه نشانوندهایی را برای عمل مهیا کند.

راه‌انداز **Trigger**

اتصال بین یک علت و معلول. یک راه‌انداز در واکنش نسبت به یک رویداد از نوع مناسب، موجب ارزشیابی یک شرط کنترلی شده، نشانوندهای موردنیاز را فراهم کرده، و عملی را که توسط قانون راه‌اندازی مربوطه‌اش مشخص شده است فرا می‌خواند (احضار می‌کند).

چندتایی **Tuple**

یک مجموعهٔ مرتب‌شدهٔ از ارزشها.

کاربر ایستای شیئی **Static Object-Based Application**

مجموعه‌ای از نوعها و رده‌های مرتبط با یک هدف خاص کاربر نهایی.

ردهٔ حافظه‌ای **Storage Class**

رده‌ای که دورهٔ حیات یک شیءرا در یک کاربرد معین می‌کند. چهار ردهٔ حافظه‌ای وجود دارند: خودکار، ایستا، آزاد، و پایدار. اشیاء خودکار به‌هنگام احضار یک متد خلق (ایجاد)شده و در زمان خروج از آن حذف می‌شوند. اشیاء ایستا به‌هنگام احضار برنامه خلق شده و تا انتهای برنامه وجود داشته و ارزشهای خود را نگه می‌دارند. اشیاء پایدار پس از اجرای یک برنامه نیز به موجودیت خود ادامه داده و اغلب داخل یک پایگاه داده‌ها یا مخزنی دیگر نگهداری می‌شوند.

Stream Library

مخزن جریان (مخزن رویه‌های ورودی و خروجی)

مجموعه‌ای از رده‌ها که رویه‌های ورودی و خروجی برای کلیهٔ نوعهای انتزاعی داده‌ها فراهم می‌کنند.

نوع‌گذاری قوی (شدید) **Strong Typing**

یک ویژگی زبانهای پیاده‌سازی که نوع هر مقدار یا عبارت بایستی صراحتاً اعلام شود. نوع‌گذاری قوی به کامپایلر اجازه می‌دهد تا از دستکاری صحیح مقادیر درکاربرد اطمینان حاصل کند.

زیررده **Subclass**

رده‌ای که صفتها و متدهای ردهٔ دیگری را به ارث می‌برد.

قابل جانشانی **Substitutability**

این ویژگی که یک متد تعریف شده برای یک نوع خاص بتواند به هر نمونه از نوع و زیرنوعهایش اعمال شود.

زیرنوع **Subtype**

نوع محدودتر و خاصتر در یک رابطهٔ تخصصی

Superclass (syn. Base Class, Parent Class)

بالارده (فوق رده)

رده‌ای که صفتها و متدهایش را ردهٔ دیگری به ارث می‌برد.

بالانوع (فوق نوع) **Supertype**

نوع انتزاعیتر و عامتر در یک رابطهٔ تخصصی.

وضعیت، حالت **State (syn. Object State)**

سازگاری حالت **State Consistency**

حصول اطمینان از اینکه حالت (وضعیت) شیءبا وضعیت مجاز در مشخصات شیءمطابقت دارد.

تمامیت حالت **State Integrity**

اطمینان از اینکه حالت (وضعیت) شیءنمی‌تواند توسط رویدادهای خارج از شیءخراب یا فاسد شود.

فضای حالت **State Space**

شمارش کلیهٔ حالتهای ممکن برای نمونه‌های یک نوع یا رده. فضای حالت توسط نام، نوع قلمرو و محدودیتهای بر روی قلمرو هر خاصیت (صفت) تعریف می‌شود.

گذار حالت **State Transition**

یک تغییر حالت برای یک شیء- چیزی که قابل علامت دادن توسط یک رویداد باشد.

نوع حالت **State Type**

یک انتزاع که قابل اعمال به مجموعه‌ای از حالتهای شیءباشد ـ شکل منطقی مجموعه‌ای از حالتهای شیءمشابه.

متغیر حالت **State Variable**

یک خصوصیت یا نوع که بخشی از یک حالت معین از یک نوع مشخص است.

درخواست تغییردهندهٔ حالت **State-Modifying Request**

درخواستی که ممکن است نتایج درخواستهای آتی را تغییر دهد.

تقیّد ایستا **Static Binding**

نسبت دادن یک نام به یک رده در زمان کامپایل کردن برنامه. در مورد درخواستها، متد خاص برای یک عمل در زمان کامپایل معلوم می‌شود. این تکنیک به کامپایلر اجازه می‌دهد که نوعهارا در زمان کامپایل کردن برنامه چک کرده و کد صحیح مقصد را برای برنامه ایجاد و تعبیه کند.

رده‌بندی ایستا **Static Classification**

این خصیصه که نوعهایی که بر یک شیءاعمال می‌شوند نمی‌توانند تغییر یابند. اغلب زبانهای شیءگرا پیاده‌سازی رده‌بندی ایستا را اجرا (تحمیل) می‌کنند.

موجودیتی که درخواست سرویس را صادر می‌کند به مشتری موسوم است.

سرویس **Service**

فراروندی که در پاسخ به یک درخواست می‌تواند انجام گیرد.

امضاء **Signature**

نام و لیست نوعهای نشانوند یک عمل یا متد.

وراثت منفرد **Single Inheritance**

یک مکانیزم زبان که اجازه می‌دهد تعریف یک رده شامل صفتها و متدهای تعریف شدهٔ حداکثر یک بالارده (فوق رده) باشد.

Specialization Relation (syn. Generalization Relation

رابطهٔ تخصصی

رابطه‌ای بین نوعها که نشانگر این است که ویژگیهای نوع عام، بخشی از تعریف یک نوع خاصتر هستند. در این رابطه، نوع عام بالانوع (فوق نوع) بوده و نوع خاص زیرنوع است. یک رابطهٔ تخصصی ایجاب می‌کند که هر نمونه از زیرنوع، نمونه‌ای از بالانوع نیز باشد ـ به‌عبارت دیگر، دنبالهٔ زیرنوع در دنبالهٔ بالانوع شامل شده است. مفهوم زیرنوع نیز بایستی با مفهوم بالانوع سازگار باشد. بنابراین، زیرنوع می‌تواند به صفتها و رفتار بالانوع علاوه کند، ولی نمی‌تواند ویژگیهای بالانوع را حذف کند یا با آنها متضاد باشد.

مشخصات **Specification**

توضیحی از یک مسئله یا موضوع که با یک سیستم محاسباتی یا غیرمحاسباتی پیاده‌سازی می‌شود. مشخصات شامل توضیحی از موضوع و همچنین جنبه‌هایی از پیاده‌سازی که در نمایش آن تأثیر دارند، می‌باشد. همچنین، فرارونده تحلیل و طراحی که به توضیحی از یک مسئله یا موضوع می‌انجامد که قابل پیاده‌سازی با یک سیستم محاسباتی یا غیرمحاسباتی است.

شیءخاص **Specific Object**

یک شیءکه بخشی از یک سرویس شیءرا برای فاصل شیءفراهم می‌کند. مقصود این است که تعداد محدودی از پیاده‌سازیهای این اشیاء، اغلب به‌صورت خدمتکاران، در سیستم وجود دارند.

درخواست **Request**

رویداد احضار یک عمل. درخواست شامل نام عمل و صفر یا بیشتر پارامتر واقعی است. مشتری درخواستی را صادر می‌کند تا سبب انجام سرویسی شود. ممکن است ارزشهایی که بایستی به مشتری برگردانده شوند نیز در درخواست شامل باشند. یک پیام می‌تواند برای پیاده‌سازی (حمل) درخواست و جوابها استفاده شود.

مسئولیت **Responsibility**

مجموعه‌ای از رفتارهای مشخص شده برای یک نوع. مسئولیتها مکانیزمی برای سازماندهی رفتارهای مختلف یک نوع از طریق مقصود یا کارکردی مشترک است.

نتیجه **Result**

اطلاعات بازگشتی به مشتری که می‌تواند شامل ارزشها و اطلاعات وضعیتی باشد ـ اطلاعات وضعیتی بیانگر شرایط استثنایی هستند که به‌هنگام اجرای سرویس درخواستی مشتری بروز کرده‌اند.

شِما **Schema**

یک نمایش رسمی با مجموعه‌ای معین از نمادها و قوانینی که حاکم بر شکل‌گیری یک نمایش با استفاده از نمادها است. انواع مختلفی از شِماها، مانند شِمای شیء، شِمای واقعه (رویداد) و شِمای فعالیت، وجود دارند.

قلمرو امنیتی **Security Domain**

زیرمجموعه‌ای از منابع محاسباتی مورد استفاده برای تعریف کردن سیاست امنیتی.

خودارجاعی **Self-Reference**

توانایی یک متد در مشخص کردن (تعیین) شیءمقصدی که برای آن متد فراخوانده (احضار) شده است. این مفهوم توسط کلمات کلیدی self در زبان Smalltalk و this در زبان C++ مورد استفاده قرار می‌گیرد.

معنی **Semantics**

معنی یک نوع همان تعریف (مقصود) نوع است.

خدمتکار، خادم **Server**

موجودیتی (مانند یک شیء، رده یا کاربرد) که نسبت به یک درخواست مشتری برای یک سرویس، از خود عکس‌العمل نشان می‌دهد.

CAD) باشد. «تعبیر کامپیوتری» مفهومی است که به نمونه‌های «طرح قطعه» اعمال می‌شود ـ هر طرحی برای یک قطعه، یک رسم الکترونیکی از طرح را به‌همراه دارد.

حفاظت‌شده **Protected**

یک محدودیت (visibility) که از دسترسی مشتریان یک رده (بجز دوستان و زیررده‌ها) به اعضاء رده جلوگیری می‌کند.

حفاظت **Protection**

توانایی محدود کردن مشتریانی که یک سرویس مورد درخواست آنها انجام خواهد گرفت.

عمومی **Public**

یک محدودیت (visibility) که به مشتریان یک رده اجازهٔ دستیابی به اعضاء رده را می‌دهد.

سؤال **Query**

انتخاب اشیاء براساس یک گزارهٔ خاص

نوع حوزه **Range Type**

نوع ارزشهایی که توسط یک رابطه به آنها نگاشت شده است

حوزه **Range**

مجموعه ارزشهایی که توسط یک رابطه به آنها نگاشت شده است

تمامیّت ارجاع **Referrential Integrity**

اطمینان از اینکه یک هویت شیء، مشخص‌کنندهٔ منحصر به‌فرد یک شیءاست.

رابطه **Relation**

نگاشتی بین نوعها. نوع قلمرو برای یک رابطهٔ خاص، نوع اشیائی است که نگاشت می‌تواند بر آنها اعمال شود. نوع حوزه، نوع ارزشی است که بازگردانده می‌شود. یک نمونه از نوع قلمرو می‌تواند از طریق رابطه با مجموعه‌ای از ارزشهای نوع حوزه مرتبط شود. همچنین، نوعی که دنباله (مصداق) آن مجموعه‌ای از چندتاییها است.

نوع درخواست **Request Type (syn. Signature)**

الگوی احضار (فراخوانی) یک عمل، شامل نام عمل، نوعهای نشانوندها و نوع نتیجهٔ بازگشتی.

مختلفی به حسب نوع شیءمقصد گردد، می‌دهد. اغلب زبانهای شیءگرا از مکانیزم انتخاب متد مناسب براساس ردهٔ شیء(چندریختی رده‌ای) پشتیبانی می‌کنند. تعداد اندکی از زبانها یا سیستمها از مکانیزم انتخاب متد مناسب براساس سایر ویژگیهای شیءمانند ارزشهای شیءو چیزهای تعریف شده توسط کاربر (چندریختی عمومیت یافته)، پشتیبانی می‌کنند.

نیرو نوع **Powertype**

نوعی که نمونه‌هایش دقیقاً زیرنوعهای نوع دیگری هستند. هر نمونه یک «نوع به‌عنوان شیء» است. به‌عنوان مثال، نمونه‌های «مدل اتومبیل»، زیرنوعهای «اتومبیل» هستند. یعنی اینکه «پیکان»، «بنز» و «مزدا» نمونه‌هایی از «مدل اتومبیل» و در عین حال زیرنوعهای «اتومبیل» هستند.

پس‌شرط **Post-condition**

محدودیتی که بایستی پس از ختم یک عمل برقرار باشد.

پیش‌شرط **Pre-condition**

محدودیتی که بایستی قبل از احضار (فراخوانی) یک عمل برقرار باشد.

فاصل اصلی **Principal Interface**

مجموعه درخواستهایی با معنی برای یک شیء.

خصوصی **Private**

یک محدودیت نمایانی که از دستیابی مشتریان یک رده (بجز دوستان رده) به اعضاء رده جلوگیری می‌کند.

فراروند **Process**

یک توالی رفتار.

محصول **Product**

نتیجهٔ یک فعالیت ـ یک چیز ارزشمند. یک گزارش، یک برنامهٔ نرم‌افزار و یک طرح از یک سیستم، نمونه‌هایی از محصول هستند.

خصیصه، خاصیت **Property**

ایده‌ای که در مفهوم یک نوع شامل و نهفته شده است ـبنابراین یک خصیصه مفهومی است که به نمونه‌های نوع اعمال می‌شود. خصیه‌ها معمولاً برای مشخص کردن یک نگاشت بین نوعها استفاده می‌شوند. به‌عنوان مثال، نوع «طرح قطعه» ممکن است دارای خصیصهٔ «تعبیر کامپیوتری» (یک رسم الکترونیکی از طرح قطعه توسط یک سیستم

شرکت کردن، دخالت کردن **Participate**

یک شیءزمانی در یک درخواست شرکت می‌کند که یک یا بیشتر از پارامترهای واقعی درخواست به شیءاشاره داشته باشد.

افراز، جزء، بخش **Partition**

نتیجه یا کنش تقسیم کردن یک نوع به زیرنوعها منفصل از یکدیگر. یک افراز در واقع یک ساختار تعیین مشخصات است که نشان‌دهندهٔ هم یک رابطه و هم یک محدودیت منفصل بودن بین زیرنوعهای افراز می‌باشد. افراز یک نوع بر خصیصه‌هایی استوار است که هر یک خاص و منحصر به یک زیرنوع است. افرازها شامل مجموعهٔ کامل یا ناقص از زیرنوعها می‌توانند باشند. یک نوع می‌تواند چند افراز داشته باشد ـ به عبارتی دیگر، می‌تواند به طرق مختلفی جزءبندی شود. به‌عنوان مثال، نوع «مستخدم» هم می‌تواند به نوعهای «پاره‌وقت» و «تمام‌وقت» جزءبندی شود، و هم می‌تواند به نوعهای «کارمندی» و «هیئت علمی» جزءبندی شود.

منفعل (غیرفعال) کردن **Passivation**

فراروند کپی کردن متدها و ارزشهای یک شیءاز فضای آدرسی قابل اجرا به یک شکل پایدار.

شیءمنفعل **Passive Object**

یک شیءکه فراروندی خاص خودش نداشته باشد. یک شیءمنفعل معمولاً فقط به درخواستهای دیگر اشیاء پاسخ می‌دهد.

شیءپایدار **Presistent Object**

یک شیءکه می‌تواند دورهٔ حیاتی طولانیتر از فراروند خالق خود داشته باشد. یک شیءپایدار تا زمانی که صراحتاً حذف نشده باشد، در قید حیات است.

اشاره‌گر **Pointer**

یک ساختار پیاده‌سازی که به یک ارجاع صریح به یک شیءصورت خارجی می‌دهد. یک اشاره‌گر معمولاً یک آدرس حافظه است.

عمل چندریختی **Polymorphic Operation**

یک عمل که توسط بیش از یک متد پیاده‌سازی شده است.

چندریختی **Polymorphism**

یک مکانیزم دستگیری درخواستها که متدی را براساس نوع شیءدریافت‌کنندهٔ درخواست انتخاب می‌کند. این مکانیزم اجازهٔ مشخص کردن یک درخواست را که می‌تواند منجر به احضار متدهای

نام عمل **Operation Name**

نامی که در یک درخواست برای مشخص کردن یک عمل استفاده می‌شود.

عملگر **Operator**

یک نشانهٔ مخصوص یک زبان که بیانگر کنشی است که بایستی به اشیاء همجوار اعمال شود. به‌عنوان مثال، عملگر «+» در عبارت «a+b» بیانگر جمع (جبری) a و b است.

مرتبه **Order**

تعداد صفتهای یک نوع خاص.

بار اضافی **Overload**

تعریف بیش از یک متد با نام یکسان، ولی با امضاء و پیاده‌سازیهای متفاوت.

عمل باراضافی **Overloaded Operation**

نام عملی که بیش از یک درخواست برای آن مشخص شده است. متدی که باید به یک درخواست خاص پاسخ دهد، از طریق بررسی امضاء انتخاب می‌شود.

عملگر باراضافی **Overloaded Operator**

عملگری که به‌عنوان یک متد برای یک رده نیز تعیین شده باشد. نمونه‌ای از اینکار، تعریف مجدد عملگر «+» به‌عنوان متدی برای ردهٔ «string» (یک توالی از نویسه‌ها) است. بدینوسیله عملگر می‌تواند برای الحاق دو شیء‌از نوع «string» استفاده شود.

حاکم کردن **Override**

یک مکانیزم زبان پیاده‌سازی که اجازه می‌دهد یک متد از زیررده جایگزین یک متد از بالارده(فوق‌رده) شده و یا آن را بسط دهد.

ردهٔ پارامترگذاری شده **Parameterized Class(syn. Class Template)**

رده‌ای که صفتها و متدهایی را تعریف می‌کند که به مجموعه‌ای از رده‌های مختلف اعمال می‌شود. صفتها و متدها ارزشهای با نوعهای نامعلوم را می‌پذیرند. ردهٔ پارامترگذاری شده می‌تواند از طریق مشخص کردن نوعهای مورد استفادهٔ هر رده، برای ایجاد (خلق) رده‌ها استفاده شود.

و وراثت را پشتیبانی کند. زبانهای C++، Eiffel، Smalltalk و Objective-C نمونه‌هایی از زبانهای شیءگرا هستند.

سرویسهای شیء **Object Services**

توابع اصلی که برای ذخیره کردن و ادارهٔ شیء‌در طول حیات شیء‌فراهم شده باشند ـ مانند، ایجاد (خلق) کردن، نابود کردن (پایان دادن)، فعال کردن، غیرفعال کردن و تشخیص دادن.

حالت شیء **Object State (syn. State)**

مجموعه ارزشهای نگاشت شده به شیء‌ـ یا مجموعه نوعهایی که به یک شیء‌اعمال می‌شوند. در تعیین مشخصات شیء، حالتهای بالقوهٔ شیء‌می‌توانند به تعدادی نوعهای مجزای حالت تجرید شوند. حالت یک شیء‌می‌تواند عکس‌العمل شیء‌رانسبت به درخواستهای آتی مشخص کند.

نوع شیء **Object Type (syn. Type, Concept)**

یک انتزاع یا گزاره‌ای که می‌تواند به مجموعه‌ای از اشیاء اعمال شود ـ شکل (فرم) منطقی مجموعه‌ای از اشیاء مشابه. یک نوع شیء‌هر ایده یا مفهومی است که می‌تواند مشخص شود که به یک شیء‌اعمال شود یا نشود ـ در واقع اشیاء را بر حسب صفتها و رفتار با معنی رده‌بندی می‌کند.

کاربرد مطابق **Object Management Architecture (OMA)-Compliant Application**

یک کاربرد که از مجموعه‌ای از اشیاء و رده‌هایی تشکیل شده است که از طریق یک شیء‌واسطهٔ درخواستها فعل و انفعال می‌کنند. نتیجتاً این اشیاء و رده‌ها با تعاریف پروتکل OMA و مشخصات OMA فاصلها تطابق دارند.

شیء‌واسطهٔ درخواستها **Object Request Broker (ORB)**

یک مکانیزم نرم‌افزاری که اشیاء توسط آن درخواستها و پاسخهای خود را صادر و وصول می‌کنند.

عمل **Operation**

یک فرارونده یا سرویسی که بتواند به‌عنوان یک واحد درخواست شود. یک عمل در واقع پیاده‌سازی یک رفتار بوده و بخشی از یک رده است. یک عمل دارای یک امضاء، پیش شرط، پس شرط و نوع رویدادی است. یک عمل یک یا بیشتر متد دارد که فرارونده مشخص شده توسط عمل را انجام می‌دهند.

وراثت چندگانه **Multiple Inheritance**

یک مکانیزم زبان پیاده‌سازی که اجازه می‌دهد یک رده شامل صفتها و متدهایی باشد که برای بیش از یک بالارده (فوق رده) تعریف شده‌اند.

شیء **Object**

هر چیزی که یک نوع به آن اعمال شود ـ نمونه‌ای از نوع یا رده. یک نمونه از یک رده از ارزشهایی که به شیءپیوند خورده‌اند (حالت شیء) تشکیل شده و می‌تواند به درخواستهایی پاسخ دهد که برای رده مشخص شده‌اند.

شیئی **Object-Based**

هر زبان، روش یا سیستمی که مفاهیم هویت شیء، رده‌بندی و دربرگیرندگی را پشتیبانی می‌کند. یک سیستم شیئی وراثت را پشتیبانی نمی‌کند. زبان ایدا نمونه‌ای از یک زبان شیئی پیاده‌سازی است.

ایجاد (خلق) شیء **Object Creation**

فرارونده به‌وجود آوردن (خلق) یک شیءمجزا.

نابود کردن شیء **Object Destruction**

فرارونده پایان بخشیدن به حیات یک شیءو در اختیار قرار دادن منابع آن برای استفاده‌های دیگر.

هویت شیء **Object Identity (syn. Identity)**

این خصیصه که موجودیت یک شیءمستقل از هرگونه ارزشهای متعلق شده به شیءاست. این خصیصه متضاد مفهوم هویت ارزشی است ـ در مدل رابطه‌ای داده‌ها، یک رکورد بایستی یک کلید اصلی منحصر به‌فرد داشته باشد. هویت شیءمعمولاً با استفاده از دستگیره‌ها یا اشاره‌گرها پیاده سازی می‌شود.

فاصل شیء **Object Interface**

مجموعه درخواستهایی که شیءمی‌تواند به آنها پاسخ دهد ـ یعنی مشخصات رفتاری شیء. یک فاصل شیءدر واقع اتحاد فاصلهای نوع شیءاست.

نام شیء **Object Name**

ارزشی که یک شیءرا مشخص می‌کند.

شیءگرا **Object-Oriented**

هر زبان، روش یا سیستمی که مفاهیم هویت شیء، رده‌بندی، دربرگیرندگی درخواست بتواند اجرا شود. یک متد می‌تواند رفتار تعریف شده در عمل را بسط یا جای‌نویس کند. در بسیاری از سیستمها، انتخاب یک متد خاص برای پاسخگویی به یک درخواست می‌تواند در زمان کامپایل کردن برنامه یا اجرای برنامه انجام گیرد. یک متد در واقع یک پیاده‌سازی از یک رفتار یک نوع است.

Mixin

یک ردهٔ انتزاعی که برای مشخص کردن یک فاصل یا کارکرد مشترک برای تعدادی از زیررده‌ها بکار می‌رود. به‌عنوان مثال، «میلهٔ پیمایش» یک ردهٔ Mixin است. یک نمونه از میلهٔ پیمایش نمی‌تواند به تنهایی وجود داشته باشد، بلکه فقط صفتها و متدهایی را برای استفادهٔ نمونه‌های یک زیررده مانند «پنجره» تعریف و مهیا می‌کند.

محدودیت توجیحی **Modality Constaint**

محدودیتی که الزام نگاشت یک شیءرا از طریق یک رابطه مقرّر می‌کند. یک رابطه می‌تواند دلبخواه یا اجباری باشد. یک رابطهٔ دلبخواه بدین معنی است که نمونه‌های نوع می‌توانند به ارزشها نگاشت شوند. یک رابطهٔ اجباری بیانگر این است که کلیهٔ نمونه‌ها بایستی به یک یا بیشتر از ارزشها نگاشت شوند.

مدل **Model**

نمایشی از یک مسئله یا قلمرو موضوع که از انتزاع برای بیان مفاهیم استفاده می‌کند. یک مدل معمولاً شامل مجموعه‌ای از شماها (طرحواره‌ها) و سایر مستندات است.

پیمانه **Module**

مجموعه‌ای از اشیاء، متدها و رده‌هایی که برای فراهم آوردن زیرمجموعه‌ای از توانائیهای (کارکردهای) یک کاربرد با یکدیگر تشریک مساعی دارند. پیمانه‌ها می‌توانند حالت خود را حفظ کرده، و اطلاعات و رفتارها را با بقیهٔ کاربرد به اشتراک گذارند.

رده‌بندی چندگانه **Multiple Classification**

این خصیصه که یک شیءبتواند نمونه‌ای از بیش از یک نوع بوده و رده‌بندی منحصر به نوعهای متصل شده توسط روابط تعمیم نباشد. به‌عنوان مثال، یک نمونه از نوع «مستخدم بیمارستان» بتواند نمونه‌ای از نوع «بیمار» نیز باشد ـ این دو نوع توسط یک رابطهٔ تعمیم مرتبط نیستند. حتی در صورتیکه این دو نوع خود زیرنوعهای نوع «شخص» باشند، رده‌بندی چندگانه کماکان وجود خواهد داشت. در این‌صورت، زیرنوعها منفصل (جدا) نیستند.

نگاشت **Mapping**

یک قانون یا فراروندی که ارزشها را به اشیاء متعلق می‌کند. یک شیءبه شرطی به یک ارزش نگاشت می‌شود که اعمال قانون یا فراروند به شیءمنجر به بازگرداندن آن ارزش شود. یک نگاشت حتی می‌تواند تعلق یک شیءبه چند ارزش یا مجموعه‌ای از ارزشها را مشخص کند. بنابراین یک نگاشت در واقع یک شیءرا به یک یا بیشتر از ارزشها اتصال می‌دهد. یک نگاشت یک جهت ذاتی است. به‌عبارت دیگر، یک نگاشت خاص از یک مجموعه از اشیاء به مجموعه‌ای از ارزشها است.

درخواست با مفهوم **Meaningful Request**

درخواستی که پارامترهای واقعی آن با امضاء عمل نامبرده شده در درخواست مطابقت داشته باشد.

تابع عضو **Member Function (syn. Method)**

یک واژهٔ زبان C++ که به متدی (تابعی) که داخل یک رده اعلان شده است اشاره دارد. یک تابع عضو می‌تواند توسط یک درخواست احضار (فراخوانده) شود.

پیام **Message**

بسته‌ای از اطلاعات که عملی را که بایستی بر یک شیءاعمال شود، به‌همراه ارزشهای نشانوندهای عمل، مشخص می‌کند. شکل پیام توسط یک درخواست خاص تعریف می‌شود. پاسخ به یک پیام، در واقع احضار (فراخوانی) یک متد خاص، یا هرکنش دیگری است که توسط کاربرد معین شده باشد.

فوق مدل **Meta-Model**

مدلی که شامل نوعهایی است که نمونه‌های آنها نیز نوع هستند. فوق مدلها اغلب برای مشخص کردن سایر مدلها به‌کار می‌روند. به‌عنوان مثال، فوق مدل یک سیستم پایگاه رابطه‌ای داده‌ها، نوعهای «جدول»، «رکورد» و «پرونده» را مشخص می‌کنند.

فوقِ نوع **Meta-Type**

نوعی که نمونه‌های آن نیز نوعها باشند. به‌عنوان مثال، ممکن است نوع «مدل محصول» شامل نمونه‌هایی مانند «بلندگو»، «ضبط‌صوت» و «رادیو» باشد. هر یک از این نمونه‌ها خود نیز یک نوع است. بنابراین «رادیو» می‌تواند نمونه‌هایی مانند «توشیبا»، «ناسیونال» و غیره داشته باشد.

متد **Method**

یک پیاده‌سازی خاص از یک عمل یک‌رده ـ کدی که در پاسخ به یک عمل‌پذیری متقابل داشته باشد که متدهایی که به یک شیءاعمال می‌شوند بتوانند سرویسهای شیءدیگری را درخواست کنند.

تغییرناپذیر **Invariant**

شرطی که بایستی همیشه برقرار باشد. گزاره‌ای که باید همیشه ارزش درست داشته باشد.

Invariant Relation (syn. Immutable Relation)

رابطهٔ تغییرناپذیر

رابطه‌ای که نگاشتی غیرقابل تغییر را برای نمونه‌های نوع تعریف می‌کند. به‌عنوان مثال، نوع «ازدواج» دو رابطهٔ تغییرناپذیر «زن» و «شوهر» خواهد داشت. نمونه‌های «ازدواج» نمی‌توانند نگاشت به ارزشهای «زن» و «شوهر» را تغییر دهند.

تکرارکننده **Iterator**

یک ساختار زبان پیاده‌سازی که فرارونده دستیابی کلیهٔ اشیاء موجود در یک مجموعه یا رده را کنترل می‌کند.

سرویس چرخهٔ حیات **Life-Cycle Service**

اعمالی که ایجاد (خلق)، ختم (پایان)، کپی کردن و هم‌ارزی شیءرا اداره می‌کنند.

پیوند **Link**

اتصالی بین یک شیءو یک مقدار یا مجموعه‌ای از مقادیر. نمونه‌ای از یک نگاشت.

نویسهٔ لفظی **Literal**

مقداری که یک شیءنیست. در بسیاری از زبانهای پیاده‌سازی و ابزارها، یک نویسهٔ لفظی نمونه‌ای از یک نوع انتزاعی داده‌ها است. نوعهای «Integer»، «Float» و «String» مثالهایی از نوعهای انتزاعی داده‌ها هستند ـ لذا، '1'، '3.14' و 'Felix' نویسه‌های لفظی هستند.

فرم (شکل) منطقی **Logical Form**

یک ترتیب خاص مؤلفه‌ها ـ نحوهٔ قرارگرفتن یا ساخته شدن یک چیز.

شیءاداره شده **Managed Object**

مشتری یک سرویس مدیریت سیستم مانند نصب کردن، فعال کردن و کنترل اعمال.

بالارده (فوق رده) و ردهٔ خاص به زیر رده موسوم است. وراثت رابطه‌ای بین رده‌ها است که استفادهٔ مجدد کد، و همچنین تعریف یک فاصل عام به یک یا بیشتر از زیر رده‌ها را مقدور می‌سازد.

ارزشگذاری آغازین **Initialization**

فرایند گمارش ارزشها به یک شیء‌که به تازگی خلق شده است. نحوهٔ ارزشگذاری توسط سازندگان تجویز می‌شود.

نمودار نمونه‌ها **Instance Diagram**

یک نمودار که اشیائی خاص را به‌همراه نگاشتهای بین اشیاء نشان می‌دهد. معمولاً یک نمودار نمونه‌ها برای تفسیر کردن یک نمودار شیء(نوع) مربوطه استفاده می‌شود.

نمونه **Instance (syn. Object)**

یک شیء. بالاخص، یک شیء‌که توسط رابطهٔ رده‌بندی به یک نوع یا رده نگاشت شده است. به‌عنوان مثال، «تراشهٔ پنتیوم» نمونه‌ای از نوع «تراشه» است.

Instance Variable (syn. Attribute)

نمونه‌سازی **Instantiation**

عمل ایجاد (خلق) نمونه‌ای از یک نوع یا رده. در صورت ایجاد یک شیء‌نرم‌افزاری، حافظهٔ متناسب برای ضبط ارزشهای صفتهای شیء‌تخصیص می‌یابد. سازندگان مربوطهٔ رده نیز احضار می‌شوند تا اطمینان حاصل کنند که شیء‌با یک حالت اولیهٔ معتبر ایجاد می‌شود.

مفهوم **Intension**

تعریف یک نوع. مفهوم مشخص می‌کند که یک نوع به چه اشیائی اعمال می‌شود و به چه اشیائی اعمال نمی‌شود. مفهوم متشکل از صفتها و رفتار مشخص شدهٔ نوع می‌باشد.

Interface (syn. Public Interface, Protocol)

فاصل، واسط

مجموعه‌ای از درخواستهای قابل اعمال به یک رده.

نوع فاصلی **Interface Type**

گزاره‌ای که ارزشها را به‌عنوان اشیاء براساس یک تعریف رفتاری رده‌بندی می‌کند.

عمل‌پذیری متقابل **Interoperability**

توانایی تبادل درخواستها بین موجودیتها. اشیاء به‌شرطی می‌توانند

عمل عمومی **Generic Operation**

عملی که می‌تواند به بیش از یک متد مقیّد شود ـ به‌عبارت دیگر، عملی که توسط بیش از یک متد پیاده‌سازی شده است.

دستگیره **Handle**

ارزشی که منحصراً و به‌وضوح یک شیء‌را مشخص می‌کند (از سایر اشیاء تمیز می‌دهد).

پیاده‌سازی **Implemenation**

تحقّق (حصول) یک نوع خاص با استفاده از ساختارهای نحوی یک زبان برنامه‌سازی یا سیستم خاصّ. معمولاً پیاده‌سازی شامل تعاریفی برای صفتها و متدها است.

وارد کردن **Import**

ایجاد (خلق) یک شیء، براساس یک توصیف دریافت شده از یک زمینهٔ خارجی.

متد درون‌برنامه‌ای **In-line Method**

متدی که کامپایلر می‌تواند درخواستهای اجرای متد را با بلوک واقعی دستورالعملهایی که متد را تعریف می‌کنند، جایگزین نماید. اینکار با فراخوانی معمولی یک روال، که کامپایلر کدی را برای فراخوانی تابع کامپایل شده و پیوندخوردهٔ روال درکد مقصد تعبیه می‌کند، تفاوت دارد. متدهای درون برنامه‌ای به‌جهت بالا بردن کارآیی برنامه‌ها استفاده شده و بیشتر در مواقعی معمول هستند که متد حاوی تعداد اندکی دستورالعمل است.

جزءبندی ناقص **Incomplete Partition**

یک جزءبندی که حاوی کلیهٔ زیرنوعهای ممکن نمی‌باشد. در یک جزءبندی ناقص، نیازی نیست که یک نمونه از بالانوع (فوق نوع)، نمونه‌ای از یک زیرنوع مشخص باشد. یک جزءبندی ناقص را می‌توان چنین در نظر گرفت که شامل یک نوع تلویحی است. نمونه‌های بالانوع، که نمونه‌های زیرنوع مشخص نمی‌باشند، نمونه‌هایی از این نوع تلویحی هستند.

Inheritance (syn. Class Inheritance, Derivation)

وراثت

یک مکانیزم زبان که اجازه می‌دهد تعریف یک رده شامل صفتها و متدهای تعریف شده برای یک ردهٔ عامتر باشد. وراثت یک ساختار پیاده‌سازی برای روابط تخصصی است. در رابطهٔ وراثت، ردهٔ عام به

چارچوب از طریق تخصصی کردن و یا افزودن نوعها و رده‌های بیشتر پالایش می‌شود تا بتواند پاسخگوی نیازهای خاص مسئله باشد.

آشنا، دوست، رفیق **Friend**

یک رده یا یک متد که اجازهٔ دستیابی به اعضاء ردهٔ دیگری به آن اعطاء شده باشد.

تابع **Function**

یک قانون یا نگاشتی که یک شیءرا به یک ارزش یا مجموعه‌ای از ارزشها پیوند می‌زند.

فاصل کارکردی **Functional Interface**

فاصلی که مشخص‌کنندهٔ اعمالی است که توسط مشتریان یک سرویس احضار (فراخوانده) می‌شوند. حضّار اینگونه فاصلها، مشتریان سرویس هستند. این فاصلها در واقع نشانگر توانائیها (اعمال مفید فایدهٔ) سرویس هستند.

آشغالزدایی **Garbage Collection**

یک مکانیزم زبان پیاده‌سازی برای واپس‌گیری (بازستانی) خودکار حافظهٔ اشیائی که دیگر قابل دسترس نبوده یا ارجاعی به آنها وجود نداشته باشد.

Genealization Relation (syn. Specialization Relation)

رابطهٔ عمومیّت بخشی، رابطهٔ تعمیم

رابطه‌ای بین نوعها که بیانگر این مطلب است که ویژگیهای نوع عام بخشی از تعریف یک نوع خاصّتر هستند. در این رابطه نوع عام در واقع بالانوع (فوق‌نوع)، و نوع خاص در واقع زیرنوع است. یک رابطهٔ تعمیم ایجاب می‌کند که هر نمونه از زیرنوع، نمونه‌ای از بالانوع (فوق‌نوع) نیز باشد ـ به‌عبارت دیگر، دنباله (مصداق) زیرنوع جزیی از دَنباله (مصداق) بالانوع می‌شود. مفهوم زیرنوع نیز بایستی با مفهوم بالانوع سازگار باشد. لذا، زیرنوع می‌تواند صفتها و رفتارهایی را اضافه کند، ولی نباید صفتها و رفتارهای بالانوع را حذف کند یا با خصیصه‌های (ویژگیهای) بالانوع متناقض باشد.

شیءعمومی **Generic Object**

یک شیءکه مقصود اصلی آن به فاصل آن مربوط نمی‌باشد. این بدین معنی است که هر شیءاز هر نوعی که باشد می‌تواند فاصل را به ارث برده و برای فاصل یک پیاده‌سازی ایجاد کند.

مجموعه‌ای از اعمال مجاز که دستیابی به و اصلاح ارزشها را کنترل می‌کنند. دربرگیرندگی (محصورسازی) یک مکانیزم زبان پیاده‌سازی است که بدینوسیله جنبه‌های خارجی از پیاده‌سازی یک شیءجدا می‌شوند.

رویداد، واقعه **Event**

تغییر قابل ملاحظه در حالت یک شیءیا علامت که موجب تغییر رفتار یک شیءمی‌شود. یک رویداد می‌تواند ایجاد، ختم، رده‌بندی، فسخ رده‌بندی، یا تغییر در ارزش را برای یک شیءعلامت بدهد. به‌عنوان مثال، «ایجاد یک حساب بانکی جدید»، یا «بدهکارکردن مبلغ ۳۰۰ تومان از یک حساب خاص بانکی».

نوع رویدادی **Event Type**

یک نوع از رویداد ـ مجموعه‌ای از تغییرات مشابه حالت حاصل شده‌از یک عمل. یک نوع رویدادی دارای یک پیش‌شرط و یک پس‌شرط است که به هر رویدادی اعمال می‌شود. «حساب بانکی بدهکار شد» و «حساب بانکی ایجاد شد» مثالهایی از نوعهای رویدادی هستند.

قالب مبادله‌ای **Exchange Format**

تشریحی از مشخصات یا پیاده‌سازی یک شیءکه به شیءقدرت صادر شدن و وارد شدن را می‌دهد.

صادر کردن، صدور **Export**

ارسال توضیح (تشریح) یک شیءبه یک زمینهٔ خارجی.

مصداق، دنباله **Extension (syn. Set, Extent)**

مجموعه‌ای از اشیاء که یک نوع بر آنها اعمال می‌شود. اشیاء موجود در مصداق را نمونه‌های نوع گویند.

مولّد **Factory**

موجودیتی که یک سرویس خاص برای خلق (ایجاد) اشیاء فراهم می‌کند.

پارامتر صوری **Formal Parameter**

یک شیءمحلی با نام که به‌عنوان نشانوند به یک عمل ردّ می‌شود. ارزش شیء(پارامتر واقعی) توسط مشتری و در زمان احضار متد، گمارده می‌شود.

چارچوب **Framework**

یک مشخصه یا پیاده‌سازی (یعنی، مجموعه‌ای از رده‌ها) که راه‌حلّی کلّی به یک مسئله خاص یا جنبه‌ای از کاربرد، ارائه می‌کنند. معمولاً یک

متخصص قلمرو **Domain Expert**

شخصی که مفاهیم یک مسئله، فرآروند یا فعالیت خاص را استفاده می‌کند ـ به‌عنوان مثال، کاربران، برنامه‌سازان، یا مدیران.

نوع قلمرو **Domain Type**

نوعی که توسط یک رابطه به مجموعه‌ای از ارزشها نگاشت شده باشد.

قلمرو **Domain**

مسئله یا موضوعی که توسط یک سیستم محاسباتی یا غیرمحاسباتی مورد توجه قرار می‌گیرد. همچنین، مجموعه اشیاء نگاشت شده توسط یک رابطه به مجموعه‌ای از ارزشها.

Dynamic Object-Based Application

کاربرد پویای شیئی

توانائیهای در دسترس کاربر نهایی که توسط یک یا بیشتر از برنامه‌های متشکل از اشیاء فعل و انفعالی مهیا شده‌اند.

تقیّد پویا **Dynamic Binding (syn. Polymorphism)**

متعلق ساختن یک اسم به یک رده در زمان اجرای برنامه. برای درخواستها، متد موردنظر و ارزشهای هر عمل در هنگام اجرا مشخص می‌شود. انتخاب متد مناسب می‌تواند براساس ردهٔ شیءمقصد یا سایر متغیرها انجام بگیرد.

رده‌بندی پویا **Dynamic Classification**

توانایی تغییر دادن نوعهایی که به یک شیءاِعمال می‌شوند. یک شیءمی‌تواند رده‌بندی خود را به مرور زمان تغییر دهد. به‌عنوان مثال، یک کپی تنها از یک کتاب می‌تواند از نمونه‌ای از نوع «موجود» به نمونه‌ای از نوع «امانت داده شده» تغییر یابد. اگر چه رده‌بندی پویا در اکثر زبانهای پیاده‌سازی پشتیبانی نشده است، ولی مفهوم مهم و مفیدی برای تعیین مشخصات محسوب می‌شود.

جاسازی **Embedding**

تعریف یک شیءکه حاوی یک ارزش غیرشیئی است. مقصود از جاسازی این است که به موجودیتهای نرم‌افزاری (مانند کاربردها) قابلیت عمل کردن در محیطهای شیءگرا را بدهد.

Encapsulation (syn. Information Hiding, Data Hiding)

دربرگیرندگی، محصورسازی

این خصیصه که یک شیءواحدی است گسسته از داده‌ها به‌همراه به‌عبارت دیگر به‌شیءدوّم نمایندگی پاسخ به‌درخواست دریافتی را می‌دهد. نمایندگی می‌تواند به‌عنوان آلترناتیوی برای وراثت استفاده شود.

اشتقاق **Derivation**

فرارونده تعریف کردن یک ردهٔ جدید از طریق ارجاع کردن به یک ردهٔ موجود و سپس اضافه کردن صفتها و متدهای آن. ردهٔ موجود، ردهٔ مبنا یا بالارده (فوق رده) بوده و بوسیلهٔ عمل اشتقاق تغییری در آن حاصل نمی‌شود. ردهٔ جدید به زیررده یا ردهٔ مشتق شده موسوم است.

Derived Class (syn. Subclass, Child Class)

ردهٔ مشتق شده

رده‌ای که صفتها و متدهای ردهٔ دیگری را به‌ارث می‌برد.

طرّاحی **Design**

بخشی از فراروند تشریح مشخصات که در برگیرندهٔ ملاحظات قبل از پیاده‌سازی است. طراحی فراروندی است که مشخصات یافت شدهٔ یک فعالیت تحلیل را توسعه می‌دهد و اصلاح می‌کند ـ بعضاً برخی کیفیتها، مانند قابلیّت استفادهٔ مجدّد، آزمون‌پذیری، نگهداشت‌پذیری و انعطاف‌پذیری را درکاربرد وارد می‌کند. طرّاحی شامل تعیین مشخصات نیازمندیهای پیاده‌سازی، مانند یک فاصل کاربر و پایداری داده‌ها، نیز می‌باشد.

ویرانگر **Destructor**

یک متد خاص که هنگام از بین بردن یک شیءبه‌طور خودکار فراخوانده می‌شود. مقصود ویرانگر در واقع برگرداندن منابع سیستم جهت استفاده‌های آتی و فراخوانی هر درخواست مورد نیاز توسط معنای رده، می‌باشد.

نمونهٔ مستقیم **Direct Instance**

یک شیءیک نمونهٔ مستقیم از یک نوع یا ردهٔ X است، به‌شرط آنکه شیءنمونه‌ای از X باشد و در عین حال نمونه‌ای از هیچ زیرداده‌ای از X نباشد.

محدودیت منفصل **Disjoint Constraint**

محدودیتی که اجازه نمی‌دهد که یک شیء، نمونه‌ای از بیش از یکی از نوعهای مشخص شده باشد. به‌عنوان مثال، نوعهای «حاضرین» و «غایبین» منفصل هستند ـ یک شخص نمی‌تواند هم یکی از حاضرین باشد و هم یکی از غائبین.

عمل مستقل از زمینه **Context-Independent Operation**

عملی که برای آن رفتار فراخوانده شده به‌هویّت یا موقعیّت مکانی مشتری صادرکنندهٔ درخواست بستگی ندارد.

قرارداد **Contract**

مجموعه‌ای چسبنده از سرویسهایی که یک خدمتکار به‌مشتریان عرضه می‌کند. یک قرارداد اغلب از مجموعه‌ای از متدها برای پاسخگویی به‌درخواستهای مشخّص تشکیل شده است.

اتّصال **Coupling**

توضیحی از وابستگیهای بین رده‌ها. دو رده به‌شرطی دارای اتّصال محکم هستند که رابطهٔ معکوس یا رابطهٔ مشتری‌ـخدمتکار بین آنها وجود داشته باشد. قطع اتصال هنگامی رخ می‌دهد که هیچیک از رده‌ها برای یک سرویس درخواستی به‌ردهٔ دیگر نیازی نداشته باشد.

Data Member (syn. Attribute,Instance or Class Variable)

عضو داده‌ای

یک خصیصهٔ بانام از یک رده که ارزشها به‌آن قابل نگاشت هستند. اعضاء داده‌ای، حالت اشیاء یا صفتهای خود رده را تعریف می‌کنند. به‌عنوان مثال، ردهٔ «حساب بانکی» می‌تواند یک عضو داده‌ای «موجودی» داشته باشد که به‌عنوان یک متغیر از نوع عدد حقیقی تعریف شده باشد.

مدل داده‌ای **Data Model**

مجموعه‌ای از موجودیتها، اعمال و قوانین سازگاری.

نسخ رده‌بندی **Declassification**

نتیجهٔ این اظهار که دیگر یک شیء‌نمونهٔ یک نوع نیست. عمل فسخ رده‌بندی در واقع شیء‌را از مجموعهٔ نمونه‌های نوع خارج می‌کند.

تجزیه **Decomposition (syn. Factorization)**

جایگزینی یک نوع تنها با یک نوع مرکّب و تعدادی نوعهای ساده‌تر مؤلفه‌ای. همچنین، جایگزینی یک ردهٔ تنها با تعدادی رده‌های ساده‌تر مؤلفه‌ای. معمولاً رده‌های مؤلفه‌ای اتصال ندارند ـ لذا، آزمودن و اداره کردن ردهٔ مولفه‌ای راحتتر بوده، و رده‌های مؤلفه‌ای برای استفادهٔ مجدّد مناسبتر هستند.

واگذاری، نمایندگی **Delegation**

صدور یک درخواست به‌یک شیء‌دیگر در پاسخ به‌درخواست دریافتی از یک شیء. شیء‌اوّل مسئولیت را به‌شیء‌دوّم واگذار می‌کند، یا

ردهٔ واقعی **Concrete Class**

رده‌ای که فقط می‌تواند نمونه‌های مستقیم داشته باشد و نمی‌تواند به‌عنوان یک ردهٔ مبنا استفاده شود.

مطابقت، همخوانی **Conformance**

رابطه‌ای بین نوعها به‌نحوی که اگر نوع X با نوع Y مطابق باشد، سپس هر ارزشی که نوع X را ارضاء نماید، نوع Y را نیز ارضاء نماید.

سازگاری **Consistency**

اطمینان از اینکه مجموعه‌ای از ارزشها شرایط مقرّر را ارضاء می‌نمایند. به‌عنوان مثال، سازگاری حالت توسط شرایط تغییر ناپذیر (عباراتی حاوی ارزشهای مشخص صفتها) برقرار می‌شود. سازگاری روابط می‌تواند از طریق محدودیتهای تعداد برقرار شود.

محدودیت **Constraint**

گزاره‌ای که درستی آن باید برقرار باشد ـ یا شرطی که بایستی ارضاء شود. معمولاً محدودیتها به‌حالتهای مجاز یک شیء‌یا مصادیق یک یا بیشتر از نوعها اعمال می‌شوند. به‌عنوان مثال، اینکه نمونه‌های نوع «حساب بانکی» ارزشهای منحصر به‌فردی برای صفت «شمارهٔ حساب» داشته باشند، و یا اینکه مصادیق نوعهای «خازن» و «مقاومت» مجزّا بوده و اشتراکی نداشته باشند، محدودیت محسوب می‌شوند.

فاصل سازندگی **Construction Interface**

یک فاصل که اعمالی را تعریف می‌کند که برای برقراری ارتباط بین یک سرویس شیء‌با سایر اشیایی که بایستی در جهت فراهم آوردن سرویس همکاری نمایند، استفاده می‌شود.

سازنده **Constructor**

یک متد مخصوص که به‌هنگام خلق یک نمونه از رده فراخوانده می‌شود. معمولاً سازنده، ارزشگذاریهای آغازین را نیز انجام می‌دهد.

ردهٔ ظرفی **Container Class**

رده‌ای که نمونه‌هایش مجموعه‌هایی از سایر اشیاء هستند. یک ردهٔ ظرفی ساختارهای داده را برای ضبط مجموعه، و متدها را برای ادارهٔ مجموعه، تعریف می‌کند. یک پشته از اعداد صحیح و یک لیست پیوندی از اشیاء نگاره‌ای مثالهایی از رده‌های ظرفی هستند. به‌طورکلّی متدها کنشهایی را انجام می‌دهند که به‌طور خاص بانوع مجموعه بستگی دارد ـ به‌عنوان مثال، «گذاردن» و «برداشتن» بر روی ردهٔ «پشته».

بخش کامل **Complete Partition**

یک بخش که حاوی کلیهٔ زیرنوعهای ممکن باشد. بنابراین یک نمونه از فوق نوع (بالانوع) لزوماً نمونه‌ای از دقیقاً یکی از زیرنوعها است.

شیءمؤلّفه‌ای **Component Object**

یک شیءکه بخشی از یک شیءمرکّب دیگر است.

Composite Object (syn. Compound Object)
شیءمرکّب

یک شیءکه از دو یا بیشتر از اشیاء جداگانه تشکیل شده است. اعتبار (درستی) شیءمرکّب به‌اتصال مداوم با اشیاء مولفه‌ای شیءبستگی دارد. شیءمرکب توسط مؤلفه‌هایش تعریف می‌شود ـ پایانیابی یک شیءمؤلفه‌ای به‌پایانیابی یا بهم‌زدن رده‌بندی شیءمرکّب نیاز دارد.

Composite Type (syn. Typed Relation, Relation)
نوع مرکّب

یک نوع که نمونه‌های آن اشیاء مرکّب هستند. یک نوع مرکّب حاوی یک یا بیشتر رابطهٔ تغییر ناپذیر است.

ترکیب **Composition (syn. Aggregation)**

مشخص کردن نوعی که هر یک از نمونه‌های آن از سایر اشیاء تشکیل شده‌اند ـ یعنی اینکه هر نمونه یک چندتایی است. یک مثال، مشخص کردن نوع «ازدواج» است. نمونه‌های این نوع شامل ارزشهای «شوهر» و «زن» می‌باشند. بنابراین هر شیء«ازدواج» یک زوج (رابطهٔ دوتایی مرتّب از ارزشهای زن و شوهر) است. همچنین، مشخص کردن رده‌ای که نمونه‌هایش مستقیماً به‌اشیاء دیگر ارجاع می‌کنند.

رابطهٔ محاسباتی **Computed Relation**

رابطه‌ای که از سایر روابط یا اطلاعات محاسبه شده باشد. به‌عنوان مثال، رابطهٔ «سنّ» می‌تواند از ارزش رابطهٔ «تاریخ تولّد» و تاریخ زمان حال محاسبه شود.

نوع محاسباتی **Computed Type**

یک نوع که از سایر روابط یا اطلاعات محاسبه شود. مشخص کردن یک نوع محاسباتی معمولاً با استفاده از عبارات بر مبنای نظریهٔ مجموعه‌ها، مانند اتّحاد (اجتماع) و تقاطع (تفریق)، انجام می‌گیرد. به‌عنوان مثال، نوع «سوانح اضطراری» می‌تواند از طریق تقاطع نوعهای «موارد اضطراری» و «سوانح» مشخص گردد.

روابط وراثتی به یکدیگر متصل می‌شوند. در صورت وجود وراثت چندگانه، ساختار روابط رده‌ها یک مشبّک (توری) خواهد بود.

عضو رده **Class Member**

یک صفت یا متد مشخص شده برای یک رده.

متد رده **Class Method**

یک متد از یک رده که به‌عوض تعریف کردن رفتار نمونه‌های رده، رفتار خود رده را تعریف می‌کند. یک متد رده می‌تواند برای دستیابی و اصلاح صفتهای رده، به‌کار رود. به‌عنوان مثال، متد رده «شماره» می‌تواند برای ردهٔ «مستخدم» تعریف شود.

شیءرده **Class Object**

یک شیءکه خود یک رده یا مولّد اشیاء است.

رده‌ای **Class-Based**

هر روش، زبان برنامه‌سازی یا سیستمی که مفهوم نوعهای انتزاعی داده‌ها را پشتیبانی کرده ملزم نماید که هر شیءنمونه‌ای از یک نوع انتزاعی داده یا رده باشد. زبان C++ نمونه‌ای از یک زبان پیاده‌سازی رده‌ای است.

رده‌بندی **Classification**

نتیجهٔ نگاشت یک شیءبه‌یک نوع که شیءیکی از نمونه‌های آن است. رده‌بندی رابطه‌ای بین یک نوع و نمونه‌هایش تعریف می‌کند. نگاشت رده‌بندی در واقع مصادیق یک نوع را مشخص می‌کند.

مشتری‌ـمخدوم **Client**

یک موجودیّت (مانند یک شیء، رده، یا کاربرد) که در خواستی برای یک سرویس صادر می‌کند. موجودیتی که به‌درخواست پاسخ می‌دهد به خدمتکار یا خادم موسوم است.

شیءمشتری **Client Object**

یک شیءکه درخواست سرویسی را می‌کند.

همکاری **Collaboration**

فراهم نمودن یک سرویس بین دو رده، از طریق یک رابطهٔ مشتری‌ـخدمتکار (خادم‌ـمخدوم).

تسهیلات مشترک **Common Facilities**

تسهیلات قابل استفاده در تعداد کثیری از کاربردها.

تعداد،اندازه **Cardinality**

عدهٔ اشیاء موجود در یک مجموعه یا نمونه‌های یک نوع.

محدودیت تعداد **Cardinality Constraint (syn. Multiplicity)**

محدودیتی برای محدودکردن تعداد نمونه‌های یک نوع. بعنوان مثال، تعداد نمونه‌های نوع «شخص» می‌تواند به ده عدد محدود شود. محدودیتهای تعداد می‌توانند به‌روابط نیز اعمال شوند. در اینصورت تعداد ارزشهایی که یک شیء‌می‌تواند در یک رابطه به‌آن نگاشت شود، محدود می‌شود. به‌عنوان مثال، رابطهٔ «کتابهای به‌امانت داده شده» از نوع «امانت گیرنده» به‌نوع «کپیها»، می‌تواند بین صفر تا ده محدود شود. لذا حداکثر تعداد کتابهایی که می‌توانند به‌امانت داده شوند ده عدد است. این محدودیت، اتصالات ممکن در نگاشت از «امانت گیرنده» به «کپیها» را محدود می‌کند.

ویژگی،خصوصیت **Charactristic**

خصیصه یا رفتار یک نوع.

پیشC **Cfront**

یک پیش‌پرداز که کد C++ را به‌کد C ترجمه می‌کند. پیشC مکانیزم اولیه زبان پیاده‌سازی بوده و به‌یک کامپایلر میانی زبان C متکی بوده است، ولی به‌طور گسترده توسط کامپایلرهایی جایگزین شده است که مستقیماً کد قابل اجرا برای کد C++ تولید می‌کنند.

رده **Class**

یک پیاده‌سازی از یک نوع دادهٔ انتزاعی. تعریفی از ساختار داده‌ها، متدها، و فاصل اشیاء نرم‌افزاری. قالبی برای ایجاد (نمونه‌سازی) اشیاء نرم‌افزاری.

صفت رده **Class Attribute(syn. Class Variable)**

صفتی که یک خصیصهٔ مجموعه اشیاء تعریف شده توسط رده را پیاده‌سازی می‌کند. ارزش نگاشت شده توسط یک صفت رده به یک ردهٔ خاص، به‌هر نمونه از رده اعمال می‌شود. یک مثال می‌تواند صفت «نرخ بهره» برای ردهٔ «حساب پس‌انداز» باشد. این نگاشت بدین معنی است که کلیهٔ حسابهای پس‌انداز دارای یک نرخ بهرهٔ یکسان هستند.

سلسله مراتب رده **Class Hierarchy**

یک نوع توصیف روابط وراثتی مابین رده‌ها. یک سلسله مراتب رده معمولاً به‌صورت یک گراف غیردوری جهت‌دار یا یک درخت، نشان داده می‌شود. گره‌ها نمایانگر رده‌ها بوده و توسط کمانهای نشان‌دهندهٔ

تقیّد **Binding (syn. Method Selection, Method Dispatching)**

متعلّق کردن یک اسم به یک رده. معمولاً تقیّد به‌انتخاب یک متد و ارزشها در پاسخ به‌یک درخواست اشاره دارد. تقیّد یا در زمان کامپایل (زود یا به‌طور ایستا) و یا در زمان اجرای برنامه (دیرتر یا به‌طور پویا) انجام می‌گیرد.

تورّق‌گر **Browser**

یک ابزار نرم‌افزاری که به‌برنامه‌ساز امکان مشاهدهٔ رده‌ها، صفتها و متدها را بدهد.

نوع پیش‌ساخته **Built-In Type**

یک نوع دادهٔ انتزاعی که به‌عنوان بخشی لاینفک از یک سیستم یا زبان پیاده‌سازی شده باشد. زبان پیاده‌سازی یا سیستم زیرین پیاده‌سازی، اعمال لازم برای دستکاری ارزشهای نوعهای پیش‌ساخته را فراهم می‌کنند. نوع دادهٔ انتزاعی 'int' در زبان C++ نمونه‌ای از نوع پیش‌ساخته است.

فراخوانی با مرجع **Call by Reference**

یک مکانیزم زبان پیاده‌سازی که در آن هویت یک شیء‌به‌عنوان ارزش نشانوند در فراخوانی یک متد استفاده می‌شود. نشانوندهایی که مرجع آنها در فراخوانیها ردّ می‌شوند، می‌توانند در حوزهٔ متد فراخوانده تغییر داده شوند. از محاسن فراخوانی با مرجع این است که به‌تخصیص حافظه برای انتقال یک کپی از شیء‌نیازی نبوده و وضعیت (حالت) شیء‌قابل اصلاح و تغییر می‌باشد. عیب این نوع فراخوانی این است که استفاده کنندهٔ متدی که خواهان فراخوانی با مرجع است بایستی صرفاً امیدوار باشد که متد فراخوانده شده وضعیت (حالت) شیء‌نشانوند را خراب نکند.

فراخوانی با مقدار **Cally by Value**

یک مکانیزم زبان پیاده سازی که در آن یک کپی از یک شیء‌به‌عنوان ارزش نشانوند در فراخوانی یک متد استفاده می‌شود. قبل از رد شدن شیء‌نشانوند به‌متد فراخوانده شده، یک کپی محلّی از آن تهیه می‌شود. نشانوندهایی که ارزش (مقدار) آنها در یک فراخوانی ردّ می‌شوند، قابل تغییر در حوزهٔ متد فراخوانده شده نمی‌باشند. حسن این نوع فراخوانی این است که از تغییر و دستکاری نشانوند توسط متد فراخوانده شده جلوگیری می‌کند. یک عیب این نوع فراخوانی این است که برای ضبط شیء‌ردّشده بایستی حافظه تخصیص یافته و متدهای سازنده برای ارزشگذاری آغازین آنها فراخوانده شوند.

بخش کامل **Complete Partition**

یک بخش که حاوی کلیهٔ زیرنوعهای ممکن باشد. بنابراین یک نمونه از فوق نوع (بالانوع) لزوماً نمونه‌ای از دقیقاً یکی از زیرنوعها است.

شیءمؤلّفه‌ای **Component Object**

یک شیءکه بخشی از یک شیءمرکّب دیگر است.

Composite Object (syn. Compound Object)
شیءمرکّب

یک شیءکه از دو یا بیشتر از اشیاء جداگانه تشکیل شده است. اعتبار (درستی) شیءمرکّب به‌اتصال مداوم با اشیاء مولفه‌ای شیءبستگی دارد. شیءمرکب توسط مؤلفه‌هایش تعریف می‌شود ـ پایانیابی یک شیءمؤلفه‌ای به‌پایانیابی یا بهم‌زدن رده‌بندی شیءمرکّب نیاز دارد.

Composite Type (syn. Typed Relation, Relation)
نوع مرکّب

یک نوع که نمونه‌های آن اشیاء مرکّب هستند. یک نوع مرکّب حاوی یک یا بیشتر رابطهٔ تغییر ناپذیر است.

ترکیب **Composition (syn. Aggregation)**

مشخص کردن نوعی که هر یک از نمونه‌های آن از سایر اشیاء تشکیل شده‌اند ـ یعنی اینکه هر نمونه یک چندتایی است. یک مثال، مشخص کردن نوع «ازدواج» است. نمونه‌های این نوع شامل ارزشهای «شوهر» و «زن» می‌باشند. بنابراین هر شیء«ازدواج» یک زوج (رابطهٔ دوتایی مرتّب از ارزشهای زن و شوهر) است. همچنین، مشخص کردن رده‌ای که نمونه‌هایش مستقیماً به‌اشیاء دیگر ارجاع می‌کنند.

رابطهٔ محاسباتی **Computed Relation**

رابطه‌ای که از سایر روابط یا اطلاعات محاسبه شده باشد. به‌عنوان مثال، رابطهٔ «سنّ» می‌تواند از ارزش رابطهٔ «تاریخ تولّد» و تاریخ زمان حال محاسبه شود.

نوع محاسباتی **Computed Type**

یک نوع که از سایر روابط یا اطلاعات محاسبه شود. مشخص کردن یک نوع محاسباتی معمولاً با استفاده از عبارات بر مبنای نظریهٔ مجموعه‌ها، مانند اتّحاد (اجتماع) و تقاطع (تفریق)، انجام می‌گیرد. به‌عنوان مثال، نوع «سوانح اضطراری» می‌تواند از طریق تقاطع نوعهای «موارد اضطراری» و «سوانح» مشخص گردد.

روابط وراثتی به یکدیگر متصل می‌شوند. در صورت وجود وراثت چندگانه، ساختار روابط رده‌ها یک مشبّک (توری) خواهد بود.

عضو رده **Class Member**

یک صفت یا متد مشخص شده برای یک رده.

متد رده **Class Method**

یک متد از یک رده که به‌عوض تعریف کردن رفتار نمونه‌های رده، رفتار خود رده را تعریف می‌کند. یک متد رده می‌تواند برای دستیابی و اصلاح صفتهای رده، به‌کار رود. به‌عنوان مثال، متد رده «شماره» می‌تواند برای ردهٔ «مستخدم» تعریف شود.

شیءرده **Class Object**

یک شیءکه خود یک رده یا مولّد اشیاء است.

رده‌ای **Class-Based**

هر روش، زبان برنامه‌سازی یا سیستمی که مفهوم نوعهای انتزاعی داده‌ها را پشتیبانی کرده ملزم نماید که هر شیءنمونه‌ای از یک نوع انتزاعی داده یا رده باشد. زبان C++ نمونه‌ای از یک زبان پیاده‌سازی رده‌ای است.

رده‌بندی **Classification**

نتیجهٔ نگاشت یک شیءبه‌یک نوع که شیءیکی از نمونه‌های آن است. رده‌بندی رابطه‌ای بین یک نوع و نمونه‌هایش تعریف می‌کند. نگاشت رده‌بندی در واقع مصادیق یک نوع را مشخص می‌کند.

مشتری‌ـمخدوم **Client**

یک موجودیّت (مانند یک شیء، رده، یا کاربرد) که در خواستی برای یک سرویس صادر می‌کند. موجودیتی که به‌درخواست پاسخ می‌دهد به خدمتکار یا خادم موسوم است.

شیءمشتری **Client Object**

یک شیءکه درخواست سرویسی را می‌کند.

همکاری **Collaboration**

فراهم نمودن یک سرویس بین دو رده، از طریق یک رابطهٔ مشتری‌ـخدمتکار (خادم‌ـمخدوم).

تسهیلات مشترک **Common Facilities**

تسهیلات قابل استفاده در تعداد کثیری از کاربردها.

تعداد،اندازه **Cardinality**

عدهٔ اشیاء موجود در یک مجموعه یا نمونه‌های یک نوع.

محدودیت تعداد **Cardinality Constraint (syn. Multiplicity)**

محدودیتی برای محدودکردن تعداد نمونه‌های یک نوع. بعنوان مثال، تعداد نمونه‌های نوع «شخص» می‌تواند به ده عدد محدود شود. محدودیتهای تعداد می‌توانند به‌روابط نیز اعمال شوند. در اینصورت تعداد ارزشهایی که یک شیء‌می‌تواند در یک رابطه به‌آن نگاشت شود، محدود می‌شود. به‌عنوان مثال، رابطهٔ «کتابهای به‌امانت داده شده» از نوع «امانت گیرنده» به‌نوع «کپیها»، می‌تواند بین صفر تا ده محدود شود. لذا حداکثر تعداد کتابهایی که می‌توانند به‌امانت داده شوند ده عدد است. این محدودیت، اتصالات ممکن در نگاشت از «امانت گیرنده» به «کپیها» را محدود می‌کند.

ویژگی،خصوصیّت **Charactristic**

خصیصه یا رفتار یک نوع.

پیشC **Cfront**

یک پیش‌پرداز که کد C++ را به‌کد C ترجمه می‌کند. پیشC مکانیزم اولیه زبان پیاده‌سازی بوده و به‌یک کامپایلر میانی زبان C متکی بوده است، ولی به‌طور گسترده توسط کامپایلرهایی جایگزین شده است که مستقیماً کد قابل اجرا برای کد C++ تولید می‌کنند.

رده **Class**

یک پیاده‌سازی از یک نوع دادهٔ انتزاعی. تعریفی از ساختار داده‌ها، متدها، و فاصل اشیاء نرم‌افزاری. قالبی برای ایجاد (نمونه‌سازی) اشیاء نرم‌افزاری.

صفت رده **Class Attribute(syn. Class Variable)**

صفتی که یک خصیصهٔ مجموعه اشیاء تعریف شده توسط رده را پیاده‌سازی می‌کند. ارزش نگاشت شده توسط یک صفت رده به یک ردهٔ خاص، به‌هر نمونه از رده اعمال می‌شود. یک مثال می‌تواند صفت «نرخ بهره» برای ردهٔ «حساب پس‌انداز» باشد. این نگاشت بدین معنی است که کلیهٔ حسابهای پس‌انداز دارای یک نرخ بهرهٔ یکسان هستند.

سلسله مراتب رده **Class Hierarchy**

یک نوع توصیف روابط وراثتی مابین رده‌ها. یک سلسله مراتب رده معمولاً به‌صورت یک گراف غیردوری جهت‌دار یا یک درخت، نشان داده می‌شود. گره‌ها نمایانگر رده‌ها بوده و توسط کمانهای نشان‌دهندهٔ

تقیّد **Binding (syn. Method Selection, Method Dispatching)**

متعلّق کردن یک اسم به یک رده. معمولاً تقیّد به‌انتخاب یک متد و ارزشها در پاسخ به‌یک درخواست اشاره دارد. تقیّد یا در زمان کامپایل (زود یا به‌طور ایستا) و یا در زمان اجرای برنامه (دیرتر یا به‌طور پویا) انجام می‌گیرد.

توزّق‌گر **Browser**

یک ابزار نرم‌افزاری که به‌برنامه‌ساز امکان مشاهدهٔ رده‌ها، صفتها و متدها را بدهد.

نوع پیش‌ساخته **Built-In Type**

یک نوع دادهٔ انتزاعی که به‌عنوان بخشی لاینفک از یک سیستم یا زبان پیاده‌سازی شده باشد. زبان پیاده‌سازی یا سیستم زیرین پیاده‌سازی، اعمال لازم برای دستکاری ارزشهای نوعهای پیش‌ساخته را فراهم می‌کنند. نوع دادهٔ انتزاعی 'int' در زبان C++ نمونه‌ای از نوع پیش‌ساخته است.

فراخوانی با مرجع **Call by Reference**

یک مکانیزم زبان پیاده‌سازی که در آن هویت یک شیء‌به‌عنوان ارزش نشانوند در فراخوانی یک متد استفاده می‌شود. نشانوندهایی که مرجع آنها در فراخوانیها ردّ می‌شوند، می‌توانند در حوزهٔ متد فراخواننده تغییر داده شوند. از محاسن فراخوانی با مرجع این است که به‌تخصیص حافظه برای انتقال یک کپی از شیء‌نیازی نبوده و وضعیت (حالت) شیء‌قابل اصلاح و تغییر می‌باشد. عیب این نوع فراخوانی این است که استفاده کنندهٔ متدی که خواهان فراخوانی با مرجع است بایستی صرفاً امیدوار باشد که متد فراخوانده شده وضعیت (حالت) شیء‌نشانوند را خراب نکند.

فراخوانی با مقدار **Cally by Value**

یک مکانیزم زبان پیاده سازی که در آن یک کپی از یک شیء‌به‌عنوان ارزش نشانوند در فراخوانی یک متد استفاده می‌شود. قبل از رد شدن شیء‌نشانوند به‌متد فراخوانده شده، یک کپی محلّی از آن تهیه می‌شود. نشانوندهایی که ارزش (مقدار) آنها در یک فراخوانی ردّ می‌شوند، قابل تغییر در حوزهٔ متد فراخوانده شده نمی‌باشند. حسن این نوع فراخوانی این است که از تغییر و دستکاری نشانوند توسط متد فراخوانده شده جلوگیری می‌کند. یک عیب این نوع فراخوانی این است که برای ضبط شیء‌ردّشده بایستی حافظه تخصیص یافته و متدهای سازنده برای ارزشگذاری آغازین آنها فراخوانده شوند.

یکپارچگی **Atomicity**

اطمینان از اینکه یک عمل وضعیت (حالت) کلیهٔ اشیاء درگیر (شرکت‌کننده) را بر طبق معنای تعریف شده برای عمل تغییر می‌دهد، یا اینکه اصلاً حالت هیچ یک از اشیاء دیگر را تغییر نمی‌دهد.

Attribute(syn. Field, Data Member, Instance Variable, Slot)

صفت

یک ویژگی تعریف شده برای یک رده که ارزش داده‌ای دارد. صفتها برای نگهداری حالت نمونه‌های یک رده به‌کار می‌روند. ارزشها از طریق صفتهای رده به نمونه‌ها مرتبط می‌شوند. ارزش مرتبط شده اغلب توسط یک عمل دارای یک پارامتر مشخص‌کنندهٔ شیء، معلوم می‌شود. صفتها در واقع خاصیتهای یک نوع را پیاده‌سازی می‌کنند.

حضّار **Audience**

نوع مشتری (فراخوانندهٔ) یک فاصل. ممکن است یک فاصل سیستم برای منظورهای زیر ساخته شده باشد: استفادهٔ کاربر نهایی سرویسهای سیستم (فاصل کارکردی)، استفادهٔ یک تابع مدیریت سیستم درون سیستم (فاصل مدیریت سیستم)، و یا استفادهٔ سایر سرویسهای سیستم به جهت ایجاد سرویسی جدید از طریق استفاده از سرویسهای موجود اشیاء متفاوت دیگر (فاصل ساختنی).

ردهٔ مبنا **Base Class**

رده‌ای که صفتها و متدهایش را ردهٔ دیگری به ارث می‌برد.

حامل **Bearer**

نوعی شیءکه بیانگر یک فاصل است. یک شیءمی‌تواند اساساً بدین طریق منحصراً توصیف شود که دارای یک فاصل خاص است (یک شیءخاص حامل یک فاصل است) یا اینکه یک شیءممکن است فاصلی داشته باشد که کمک به مقصود اصلی شیءدر جهت فراهم آوردن توانائیهای خاصی باشد (یک شیءعمومی حامل فاصل است).

رفتار **Behaviour**

تأثیر مشهود یک درخواست.

سازگاری رفتار **Behaviour Consistency**

اطمینان از اینکه اعمال یک رده، سازگاری حالت تعریف شدهٔ رده را نقض نمی‌کنند.

معماری **Architecture**

ساختار یا ترتیب مؤلفه‌ها در یک سیستم محاسباتی و یا غیرمحاسباتی. معماری یک کاربرد خاص توسط رده‌ها و روابط فیمابین رده‌ها مشخص و تعریف می‌شود. در یک سطح دیگر، معماری یک سیستم از طریق ترتیب قرارگرفتن مؤلفه‌های نرم‌افزاری و سخت‌افزاری مشخص می‌شود. اغلب از واژه «معماری منطقی» و «معماری فیزیکی» برای این تمایز دادن استفاده می‌شود.

Argument Type (syn. Formal Parameter)

نوع نشانوند

نوع یک مقدار که توسط متقاضی فراخوانی یک عمل بر روی یک شیءعرضه می‌شود. یک نوع نشانوند در امضاء یک عمل مشخص و تعریف می‌شود.

نشانوند **Argument(syn. Actual Parameter)**

ارزشی که توسط متقاضی فراخوانی یک عمل بر روی یک شیءعرضه می‌شود. همچنین یک شیءخاص که توسط یک رابطه به‌یک ارزش یا مجموعه‌ای از ارزشها نگاشت می‌شود.

ادعا **Assertion**

شرط یا گزاره‌ای که ارزش آن درست یا نادرست است

گمارش **Assignment**

فرارونده دستیابی به ارزشهای نمونه‌ای از یک رده، و کپی کردن ارزشها به فضای تخصیص‌یافته برای یک شیءدیگر. معنای کامل گمارش به زبان پیاده‌سازی بستگی دارد.

تداعی، متعلق ساختن **Association**

مکانیزمی برای پیوند زدن نمونه‌های نوعها به یکدیگر. یک تداعی از یک سری روابطی تشکیل شده است که نگاشتهای یک شیءبه شیءدیگر یا مجموعه‌ای از اشیاء دیگر را مشخص می‌کند. یک مورد معمول، تداعی بین دو نوع است که شامل روابط معکوس باشند. به‌عنوان مثال، نوعهای «شخص» و «کمپانی»، که روابط «کارفرما» و «مستخدم» بر روی نوعها تعریف شده است. این جفت رابطه‌ها، تداعی «استخدام» را تعریف می‌کنند.

درخواست ناهمگام **Asynchronous Request**

درخواستی که متقاضی، پس از ارسال درخواست، در انتظار دریافت جواب(ها) معطل نمی‌ماند.

واژه‌نامهٔ پیشنهادی شیءگرایی

توضیح: *این واژه‌نامه توسط آقای دکتر شریفی عضو هیأت علمی دانشگاه علم و صنعت ایران برای چاپ این ویژه‌نامه در اختیار انجمن قرارگرفته است. ضمن ارج نهادن به کار ارزشمند آقای دکتر شریفی به اطلاع خوانندگان گرامی می‌رساند که معادلهای پیشنهاد شده در این واژه‌نامه، نظر شخصی ایشان است. جا دارد زیرکمیتهٔ تخصصی شیءگرایی با مبنا قرار دادن این کار، نسبت به تهیه و تدوین یک واژه‌نامهٔ توصیفی جامع شیءگرایی اقدام نماید.*

فعالیت **Activity (syn. Function, Process)**

فراوندی که دوام آن قابل ملاحظه است.

کننده، کنشگر **Actor**

یک شیءفعال که بخش مهمّی از توانایی یک کاربرد را تشکیل می‌دهد. به‌عنوان مثال، ممکن است یک سیستم مدیریت رومیزی فقط یک شیءبه‌نام «آشغال‌دانی» داشته باشد. این شیءفعال توانایی عملیاتی کاملی را برای حذف فایلها از مدیریت رومیزی و سیستم زیرین فایلها فراهم می‌کند.

پارامتر واقعی **Actual Parameter (syn. Argument)**

ارزش ارائه شده به‌یک پارامتر صوری در یک تقاضا.

تحلیل **Analysis**

فراروند تعیین مشخصات که در آن فرم منطقی یک مسئله یا موضوع خاص درنظر گرفته می‌شود. تحلیل فقط ساختار یک موضوع را مشخص کرده و به‌پیاده‌ساز آن توجّهی ندارد.

کاربرد **Application**

یک برنامه یا مجموعه‌ای از برنامه‌های مرتبط نرم‌افزاری که توانایی مفیدی را در اختیار می‌گذارند.

تسهیلات کاربردی **Application Facilities**

تسهیلات مشترکی که داخل یک کاربرد خاص مفید فایده هستند.

اشیاء کاربردی **Application Objects**

کاربردها و مؤلفه‌های کاربردها که داخل یک سیستم شیءگرا اداره می‌شوند. آعمال نمونه‌ای که بر اینگونه اشیاء إعمال می‌شوند عبارتند از: 'Open'، 'Install'، 'Move'، و 'Remove'.

ردهٔ انتزاعی **Abstract Class**

رده‌ای که فقط می‌تواند به‌عنوان یک ردهٔ مبنای ردهٔ دیگری استفاده شود. نمی‌توان اشیای از یک ردهٔ انتزاعی خلق کرد، ولی می‌توان نمونه‌های زیر-رده‌های ردهٔ انتزاعی را ایجاد کرد. یک ردهٔ انتزاعی معمولاً برای تعریف کردن یک واسط مشترک برای تعدادی از زیر-رده‌ها به‌کار می‌رود.

نوع دادهٔ انتزاعی **Abstract Data Type (syn. ADT, Data Type)**

توضیحی از یک مجموعه از عناصر داده‌ای مشابه. یک ADT مکانیزمی برای تعریف کردن ساختارهای مورد استفاده در نگهداری داده‌ها را فراهم می‌کند. داده‌ها توسط مجموعه‌ای از اعمال محاط شده‌اند که به مقادیر داده دسترسی داشته و می‌توانند آنها را دستکاری کنند. نوعهای Integer، Float و String نمونه‌هایی از نوعهای انتزاعی داده هستند. یک پیاده‌سازی خاص از یک نوع انتزاعی داده را یک رده می‌گویند.

انتزاع **Abstraction (syn. Concept, Type)**

یک فرم منطقی که چیزهای مشابه، به‌اشتراک دارا هستند. همچنین، فراروند مشخص کردن ویژگیهای تمایزدهندهٔ مجموعه‌ای از اشیاء مشابه نتیجهٔ فراروند انتزاع یک نوع است.

کنش **Action**

عملی که دوام آن بسیار اندک بوده و قابل ملاحظه نباشد.

فعال‌سازی **Activation**

فراروند کپی کردن متدها و ارزشهای یک شیءبه‌یک فضای آدرس دهی قابل اجرا به‌جهت آماده‌سازی فراخوانی متدها بر روی اشیاء.

شیءفعّال **Active Object (syn. Actor)**

یک شیءکه حاوی یک فراروند برای خود است ـ مادامیکه شیءفعال در قید حیات است، فرآروند شیءفعّال است.

پیشگام در نشر آثار برگزیده علوم کاربردی کامپیوتر

مرجع فنی سیستم عامل MS-DOS 6

این کتاب برای کسانی است که اطلاعات فنی بیشتری در مورد ویژگیهای سیستم عامل MS-DOS 6 نیاز دارند. این کتاب عمدتاً به دو بخش تقسیم شده است و کاملترین راهنما برای فراگیری دستورات نگارش ششم MS-DOS و آشنایی با ویژگیهای تکنیکی آن به‌شمار می‌آید. شما در بخش اول با ویژگیهای تکنیکی DOS 6 و در بخش دوم با تمامی دستورات سیستم عامل MS-DOS 6 آشنا خواهید شد. برای هر دستور مثالهای متعددی ارائه شده تا با کاربردهای مختلف آن آشنا شوید. مترجم: فرهاد قلی زاده نوری ـ قطع: وزیری ـ ۴۱۶ صفحه ـ ۴۶۰ تومان

کتاب آموزشی MS-DOS 6 پیشرفته

با مطالعهٔ کتاب آموزشی MS-DOS 6 پیشرفته از سری کتب نرم‌افزاری پیتر نورتن نحوهٔ به‌حداکثررسانی کارایی ویژگیهای جدید DOS-6، از جمله DubleSpace، Backup، Anti-Virus و Undelete را فرا خواهید گرفت. پیتر نورتن در این مرجع بی‌نظیر، بهترین و عملی‌ترین تکنیکهای بهره‌برداری حداکثر از DOS را در اختیار شما قرار می‌دهد و شما می‌توانید با کشف و به‌کارگیری قابلیتهای نهفتهٔ DOS کارایی سیستم خود را به حداکثر برسانید. مترجم: مجید سماوی ـ قطع: وزیری ـ ۴۱۶ صفحه ـ ۴۶۰ تومان

مدیریت حافظه در Windows 3.1

صرفنظر از اینکه یک کاربر مبتدی یا باتجربه هستید مطالب زیادی می‌توانید برای استفادهٔ بهینهٔ حافظه فرا بگیرید! این کتاب شما را در فراگیری مطالب بنیادی حافظهٔ ویندوز یاری کرده و شما را در استفادهٔ بهینهٔ کامپیوتر راهنمایی می‌کند. با درک مفاهیم عملی حافظه یک زمینهٔ ایده‌آل برای استفادهٔ بهینه و خرید حافظه بدست خواهید آورد! قسمتی از کنترل حافظهٔ ویندوز در سطح DOS صورت می‌گیرد ـ بدین ترتیب با بهترین روشهای آماده‌سازی DOS آشنا خواهید شد. مترجم: فرهاد قلی‌زاده نوری ـ قطع: وزیری ـ ۲۲۴ صفحه ـ ۲۸۰ تومان

مدیریت حافظه در MS-DOS 6

این کتاب به شما چگونگی استفاده از MemMaker، Microsoft Diagnostic و دیگر ابزار حافظه را نشان داده و در مدیریت حافظه در PC به کمکتان می‌آید. در کتاب مطالب زیر مورد بحث قرار می‌گیرند: حافظه چیست و کامپیوتر چگونه از آن استفاده می‌کند ـ تفاوت بین حافظه‌های توسعه‌یافته، متعارف و اضافی ـ چگونگی بکارگیری برنامهٔ کمکی MemMaker برای بهینه‌سازی در سیستم به صورت اتوماتیک ـ وسعت بخشیدن به توانایی‌های MS-DOS و Windows. مترجم: محمد نوزری ـ قطع: وزیری ـ ۲۰۰ صفحه ـ ۱۸۰ تومان

برنامه‌نویسی فایلهای دسته‌ای در MS-DOS 6

این کتاب به شما می‌آموزد که چگونه از قدرت فایلهای دسته‌ای استفاده نمایید. همچنین قادر خواهید بود که با استفاده ازفایلهای دسته‌ای درسیستم عامل MS-DOS 6 بهمراه ۱۰۰ برنامه دسته‌ای آماده اجرا، دستورات جدیدی را ایجاد نمایید. با ایجاد فایلهای دسته‌ای چون فایلهای زیر در وقت خود صرفه‌جویی کنید: یک منوی شخصی ـ یک سیستم بایگانی خودکار ـ یک کتابخانه جهت محافظت فایلهای دسته‌ای در مقابل ویروسها. در انتهای هر بخش سؤالاتی برای آزمایش درصد پیشرفت شما در نظر گرفته شده است. مترجم: فرهاد قلی‌زاده نوری ـ قطع: وزیری ـ قیمت: ۵۴۰ تومان

خودآموز استفاده از EXCEL 5

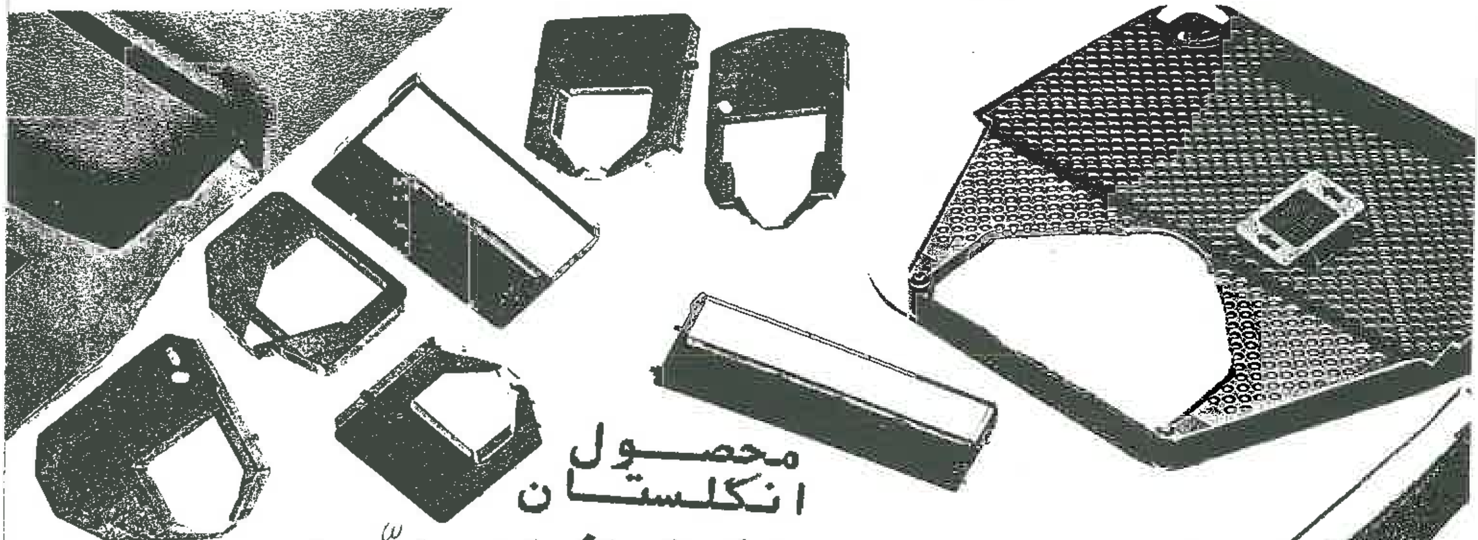
درسهای پیاپی میکروسافت اکسل نگارش ۵ امکانات ایده‌آلی را در اختیار شما قرار می‌دهند. این سیستم آموزشی با مثالهای متعدد و کارا به شما می‌آموزد تا بتوانید مطالب مورد نیازتان را با سرعتی مطلوب نظر خود بیاموزید. ویژگیهای این کتاب از جمله: سفارشی ساختن گزارشها، تجزیه تحلیل و به اشتراک گذاری داده‌ها و سفارشی ساختن محیط کاری میکروسافت اکسل بطور دلخواه به شما ارائه می‌گردد تا بتوانید کارایی برنامه صفحه گستردهٔ خود را افزایش دهید. مترجم: مجید سماوی ـ قطع: وزیری ـ ۳۲۰ صفحه ـ ۴۶۰ تومان

و بیش از صد عنوان کتاب الفورماتیک دیگر

منتظر کتابهای تخصصی دیگر ما باشید

فروش در کلیه کتابفروشی‌ها و فروشگاههای معتبر کامپیوتری سراسر کشور

کانون نشر علوم
تهران ـ صندوق پستی ۷۱۸ـ۱۷۱۸۵



؟؟؟؟؟ بهترین تبلیغ، اثبات ادّعاست!!!!

درصورت انصراف ، کالا را تا **ده روز** سالم پس فرستاده، **پول** آنرا دریافت دارید.

کیت کامل شارژ پانترجهت نوسازی ویا تغییررنگ دادن ریبون انواع پرینتر، تایپ، تلکس،....

این کیت شامل: دستگاه شارژر،نگهدارنده دستگاه، ریبون گردان، چهار عدد دستکش، یک بطر(40 cc) جوهر مخصوص مشکی، قطره چکان، لاک ویژه و بهمراه راهنمای فارسی - انگلیسی میباشد.

- محل برش

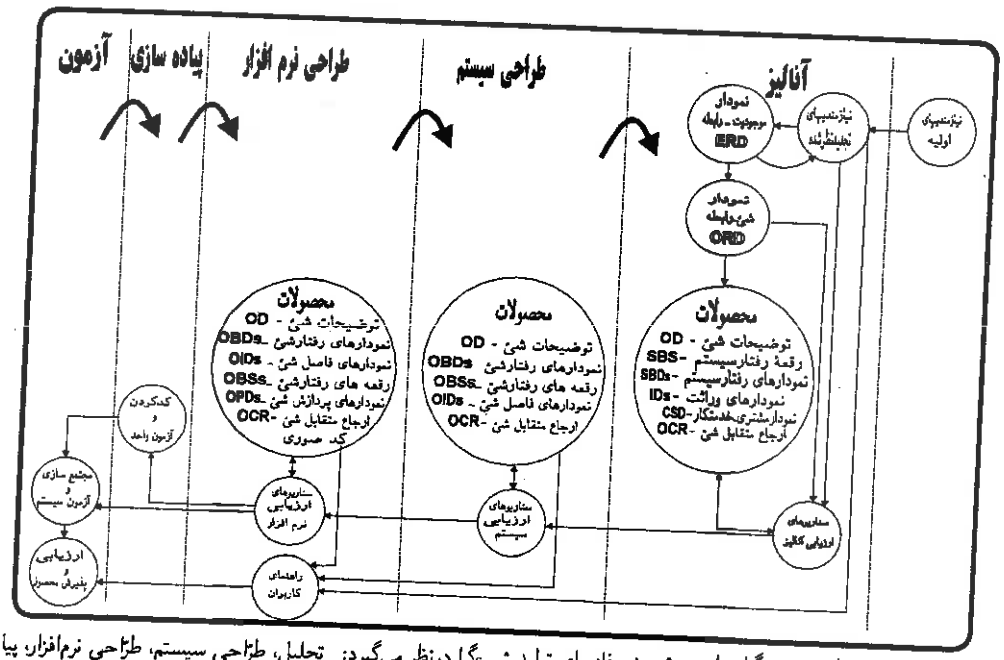
اینجانب....... بادرس ....

کدپستی........تلفن.............مایل بدریافت سفارش زیر میباشم .

وجه سفارش از شعبه .... بحساب ۶۸۷۸۲ (ناغانی) نزد بانک صادرات شعبه یک بازارتهران واریزو اصل حواله ضمیمه شده است.

کیت کامل پانتر...عدد(۱۱۰۰ تومان) جمعا.... تومان+ جوهر اضافه مشکی ...عدد (۲۰۰ تومان هر بطر40 cc) جمعا.... تومان + جوهر یدک رنگی (سرخ،سبز،آبی،زرد)....عدد برنگهای.................. (۳۰۰ تومان هر بطر 40 cc) جمعا.... تومان + هزینه پست ۱۰۰ تومان. جمع کل.......... تومان.

لطفا کوپن پرشده را بااصل حواله بادرس: تهران صندوق پستی۱۴۵-۱۱۱۷۵ (ناغانی) ارسال نمائید.



شکل۱: منظر کلّی فرارونـد تولید در متدولوژی شیءگرا. این روش پنج فاز برای تولید شیءگرا درنظر می‌گیرد: تحلیل، طرّاحی سیستم، طرّاحی نرم‌افزار، پیاده‌سازی و آزمون.

مقایسه با سایر متدولوژیهای تولید، از پتانسیل بیشتری برای مجتمع‌سازی فرارونـد تولید، و همچنین برای تولید سیستمهای نرم‌افزاری انعطاف‌پذیرتر و با قابلیت نگهداری بهتر، برخوردار است.

از متدولوژی پیشنهادی برای ایجاد یک آزمایشگاه شبیه‌سازی و نمونه‌سازی **سیستمهای نرم‌افزاری کلان و حیاتی** استفاده خواهد شد. در حال حاضر یک سیستم **کنترل ترافیک هوایی (کته)** انتخاب شده و مفاهیم پیچیدهٔ این سیستم با استفاده از این متدولوژی مورد بررسی و ارزیابی قرار گرفته و مشخصّات آنها تعیین و با استفاده از سیستم نشانگذاری متدولوژی پیشنهادی نمایش داده شده‌اند. به‌طور کلّی، پیشنهاد متدولوژی جدید نتیجهٔ این کار بوده و اَشکال نمایش داده شده در این مقاله نیز از مستندات ایجاد شده به‌هنگام اعمال این متدولوژی به‌سیستم **کته** (یا به‌عبارت دیگر، شبیه‌سازی و نمونه‌سازی مفاهیم سیستم **کته**) به‌دست آمده‌اند.

متدولوژی پیشنهادی یک سیستم را از ابعاد مختلف، بالاخص در فازهای تحلیل و طرّاحی، بررسی و به‌تصویر کشیده و باعث فهم بهتر و بیشتر از محتوی سیستم (بُعد اطلاعاتی) و رفتار سیستم (بُعد فعل و انفعالی) می‌شود. این متدولوژی ضرورت تصدیق[11] فازهای مختلف فرارونـد تولید را نیز مورد تأکید قرار داده، و انعطاف‌پذیری لازم برای ایجاد یک فرارونـد چرخه‌ای و تکراری تولید را فراهم می‌کند. به‌علاوه ضرورت وجود یک محیط پیچیدهٔ CASE را برای تسهیل توضیح دادن و بررسی سیستم از ابعاد مختلف، و همچنین فراهم آوردن چارچوبی کلّی برای برقرار نمودن سازگاری در سرتاسر این ابعاد مختلف، مورد تأکید قرار داده و نشان می‌دهد.

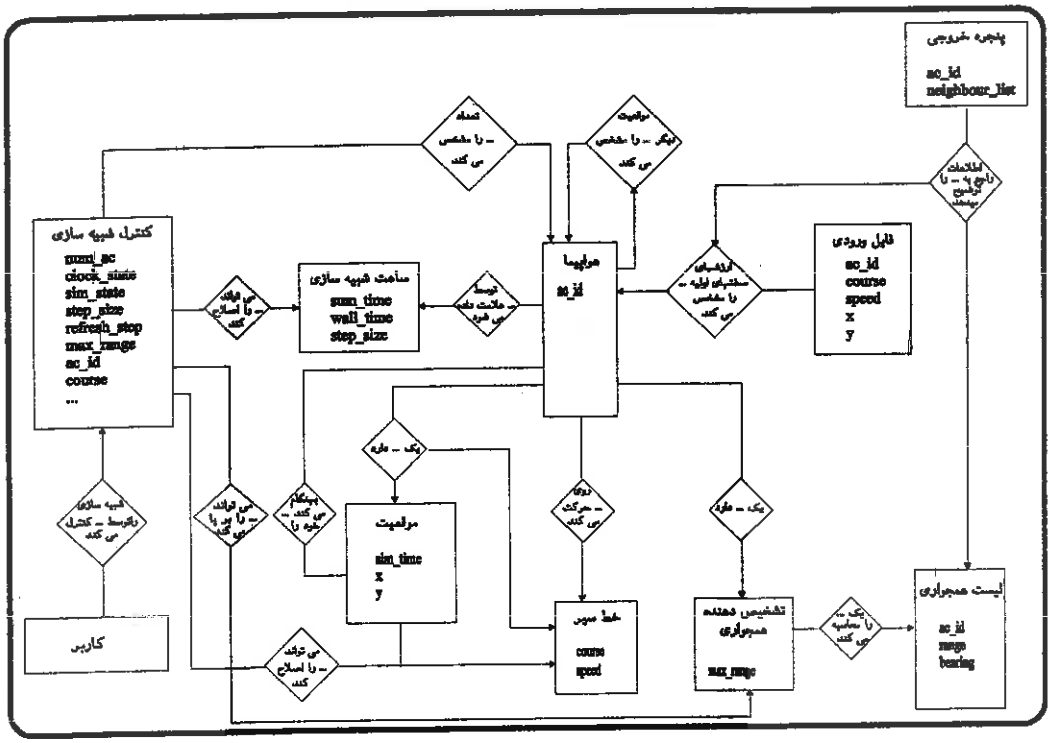
11) Validation

سیستم کنترل ترافیک هوایی

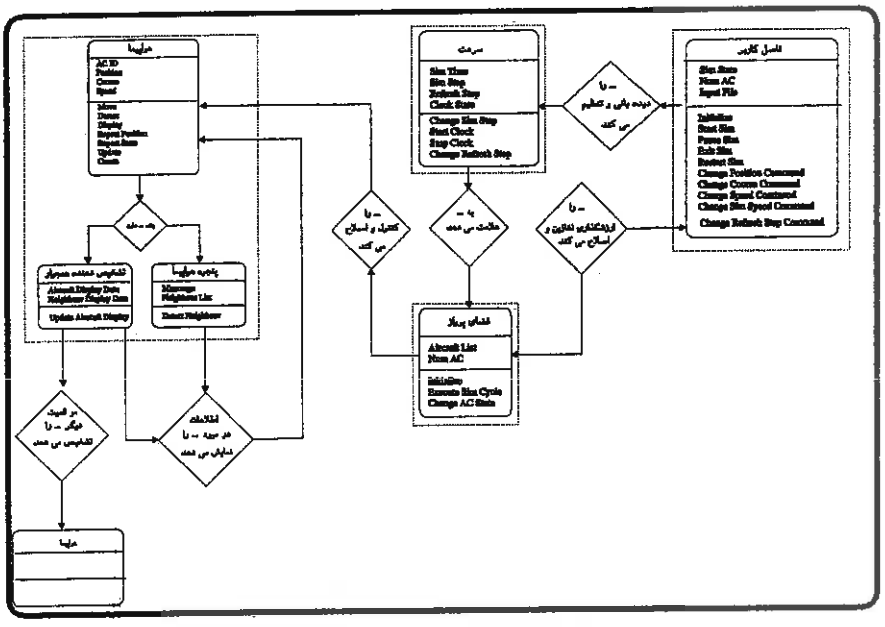
برای آشنایی خواننده با مفاهیم به‌تصویرکشیده شده در مقاله، سیستم متصوّرهٔ **کته** در اینجا اجمالاً توضیح داده می‌شود. همانطور که از اختصار **کته** قابل استنباط است، چنین سیستمی فضای پروازی هواپیماها را دیده‌بانی کرده و از طریق جداسازی فضای پروازی هواپیماها، این فضا را کنترل می‌کند. در حال حاضر جداسازی فضا توسط افراد متخصّص **کته** به‌صورت دستی انجام گرفته و از تجهیزات الکترونیکی و مخابراتی اغلب فقط برای تعیین پارامترهایی که به‌صورت دستی اِعمال می‌شوند استفاده می‌شود.

نمونه‌سازی و شبیه‌سازی مفاهیم سیستم **کته** که محصول آن تحت یک متدولوژی جدید در این مقاله پیشنهاد شده است، در راستای مکانیزه کردن این سیستم انجام گرفته است. سیستم مکانیزهٔ **کته** قادر خواهد بود بخشی از فعّالیتهای جداسازی را مکانیزه نموده، و ضمن بهره‌برداری از پیشرفتهای تکنولوژیکی فعلی و آتی تجهیزات فضایی هواپیماها، متخصّصین **کته** را از اپراتورهای جداسازی صرفاً دستی فضای پروازی به‌مدیرانی مبدّل کند که مسئولیت مدیریت گنجایش و توان عملیّاتی فضای پروازی را عهده‌دار باشند. لذا آنچه به‌نام یک سیستم **کته** در این مقاله مورد نظر است، سیستمی **مکانیزه** است که قادر به‌مدیریت و دستگیری مؤثّرتر، امنتر، کم خرجتر، و سهلتر سطوح رو به‌افزایش ترافیکی و توانائیهای پیشرفته و گوناگون هواپیماها باشد.

برای ساختن یک چنین سیستم مکانیزهٔ حیاتی وکلان، نمونه‌سازی و شبیه‌سازی ویژگیهای عملیّاتی و غیر عملیّاتی (یا به‌عبارت دیگر، ویژگیهای وظیفه‌مند و غیروظیفه‌مند) خود سیستم مکانیزه، و همچنین ویژگیهای محیطی که سیستم در آنجا عملیّاتی خواهد شد، امری ضروری به‌نظر می‌رسد. این سیستم



شکل۲: نمودار موجودیت ـ رابطه (ERD). نمودار ERD مدلی از سیستم است که به‌همراه فعالیتهای تحلیل نیازمندیها و تحلیل وقایع جهت تشخیص این‌که قلمرو مسئله چه می‌باشد، به‌کار می‌رود. نمودار ERD مدلی مسطّح بوده و هیچگونه ساختار سلسله مراتبی را نشان نمی‌دهد.



شکل۳: نمودار شیء- رابطه (ORD).

از مؤلفه‌های سخت‌افزاری و نرم‌افزاری متعدّدی تشکیل می‌شود که در یک محیط توزیع یافتهٔ چند کاربره عمل می‌کنند. ایجاد چنین سیستم نرم‌افزاری یک کارکلان مهندسی نرم‌افزار بوده و به‌اعمال یک متدولوژی منظّم و در عین حال انعطاف‌پذیر برای فرایند تولید نرم‌افزار، نیاز دارد. متدولوژی شیءگرا برتریهای زیادی نسبت به‌سایر متدولوژیها داشته و لذا برای ایجاد چنین سیستمی مناسب تشخیص داده شده است.

متدولوژی پیشنهادی برای تولید به‌روش شیءگرایی

متدولوژی و فرایند تولید پیشنهادی ترکیبی از تکنیکها و سیستمهای نشانگذاری موجود می‌باشد. البتّه، این متدولوژی چیزی بیش از صرفاً جمع بخشهایش را ارائه می‌کند. این متدولوژی قابلیت ردّیابی و تداوم کار را در میان چرخهٔ حیات تولید فراهم می‌کند، عمدتاً بدین دلیل که مستندّات و نمودارهای ایجاد شده در فازهای اوّلیهٔ تولید، در مراحل بعدی تولید پالایش و متکامل می‌شوند. مزیّت دیگر این متدولوژی اینست‌که گذار ملایم از یک فاز به‌فاز بعدی تولید را امکانپذیر می‌سازد. همچنین منظرهای چندگانهٔ سیستم در دست تولید را نمایان ساخته و بدینوسیله فهم بیشتر سیستم را توسط تحلیل‌گران، طرّاحان و پیاده‌سازان سیستم مقدور می‌سازد. نهایتاً، مکانیزمی برای تصدیق مدلهای ایجاد شده در هر فاز را، جهت تأیید درستی محصولات تولید شدهٔ آن فاز، فراهم می‌کند.

همانطور که در شکل شمارهٔ ۱ نشان داده شده است، این متدولوژی پنج فاز برای تولید شیءگرا در نظر می‌گیرد: **تحلیل، طرّاحی سیستم، طرّاحی نرم‌افزار، پیاده‌سازی و آزمون.** فازهای پیاده‌سازی و آزمون در این مقاله مورد بحث قرار نگرفته‌اند (توضیح در مورد آزمون و برنامه‌سازی شیءگرا را می‌توان در منابع دیگر [۷،۸] یافت). این فازها شبیه فازهای تعریف شده در روشهای سنّتی تولید هستند. پیکانهای چپ ـ به‌راست در شکل شمارهٔ ۱، بیانگر این مطلب هستند که فرایند تولید شامل چرخش بین فازها می‌باشد ـ به‌عبارت دیگر، بازگشت به‌فازهای قبلی به‌منظور پالایش ایده‌ها یا تغییر سازماندهی امری غیرمنتظره نمی‌باشد.

فاز تحلیل، سیستم را به‌عنوان راه‌حلّی به‌یک مسئله در محیط یا قلمرو مسئله تلقّی می‌کند. در این فاز از تولید، اشیاء اصلی سیستم و قلمرو سیستم مشخّص می‌شوند. روابط فیمابین این اشیاء، به‌همراه صفات[12] و رفتارهای[13] آنها نیز مشخّص می‌شوند. ملاحظات پیاده‌سازی مطلقاً در نظر گرفته نمی‌شوند ـ به‌عنوان مثال، حافظه و قدرت نامحدود فرض شده است. زمانی که فاز تحلیل به‌پایان خود نزدیک می‌شود، سناریوهای ارزیابی تحلیل برای تأیید مدل ایجاد شده از سیستم در این فاز تهیّه می‌شوند. **سناریوهای ارزیابی تحلیل**[14]، وضعیّتهای ساده‌ای را در نظر می‌گیرند تا مشخّص کنند که آیا مدل تحلیل کلیهٔ نیازمندیهای اصلی سیستم را به‌حدّ کافی پوشش می‌دهد یا خیر.

12) Attributes 13) Behaviours 14) Analysis Evaluation Scenarios

فاز طرّاحی سیستم توجّه خود را به‌نحوهٔ کار سیستم متمرکز می‌کند. دو فعّالیت عمده در این فاز رخ می‌دهند. اوّلین فعّالیت، مشخّص نمودن جزئیات رفتاری اشیاء سیستم و فعل و انفعالات فیمابین اشیاء است. این عمل، پالایش اشیاء مشخّص شده در زمان تحلیل سیستم است. فعّالیت دوّم عبارتست از مشخّص نمودن اشیاء بیشتری که وجودشان برای کارکردن سیستم مورد نیاز است. همچون تحلیل، طرّاحی سیستم صرف‌نظر از ملاحظات پیاده‌سازی صورت می‌گیرد. به‌علاوه، سناریوهای ارزیابی نیز جهت طرّاحی سیستم تهیّه می‌شوند. **سناریوهای ارزیابی طرّاحی**[15]، روایتهای پالایش یافتهٔ سناریوهایی هستند که در فاز تحلیل تهیه شده‌اند. سناریوهای ارزیابی طرّاحی مشخّص می‌کنند که آیا تمامی مکانیزمهای لازم برای انجام نیازمندیهای سیستم در مدل طرّاحی سیستم وجود دارند یا خیر.

فاز طرّاحی نرم‌افزار، چگونگی پیاده‌سازی یک سیستم را با استفاده از یک زبان برنامه‌سازی خاصّ بر روی یک جایگاه خاصّ سخت‌افزاری و یک محیط خاصّ نرم‌افزاری، مورد توجّه قرار می‌دهد. فعّالیت این فاز عمدتاً عبارت از اضافه نمودن جزئیات پیاده‌سازی به‌اشیاء تعریف شده سیستم در فازهای تحلیل و طرّاحی سیستم است. همچون فازهای دیگر، سناریوهای ارزیابی برای تصدیق طرّاحی نرم‌افزار تهیّه می‌شوند. **این سناریوهای ارزیابی نرم‌افزار**[16]، روایتهای پالایش شدهٔ سناریوهای ارزیابی طرّاحی سیستم بوده و مجموعه‌ای کامل از داده‌های آزمون سیستم را نیز ارائه می‌کنند. انتظار می‌رود که پیاده‌سازی سیستم از محصولات تولید شدهٔ فاز طرّاحی نرم‌افزار سهل و آسان باشد.

در هر نوع روش و متدولوژی تولید، بازبینیهای[17] مکرّر لازم هستند. پس از اتمام هر یک از فازهای تحلیل، طرّاحی سیستم و طرّاحی نرم‌افزار، حداقل یکبار بازبینی لازم است. سناریوهای ارزیابی نقش عمده‌ای در این بازبینیها ایفا می‌کنند.

فازهای تولید و مستندّات آنها در ادامهٔ مقاله توضیح داده شده‌اند. مستندّات به‌صورت توضیحات متن‌وار و نمودارها هستند. نمونه‌هایی نیز در قالب اشکال ارائه شده‌اند.

فاز تحلیل

فاز تحلیل بر تشخیص اشیاء اصلی موجود در سیستم و قلمرو سیستم، و همچنین روابط فیمابین این اشیاء، توجّه دارد. تحلیل پاسخگوی این سؤال است که: چه چیزی در سیستم و قلمرو آن وجود دارد؟ . در پاسخ دادن به‌این سؤال، سه تکنیک متعارف در تولید به‌روشهای سنّتی استفاده شده‌اند: **تحلیل نیازمندیها، تحلیل اطّلاعات و تحلیل وقایع (یا کنشها).** تحلیل نیازمندیها ابهامات و نقصهای موجود در اظهارات نیازمندیها را مشخّص نموده و منجر به‌اظهارات تجدید نظر شده‌ای از نیازمندیها می‌شود. تحلیل اطّلاعات، اطّلاعات یا داده‌های موجود سیستم را مشخّص می‌کند. در

15) Design Evaluation Scenarios 16) Software Evaluation Scenarios 17) Reviews

مقصود:	شیء هواپیما حرکت هواپیما را برروی یک خط سیر شبیه سازی نموده و اعمالی را فراهم‌می‌کند که توسط آنها صفتهای غیرهویتی شیء‌می‌توانند تغییر داده‌شوند. سرویسهای عمومی ارائه‌شده عبارتند از: ارزشگذاری آغازین شیء، گزارش کردن موقعیت هواپیما، حرکت دادن هواپیما، تشخیص دادن هواپیماهای همجوار، نمایش اطّلاعات در مورد هواپیما و لیست همجواری، تغییر ارزشهای صفتهای شیء، و برگرداندن اطّلاعات در مورد ارزشهای صفتها. سرویسهای خصوصی در واقع عمل تغییر ارزشهای صفتهای شیء را انجام می‌دهند.
ریز ـ شیء از:	{}
زیر شیءها (رابطهٔ دارد ـ یک):	پنجرهٔ هواپیما، تشخیص دهندهٔ همجوار
وراثت/بالا رده (رابطهٔ هست ـ یک):	{}
صفتها:	AC ID
	Position
	Course
	Speed
	*Neighbour - List
	*AC - List
سرویسهای ارائه شده:	Create()
	Move()
	Detect()
	Display()
	Report - Position()
	Report - State()
	Update()
سرویسهای مورد نیاز:	Detect (از شیء تشخیص دهندهٔ همجوار)
	Display (از شیء پنجرهٔ هواپیما)
تعاریف سرویسها:	"در زمان انجام فاز طرّاحی سیستم اضافه خواهد شد"

شکل۴: توضیح شیء(OD).

تولید شیءگرا، این به‌معنای یافتن و مشخّص نمودن اشیاء و صفتهای این اشیاء می‌باشد. تحلیل وقایع، رفتار اشیاء سیستم را مشخص می‌کند. تحلیل نیازمندیها، تحلیل اطّلاعات و تحلیل وقایع تا حدّ زیادی به‌هم مرتبط بوده و خروجی آنها می‌تواند اساسی برای دستیابی به‌موافقت استفاده‌کنندگان سیستم دربارهٔ قلمرو و محتوای سیستم باشد. اگر چه امکان دارد که این سه فعّالیت متعاقب یکدیگر انجام بگیرند، ولی چنین انتظار می‌رود که آنها به‌صورت همزمان اجرا شده و نتایج هرکدام تصدیقی بر نتایج سایر فعّالیتها باشد. نتیجهٔ فاز تحلیل مدلی از سیستم، در قلمرو سیستم است. این مدل شامل اشیاء، صفتها و رفتار اشیاء، و روابط فیمابین اشیاء می‌باشد.

تحلیل نیازمندیها

هدف از تحلیل نیازمندیها، ایجاد اظهار روشنی از مقصود، حوزه و وظایف خواسته شدهٔ سیستم است. این اظهار در شرایط ایده‌آل توسط مشتری (استفاده کنندهٔ سیستم) تهیّه می‌شود. ولی در واقع غالباً چنین اظهاری تهیّه نشده و یا اگر بشود، ناسازگار و مبهم است. دو سؤال عمده که تحلیل‌گر بایستی به‌هنگام انجام تحلیل نیازمندیها مطرح نماید از این قرار می‌باشند:

- به‌دانستن چه چیزی دربارهٔ قلمرو مسئله احتیاج است؟
- چه منابع اطّلاعاتی در مورد قلمرو مسئله وجود دارند؟

از مؤلفه‌های سخت‌افزاری و نرم‌افزاری متعدّدی تشکیل می‌شود که در یک محیط توزیع یافتهٔ چندکاربره عمل می‌کنند. ایجاد چنین سیستم نرم‌افزاری یک کارکلان مهندسی نرم‌افزار بوده و به‌اِعمال یک متدولوژی منظّم و در عین حال انعطاف‌پذیر برای فرایند تولید نرم‌افزار، نیاز دارد. متدولوژی شیءگرا برتریهای زیادی نسبت به‌سایر متدولوژیها داشته و لذا برای ایجاد چنین سیستمی مناسب تشخیص داده شده است.

متدولوژی پیشنهادی برای تولید به‌روش شیءگرایی

متدولوژی و فرایند تولید پیشنهادی ترکیبی از تکنیکها و سیستمهای نشانگذاری موجود می‌باشد. البتّه، این متدولوژی چیزی بیش از صرفاً جمع بخشهایش را ارائه می‌کند. این متدولوژی قابلیت ردّیابی و تداوم کار را در میان چرخهٔ حیات تولید فراهم می‌کند، عمدتاً بدین دلیل که مستندّات و نمودارهای ایجاد شده در فازهای اوّلیهٔ تولید، در مراحل بعدی تولید پالایش و متکامل می‌شوند. مزیّت دیگر این متدولوژی اینستکه گذار ملایم از یک فاز به‌فاز بعدی تولید را امکانپذیر می‌سازد. همچنین منظرهای چندگانهٔ سیستم در دست تولید را نمایان ساخته و بدینوسیله فهم بیشتر سیستم را توسط تحلیل‌گران، طرّاحان و پیاده‌سازان سیستم مقدور می‌سازد. نهایتاً، مکانیزمی برای تصدیق مدلهای ایجاد شده در هر فاز را، جهت تأیید درستی محصولات تولید شدهٔ آن فاز، فراهم می‌کند.

همانطور که در شکل شمارهٔ ۱ نشان داده شده است، این متدولوژی پنج فاز برای تولید شیءگرا در نظر می‌گیرد: **تحلیل، طرّاحی سیستم، طرّاحی نرم‌افزار، پیاده‌سازی و آزمون.** فازهای پیاده‌سازی و آزمون در این مقاله مورد بحث قرار نگرفته‌اند (توضیح در مورد آزمون و برنامه‌سازی شیءگرا را می‌توان در منابع دیگر [۷،۸] یافت). این فازها شبیه فازهای تعریف شده در روشهای سنّتی تولید هستند. پیکانهای چپ ـ به‌راست در شکل شمارهٔ ۱، بیانگر این مطلب هستند که فرایند تولید شامل چرخش بین فازها می‌باشد ـ به‌عبارت دیگر، بازگشت به‌فازهای قبلی به‌منظور پالایش ایده‌ها یا تغییر سازماندهی امری غیرمنتظره نمی‌باشد.

فاز تحلیل، سیستم را به‌عنوان راه‌حلّی به‌یک مسئله در محیط یا قلمرو مسئله تلقّی می‌کند. در این فاز از تولید، اشیاء اصلی سیستم و قلمرو سیستم مشخّص می‌شوند. روابط فیمابین این اشیاء، به‌همراه صفات[12] و رفتارهای[13] آنها نیز مشخّص می‌شوند. ملاحظات پیاده‌سازی مطلقاً در نظر گرفته نمی‌شوند ـ به‌عنوان مثال، حافظه و قدرت نامحدود فرض شده است. زمانی که فاز تحلیل به‌پایان خود نزدیک می‌شود، سناریوهای ارزیابی تحلیل برای تأیید مدل ایجاد شده از سیستم در این فاز تهیّه می‌شوند. **سناریوهای ارزیابی تحلیل**[14]، وضعیّتهای ساده‌ای را در نظر می‌گیرند تا مشخّص کنند که آیا مدل تحلیل کلیهٔ نیازمندیهای اصلی سیستم را به‌حدّ کافی پوشش می‌دهد یا خیر.

12) Attributes 13) Behaviours 14) Analysis Evaluation Scenarios

فاز طرّاحی سیستم توجّه خود را به‌نحوهٔ کار سیستم متمرکز می‌کند. دو فعّالیت عمده در این فاز رخ می‌دهند. اوّلین فعّالیت، مشخّص نمودن جزئیات رفتاری اشیاء سیستم و فعل و انفعالات فیمابین اشیاء است. این عمل، پالایش اشیاء مشخّص شده در زمان تحلیل سیستم است. فعّالیت دوّم عبارتست از مشخّص نمودن اشیاء بیشتری که وجودشان برای کارکردن سیستم مورد نیاز است. همچون تحلیل، طرّاحی سیستم صرف‌نظر از ملاحظات پیاده‌سازی صورت می‌گیرد. به‌علاوه، سناریوهای ارزیابی نیز جهت طرّاحی سیستم تهیّه می‌شوند. **سناریوهای ارزیابی طرّاحی**[15]، روایتهای پالایش یافتهٔ سناریوهایی هستند که در فاز تحلیل تهیه شده‌اند. سناریوهای ارزیابی طرّاحی مشخّص می‌کنند که آیا تمامی مکانیزمهای لازم برای انجام نیازمندیهای سیستم در مدل طرّاحی سیستم وجود دارند یا خیر. فاز طرّاحی نرم‌افزار، چگونگی پیاده‌سازی یک سیستم را با استفاده از یک زبان برنامه‌سازی خاصّ بر روی یک جایگاه خاصّ سخت‌افزاری و یک محیط خاصّ نرم‌افزاری، مورد توجّه قرار می‌دهد. فعّالیت این فاز عمدتاً عبارت از اضافه نمودن جزئیات پیاده‌سازی به‌اشیاء تعریف شده سیستم در فازهای تحلیل و طرّاحی سیستم است. همچون فازهای دیگر، سناریوهای ارزیابی برای تصدیق طرّاحی نرم‌افزار تهیّه می‌شوند. این **سناریوهای ارزیابی نرم‌افزار**[16]، روایتهای پالایش شدهٔ سناریوهای ارزیابی طرّاحی سیستم بوده و مجموعه‌ای کامل از داده‌های آزمون سیستم را نیز ارائه می‌کنند. انتظار می‌رود که پیاده‌سازی سیستم از محصولات تولید شدهٔ فاز طرّاحی نرم‌افزار سهل و آسان باشد.

در هر نوع روش و متدولوژی تولید، بازبینیهای[17] مکرّر لازم هستند. پس از اتمام هر یک از فازهای تحلیل، طرّاحی سیستم و طرّاحی نرم‌افزار، حداقل یکبار بازبینی لازم است. سناریوهای ارزیابی نقش عمده‌ای در این بازبینیها ایفا می‌کنند.

فازهای تولید و مستندّات آنها در ادامهٔ مقاله توضیح داده شده‌اند. مستندّات به‌صورت توضیحات متن‌وار و نمودارها هستند. نمونه‌هایی نیز در قالب اشکال ارائه شده‌اند.

فاز تحلیل

فاز تحلیل بر تشخیص اشیاء اصلی موجود در سیستم و قلمرو سیستم، و همچنین روابط فیمابین این اشیاء، توجّه دارد. تحلیل پاسخگوی این سؤال است که: چه چیزی در سیستم و قلمرو آن وجود دارد؟ . در پاسخ دادن به‌این سؤال، سه تکنیک متعارف در تولید به‌روشهای سنّتی استفاده شده‌اند: **تحلیل نیازمندیها، تحلیل اطّلاعات و تحلیل وقایع (یا کنشها).** تحلیل نیازمندیها ابهامات و نقصهای موجود در اظهارات نیازمندیها را مشخّص نموده و منجر به‌اظهارات تجدید نظر شده‌ای از نیازمندیها می‌شود. تحلیل اطّلاعات، اطّلاعات یا داده‌های موجود سیستم را مشخّص می‌کند. در

15) Design Evaluation Scenarios 16) Software Evaluation Scenarios 17) Reviews

مقصود:	شیء هواپیما حرکت هواپیما را برروی یک خط سیر شبیه سازی نموده و اعمالی را فراهم‌می‌کند که توسط آنها صفتهای غیرهویتی شیءمی‌توانند تغییر داده‌شوند. سرویسهای عمومی ارائه‌شده عبارتند از: ارزشگذاری آغازین شیء، گزارش کردن موقعیت هواپیما، حرکت دادن هواپیما، تشخیص دادن هواپیماهای همجوار، نمایش اطّلاعات در مورد هواپیما و لیست همجواری، تغییر ارزشهای صفتهای شیء، و برگرداندن اطّلاعات در مورد ارزشهای صفتها. سرویسهای خصوصی در واقع عمل تغییر ارزشهای صفتهای شیء را انجام می‌دهند.
ریز ـ شیء از:	{}
زیر شیءها (رابطهٔ دارد ـ یک):	پنجرهٔ هواپیما، تشخیص دهندهٔ همجوار
وراثت/بالا رده (رابطهٔ هست ـ یک):	{}
صفتها:	AC ID Position Course Speed *Neighbour - List *AC - List
سرویسهای ارائه شده:	Create() Move() Detect() Display() Report - Position() Report - State() Update()
سرویسهای مورد نیاز:	Detect (از شیء تشخیص دهندهٔ همجوار) Display (از شیء پنجرهٔ هواپیما)
تعاریف سرویسها:	"در زمان انجام فاز طرّاحی سیستم اضافه خواهد شد"

شکل۴: توضیح شیء(OD).

تولید شیءگرا، این به‌معنای یافتن و مشخّص نمودن اشیاء و صفتهای این اشیاء می‌باشد. تحلیل وقایع، رفتار اشیاء سیستم را مشخص می‌کند. تحلیل نیازمندیها، تحلیل اطّلاعات و تحلیل وقایع تا حدّ زیادی به‌هم مرتبط بوده و خروجی آنها می‌تواند اساسی برای دستیابی به‌موافقت استفاده‌کنندگان سیستم دربارهٔ قلمرو و محتوای سیستم باشد. اگر چه امکان دارد که این سه فعّالیت متعاقب یکدیگر انجام بگیرند، ولی چنین انتظار می‌رود که آنها به‌صورت همزمان اجرا شده و نتایج هر کدام تصدیقی بر نتایج سایر فعّالیتها باشد. نتیجهٔ فاز تحلیل مدلی از سیستم، در قلمرو سیستم است. این مدل شامل اشیاء، صفتها و رفتار اشیاء، و روابط فیمابین اشیاء می‌باشد.

تحلیل نیازمندیها

هدف از تحلیل نیازمندیها، ایجاد اظهار روشنی از مقصود، حوزه و وظایف خواسته شدهٔ سیستم است. این اظهار در شرایط ایده‌آل توسط مشتری (استفاده کنندهٔ سیستم) تهیّه می‌شود. ولی در واقع غالباً چنین اظهاری تهیّه نشده و یا اگر بشود، ناسازگار و مبهم است. دو سؤال عمده که تحلیل‌گر بایستی به‌هنگام انجام تحلیل نیازمندیها مطرح نماید از این قرار می‌باشند:

- به‌دانستن چه چیزی دربارهٔ قلمرو مسئله احتیاج است؟
- چه منابع اطّلاعاتی در مورد قلمرو مسئله وجود دارند؟

در پاسخ دادن به‌سؤال اوّل، تحلیل‌گر قلمرو سیستم را مشخّص می‌نماید. جوابهای به‌سؤال دوّم می‌توانند شامل مشتری سیستم، تجربهٔ خود تحلیل‌گر و سایر منابع منحصر به‌فرد قلمرو مسئله باشند. محصول تحلیل نیازمندیها اظهار روشنی از مقصود، حوزه و وظایف سیستم بوده و معمولاً پالایشی از اظهارات اوّلیهٔ نیازمندیها می‌باشد (توضیحات مفصلتر راجع به‌تحلیل نیازمندیها در منابعی همچون [۹] یافت می‌شود).

تحلیل اطّلاعات

هدف از تحلیل اطّلاعات مدل‌سازی موجودیّتهای قلمروی مسئله و روابط فیمابین این موجودیّتها است. موجودیّتها، عناصر داده‌ای هستند که توضیح می‌دهند چه چیزی در قلمرو مسئله وجود دارد. هدف ابتدائی، فقط مشخّص نمودن اینستکه چه عناصر داده‌ای در قلمرو مسئله وجود دارند و چه روابطی فیمابین این عناصر برقرار است. به‌دلیل آنکه تحلیل اطّلاعات فراروندی بازجویانه است، تحلیل‌گر می‌بایستی کلیهٔ منابع موجود را جهت مشخّص کردن موجودیّتهای قلمرو مسئله به‌کار گیرد.

این مدل در یک نمودار **موجودیت ـ رابطه**[18] (ERD) به‌صورت گرافیکی در شکل ۲ نمایش داده شده است. به‌این نمودار یک **مدل اطّلاعات**[19] نیز اطلاق می‌شود. مدل اطّلاعات یک مدل مسطّح است که سعی در نشان دادن هیچگونه ساختار سلسله مراتبی ندارد. نمودار ERD مدلی است که به‌همراه فعّالیتهای تحلیل نیازمندیها و تحلیل وقایع برای تشخیص اینکه چه چیزی در قلمرو مسئله وجود دارد، به‌کار می‌رود.

پس از تشخیص موجودیّتها و نشان دادن آنها در نمودار ERD، بایستی موجودیّتها به‌اشیاء نگاشت شوند. این عمل از طریق اینکه چه شیئی را هر موجودیّت به‌بهترین نحو توضیح می‌دهد انجام می‌گیرد. سؤالی که بایستی مطرح شود اینستکه: یک عنصر داده‌ای بهترین بیانگر (توضیح‌دهندهٔ) کدام شیء‌است؟ عنصر داده‌ای به‌یک شیء‌یا به‌یکی از صفتهای یک شیء‌نگاشت می‌شود. این نوع نگاشتها در نمودار **شیء‌ـ رابطه**[20] (ORD)، همانطور که در شکل شمارهٔ ۳ نشان داده شده است، نمایش داده می‌شوند. این مدل نشان دهندهٔ اشیاء، صفتهای اشیاء، زیر ـ شیء‌ها[21]، رفتارها (که به‌سرویسها یا متدها[22] نیز موسوم هستند) و روابط فیمابین اشیاء، می‌باشد. اشیاء نشان داده شده در نمودار ORD می‌توانند به**موضوعاتی**[23] دسته‌بندی شوند. موضوعات گروههایی از اشیاء هستند که بر اثر کار دسته جمعی، سرویسهای مرتبطی را درون سیستم فراهم می‌کنند. این نوع گروه‌بندی سطح بالاتر، سبب تجزیهٔ بیشتر قلمرو مسئله به‌واحدهای قابل دستگیری شده و نوعی سلسله مراتب را ایجاد می‌نماید. موضوعات توسط خطوط خط‌چین حول اشیاء نشان داده می‌شوند (شکل شمارهٔ ۳).

همانطور که در شکل شمارهٔ ۴ نشان داده شده است، توضیح متن‌وار هر

18) Entity Relationship Diagram 19) Information Model
20) Object Relationship Diagram 21) SubObjects
22) Services or Methods 23) Subjects

جدول شماره ۱: ارجاع متقابل شیئ

شیئ	سرویسهای ارائه شده	سرویسهای مورد نیاز	شیئ فراهم کننده سرویس
مریخ	Move Create Report Position Report State Display Detect Update	Display Detect	پدیده مریخ تشخیص دهنده سیمبار
پدیده مریخ	Create Detect		
تشخیص دهنده سیمبار	Create Display	Report Position	
ساعت	Change Sim Step Change Refresh Step Start Clock Stop Clock	Report Clock State Ok to Tick	ساعت ساعت
هسته گرافیک	Create Setup Exit Start Pause Resume	Update Clock Update State Execute Sim Cycle Display	ساعت شمارنده پیمانه شمارنده پیمانه پدیده کارائی شبیه سازی
ساعت	Create Update Clock OK to Tick Report Clock State	Display	پدیده کارائی شبیه سازی
پدیده کارائی شبیه سازی	Display		
شمارنده پیمانه	Create Execute Sim Cycle Update State Report	Move Create Report State Display Delete Update	[illegible]

جدول۱: ارجاع متقابل شیء(OCR).

جدول شماره ۲: رقعهٔ رفتار سیستم

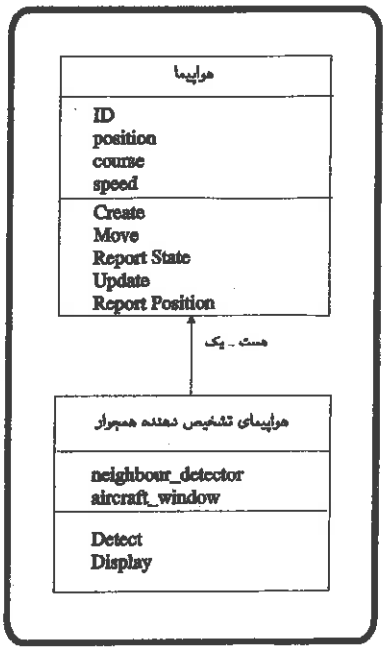
مأموریت سیستم

فعالیتها

واکنش ها

فعالیتهای جانبی

جدول۲: رقعهٔ رفتار سیستم (SBS).



شکل۵: نمودار وراثت (ID). نمودار ID رابطهٔ کلّیت بخشی ـ تخصّصی نمودن (وراثت) بین رده‌های اشیاء را به‌نحوی نشان می‌دهد که بالا ـ رده روایتی کلّیتر از زیر ـ ردهٔ خاصّتر است.

شیء، که **توضیح شیء**[24] (OD) نامیده می‌شود، ویژگیهای شیءرا در نمودار ORD بیان می‌کند. نمودارهای OD از مستندّاتی هستند که در طول چرخهٔ حیات تولید بمرور بهنگام درآورده شده و بیشتر بسط داده می‌شوند. سند دیگری که در طول چرخهٔ حیات تولید بهنگام شده و بسط داده می‌شود، **جدول ارجاع متقابل اشیاء**[25] (OCR)، به‌شرح جدول شمارهٔ ۱، می‌باشد. جدول OCR یک فرم جدولی از کلّیهٔ اشیاء، سرویسهای ارائه شده توسط اشیاء و سرویسهای بکارگرفته شده از سایر اشیاء می‌باشد. به‌دلیل این‌که جدول OCR نمایشی خلاصه از اشیاء سیستم می‌باشد، یکی از ارزشمندترین مستندّات مورد استفاده به‌هنگام نگهداری سیستم تلقّی می‌شود. جدول OCR قابل استحصال از نمودارهای OD بوده و می‌تواند در شرایط ایده‌آل به‌صورت خودکار ایجاد شود.

نهایتاً، پس از اینکه اشیاء، صفتها و سرویسهای اشیاء مشخّص شدند، روابط وراثتی بین رده‌های اشیاء می‌توانند مشخّص شوند. وراثت نوعی رابطهٔ **کلّیت بخشی ـ تخصّصی نمودن**[26] است، به‌طوری‌که بالا-رده[27] روایتی تعمیم یافته از زیر-ردهٔ[28] تخصّصی‌تر است. اینگونه روابط، همانطورکه در شکل شمارهٔ ۵ نشان داده شده‌اند، در یک نمودار وراثت نمایش داده می‌شوند (توضیح مفصّلتر تحلیل و مدل‌سازی اطّلاعات را می‌توان در منابعی همچون [۱۰،۱۱،۱] یافت).

24) Object Description 25) Object Cross-Reference
26) Generalization-Specialization 27) SuperClass
28) SubClass

همچون تحلیل اطّلاعات، تحلیل وقایع سعی می‌کند عناصر داده‌ای موجود در قلمرو مسئله را تشخیص و سپس به‌اشیاء نگاشت نماید. تحلیلگر به‌هنگام تحلیل وقایع، سیستم را به‌عنوان یک ماشین **محرّکه‌ای ـ واکنشی**[29] (SRM) در نظر می‌گیرد که در اثر یک سری تحریکات خارجی از خود واکنش نشان می‌دهد. تحلیل وقایع می‌تواند به‌همراه تحلیل اطّلاعات، یا اینکه جداگانه و به‌عنوان تمرینی برای تصدیق فاز تحلیل، انجام گیرد. در متدولوژی پیشنهادی، تحلیل وقایع عمدتاً برای تصدیق فاز تحلیل به‌کارگرفته می‌شود.

در صورت موجود بودن یک اظهار روشن از مقصود اصلی سیستم، می‌توان کلیهٔ فعّالیتهایی را که سیستم در پاسخ به‌محرّکهای خارجی انجام خواهد داد، مشخّص نمود. از میان لیست این فعّالیتها، تحلیلگر باید **فعّالیتهای بنیادین** سیستم را که بایستی در پشتیبانی مستقیم از مقصود اظهار شدهٔ سیستم انجام بگیرد، مشخّص نماید. این فعّالیتها بدین جهت بنیادین تلقی می‌شوند که بدون وجود آنها دستیابی به‌مقصود اصلی سیستم امکانپذیر نمی‌باشد. سایر فعّالیتهایی که در پشتیبانی از فعّالیتهای بنیادین انجام می‌گیرند، **فعّالیتهای جنبی** هستند. فعّالیتهای تحریکی و واکنشی در **رقعه‌های رفتار سیستم**[30] (SBS) درج می‌شوند (جدول شمارهٔ ۲).

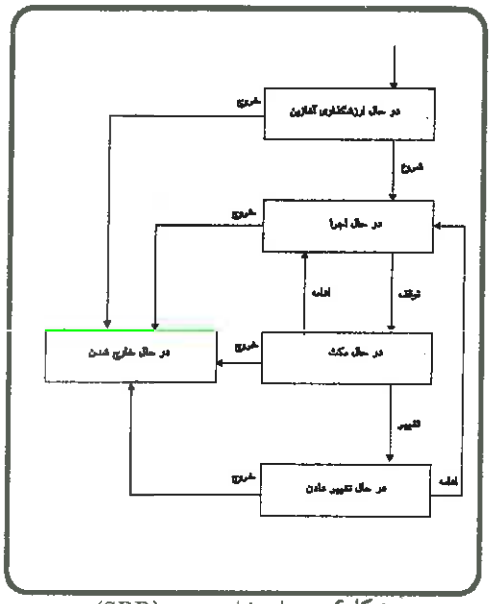
بسیار مهم است که رفتار کلّی سیستم از یک دیدگاه خارجی نسبت به‌سیستم مشخّص شود. این رفتار توسط **یک نمودار رفتار سیستم**[31] (SBD) نشان داده می‌شود (شکل شمارهٔ ۶). نمودار SBD یک **نمودار گذار حالتها**[32] (STD) است که حالتهایی از سیستم را که از خارج سیستم قابل رؤیت هستند، به‌همراه گذارهای ممکنهٔ بین این حالتها نشان می‌دهد. محرّکهایی که سبب گذار بین حالتها می‌شوند، می‌توانند همان محرّکهایی باشند که در رقعه‌های SBS قید شده‌اند.

قدم بعدی، تشخیص عناصر داده‌ای است که همراه هر تحریک بوده و هر واکنش را تشکیل می‌دهند. اغلب این عناصر داده‌ای می‌بایستی قاعدتاً به‌هنگام تحلیل اطّلاعات مشخّص شده باشند. لذا این عناصر می‌توانند اعتبار مجموعهٔ موجودیّتهای تشخیص داده شده در تحلیل اطّلاعات را تأیید نمایند. اگر موجودیّتهای جدیدی یافت شوند، بایستی بلافاصله به‌نمودار ERD افزوده شوند.

همانطورکه در تحلیل اطّلاعات عمل می‌شود، مرحلهٔ بعدی پس از مشخّص نمودن عناصر داده‌ای، نگاشت این عناصر به‌اشیاء یا صفتهای اشیاء است. این فرارونـد شبیه فرارونـد مدل کردن اطّلاعات بوده و بایستی مکمّل نتایج تحلیل اطّلاعات باشد. بنابراین نمودار ORD بایستی به‌نحوی تغییر یابد تا اشیائی را که جدیداً مشخّص شده‌اند دربر بگیرد.

به‌هنگام تشخیص اشیاء از یک دیدگاه محرّکه‌ای ـ واکنشی، تحلیلگر مشخّص می‌کند که چه اشیائی چه فعّالیتهایی از سیستم را انجام می‌دهند. این عمل

29) Stimulus Response Machine 30) System Behaviour Scripts 31) System Behaviour Diagram 32) State Transition Diagram



شکل۶: نمودار رفتار سیستم (SBD).

منجر به‌متعلّق ساختن رفتارهای سیستم به‌اشیاء خاصّی می‌شود. رفتار هر شیء‌در توضیح شیء(OD) درج می‌شود (توضیح مفصّلتر تحلیل وقایع را می‌توان در مرجع [۶] یافت).

گذار به‌طرّاحی سیستم

محصول نهایی قلمرو مسئله، **نمودار خادم‌ـمخدوم** (CSD) است (شکل شمارهٔ ۷). نمودار CSD دید وسیعتری از فعل و انفعالات فیمابین اشیاء را، در مقایسه با فعل و انفعالات نشان داده شده در نمودار ORD، نشان می‌دهد. لذا، نمودار CSD گذار بین تحلیل و طرّاحی سیستم را مقدور می‌سازد. پیکانهای جهتدار از مخدوم به خادم اشاره می‌کنند، روابط چندگانه[۳۳] فیمابین اشیاء، گروههای موضوعات و پیغامهای قابل ردّوبدل شدن بین اشیاء، نشان داده می‌شوند.

لازم به‌تذکّر است که متدولوژی پیشنهادی تولید، فراروندی چرخه‌ای است. به‌عبارت دیگر، بازگشت به‌فازهای قبلی برای تغییر سازماندهی و پالایش ایده‌ها، دور از انتظار نیست. این نکته لازم به‌ذکر است که تحلیل بایستی فراروندی کامل و دقیق باشد. با نگریستن به‌سیستم از دو بعد اطّلاعات و وقایع، تحلیل‌گر اطمینان بیشتری نسبت به‌مناسب بودن تجزیهٔ قلمرو مسئله دارد. این ویژگی بسیار مهّم است، چون تحلیل چارچوبی را برای طرّاحی و پیاده‌سازی فراهم می‌کند. پایان فاز تحلیل نقطه‌ای استراتژیک است که بایستی در این نقطه یک جلسه برای بررسی مجدّد با کاربر ترتیب داده شود.

طرّاحی سیستم

فاز طرّاحی سیستم بر تعریف نمودن نحوهٔ کارکردن سیستم توجّه دارد. در این فاز، منابع محاسباتی و حافظه‌ای لایتناهی مفروض بوده و لذا جزئیات پیاده‌سازی مطرح نیستند. دو فعّالیت عمده در این فاز رخ می‌دهند: تشخیص اشیاء جدیدی که خاصّ طرّاحی سیستم هستند، و پالایش اشیائی که در زمان تحلیل مشخّص شده‌اند. یک لیست پیوندی می‌تواند نمونه‌ای از یک شیء‌جدید باشد که خاصّ طرّاحی سیستم بوده و در زمان تحلیل سیستم مشخّص نشده باشد. در اشکال نمونه‌ای که در این مقاله به‌کارگرفته شده‌اند، هواپیما به‌عنوان یک شیء‌بنیادین در هنگام تحلیل سیستم مشخّص شده است. یک لیست پیوندی می‌تواند به‌عنوان یک شیء‌به‌کار رود که قادر به‌دستکاری نمودن مجموعه‌ای از نمونه‌های هواپیما است. در این صورت، لیست پیوندی یک شیء‌بنیادین نمی‌باشد، بلکه فقط جهت تعریف این‌که سیستم چگونه کار می‌کند به‌کارگرفته شده و لذا خاصّ طرّاحی سیستم است.

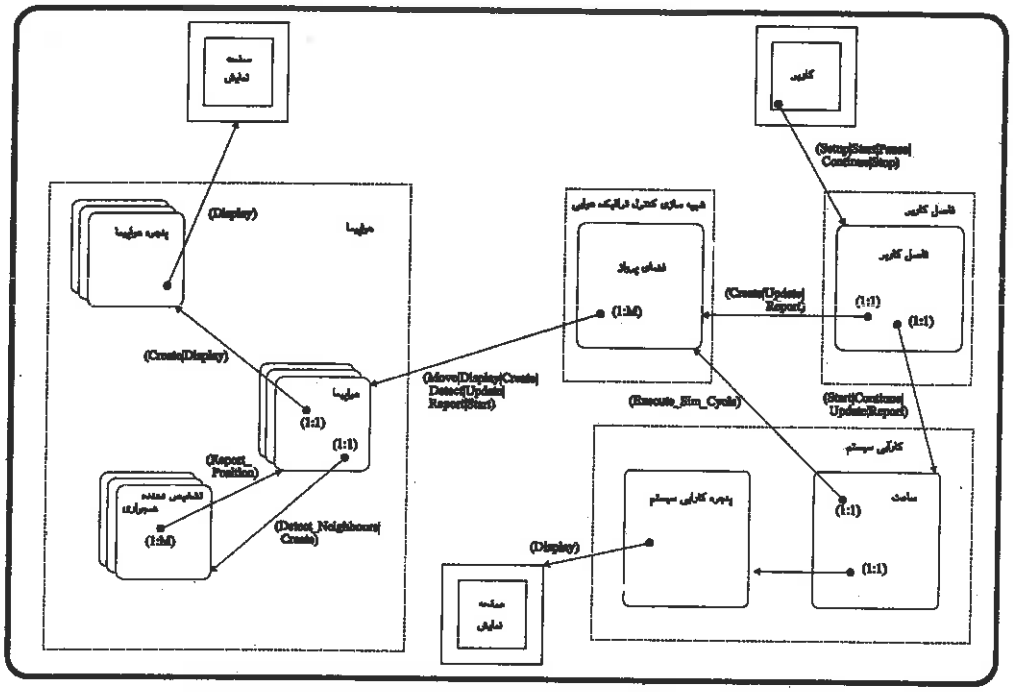
در تولید شیء‌گرا، مشخّص نمودن نحوهٔ عمل سیستم، کم و بیش، توسط فعل و انفعالات فیمابین اشیاء انجام می‌گیرد. تعریف نمودن کامل و واضح این فعل و انفعالات یکی از اهداف عمدهٔ فاز طرّاحی سیستم بوده و عمدتاً بدینطریق اشیاء تعریف شده و در زمان تحلیل سیستم در هنگام فاز طرّاحی پالایش می‌یابند.

نمودار فاصل شیء[۳۴] (OID) فعل و انفعالات اشیاء را از دیدگاه یک شیء‌تنها نشان می‌دهد. نمودار OID، که در شکل شمارهٔ ۸ نشان داده شده است، معمولاً از نمودار CSD، که نشان دهندهٔ فعل و انفعالات اشیاء برای تمام سیستم است، استحصال می‌شود. نمودار CSD روابط بین اشیاء را در سطح سیستم نشان می‌دهد ـ نمودار OID سطح جزئیات را افزایش داده و به‌روابط یک شیء‌تنها، به‌عوض کلّ سیستم، توجّه می‌کند. اشیائی که سرویسهای ارائه شده توسط شیء‌موردنظر را استفاده می‌کنند، به‌همراه اشیائی که شیء‌موردنظر از سرویسهای آنها استفاده می‌کند، نشان داده می‌شوند. در فاز تحلیل، رفتار سیستم توسط **نمودار رفتار** (SBD) و **رقعهٔ رفتار سیستم** (OBS) نمایانده می‌شود. نمایشهای مشابهی نیز برای رفتار هر شیء‌تنها در **نمودار رفتار شیء** (OBD) و **رقعهٔ رفتار شیء** (OBS) در فاز طرّاحی ایجاد می‌شوند. به‌دلیل اینکه هر شیء‌دارای حالتهای مورد توجّه نمی‌باشد، لذا یک OBD برای هر شیء‌لازم نیست. یک OBS نمونه در جدول شمارهٔ ۳ نشان داده شده است. یک OBS قالبی شبیه یک SBD (شکل شمارهٔ ۶) دارد، لذا از ذکر یک نمونهٔ OBD در اینجا خودداری شده است.

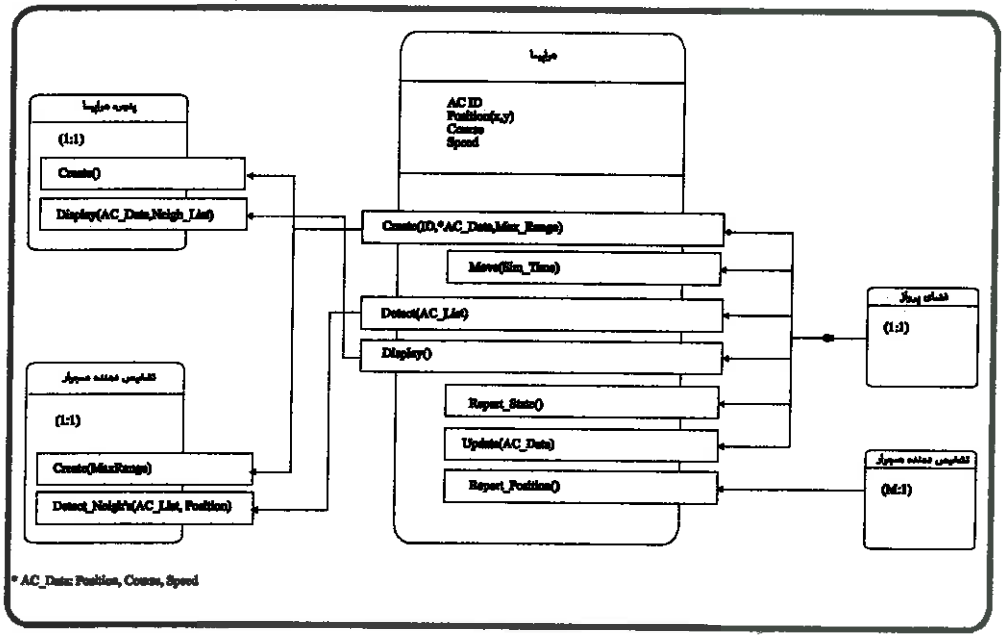
مستندّات OD و OCR که در فاز تحلیل ایجاد شده‌اند، در فاز طرّاحی سیستم پالایش شده و حاوی جزئیات بیشتری دربارهٔ رفتار شیء‌موردنظر می‌شوند. نمودار OD به‌نحوی بسط داده می‌شود که حاوی یک **تعریف سرویس**[۳۵] برای هر سرویسی که شیء‌موردنظر ارائه می‌کند، باشد. تعریف سرویس شامل اسامی و گونه‌های پارامترهای ورودی و خروجی می‌باشد. جدول OCR نمایشی جدول‌گونه از اطّلاعات راجع به‌اشیاء است ـ البته حالا در سطح طرّاحی سیستم، شامل ستونهایی برای پارامترهای سرویسهای

33) n-ary

34) Object Interface Diagram 35) Service Definition



شکل۷: نمودار خادم-مخدوم (CSD).



شکل۸: نمودار فاصل شیء(OID).

ارائه شده توسط شیءنیز می‌باشد. به‌علاوه، چنین انتظار می‌رود که OCR به‌صورت خودکار ایجاد شود.

طرّاحی نرم‌افزار

فاز طرّاحی نرم‌افزار، محصولات فاز طرّاحی سیستم را افزایش می‌دهد تا بتواند جزئیات خاص پیاده‌سازی طرّاحی سیستم را لحاظ کند. اگرچه جزئیات زبان پیاده‌سازی درنظرگرفته نمی‌شوند، ولی جزئیات جایگاه سخت‌افزاری و محیط نرم‌افزاری مقصد مورد توجّه قرار می‌گیرند.

مستنّدات تهیه شده در فاز طرّاحی سیستم، همچون OD، OBD، OID، OBS و OCR می‌توانند در این فاز تولید بسط داده شوند. البته، اینکه کدام سند احتیاج به‌بسط داده شدن در فاز طرّاحی نرم‌افزار را دارد، به‌ویژگیهای خاصّ سیستم و جایگاه مقصد سیستم بستگی دارد. به‌طور کلی، اگر

جدول شماره ۳ : رقعه رفتار شیء	
پیغام	رقعه رفتار
Create (ID, Position, Course,Speed, Max-Range)	خلق یک شیء هواپیما و ارزشگذاری آغازین مشخصه‌های آن : ارزشگذاری صفت Position(x,y) ارزشگذاری صفت Course ارزشگذاری صفت Speed خلق دو شیء : خلق پنجره هواپیما خلق تشخیص دهنده همجوار
Move(Sim_Time)	محاسبه موقعیت جدید هواپیما بر اساس [illegible]
Update(Position, Course,Speed)	باز نشاندن بعضی از ارزشهای مشخصه‌های شیء هواپیما - تغییر ارزش جدید صفت Position(x,y) - تغییر ارزش جدید صفت Course - تغییر ارزش جدید صفت Speed
Report_State()	بازگشت دادن ارزشهای مشخصه‌های شیء هواپیما
Display()	[illegible] اطلاعات در مورد هواپیماهای همجوار از تشخیص دهنده همجوار شیء هواپیما ارسال پیغام (Report-State) پاس کردن پیغام Display به شیء - شیء پنجره هواپیما به این شکل : Display(Position,Course,Speed,Neigh_List)
Detect(AC_List)	پاس کردن پیغام Detect به شیء شیء تشخیص دهنده همجوار از طریق فراخواندن تشخیص دهنده همجوار که لیستی از کلیه هواپیماهای در فضای پرواز به آن پاس می‌شود. این شیء به ترتیب موقعیت هر هواپیما را جویا شده و با MaxRange مقایسه می‌کند تا تشخیص دهد که آیا این هواپیما همجوار هواپیماهای دیگر است یا خیر.
Report_Position()	بازگشت دادن ارزش صفت Position(x,y) شیء هواپیما

جدول۳: رقعهٔ رفتار شیء(OBS)

جزئیات پیاده‌سازی به‌نمودار خاصی مرتبط می‌شوند، آن نمودار بسط داد می‌شود ـ در غیر اینصورت، نمودار ایجاد شده در زمان طراحی سیستم بدون تغییر استفاده می‌شود. برای بسیاری از اشیاء، مستندات OBD، OID و OBS نیازی به‌پالایش شدن در فاز طراحی نرم‌افزار ندارند. انتظار می‌رود که نمودار OD برای هر شیءبه‌نحوی بسط داده شود تا شامل اطلاعاتی از این قبیل گردد: استفاده از فراخوانیهای سیستم‌عامل، و اینکه چه روالهایی (یا متدهایی) خصوصی و کدام عمومی هستند. یک متد خصوصی فقط برای استفادهٔ خود شیءاختصاص می‌یابد، در حالی که یک متد عمومی می‌تواند توسط عموم استفاده شود. مشخص کردن متدهای خصوصی و عمومی، حتی زمانی که پیاده‌سازی چنین تمایزی را پشتیبانی نکند، برای طراحی نرم‌افزار بسیار مفید فایده است. همچنین، واحدهای اندازه‌گیری صفتهای هر شیءبایستی در نمودار پالایش شدهٔ OD برای آن شیءشامل شده و از جهت سازگاری در میان اشیایی که پیغام رد و بدل می‌کنند بازبین شوند.

برای نمایش پردازشهای مورد نیاز در داخل هر شیء، به‌یک **نمودار پردازش شیء**[۳۶] (OPD) نیاز است. این نمودار علاوه بر نمایش متدهای خصوصی و عمومی، نشان می‌دهد که چه متدهایی ارزشهای صفتهای شیءرا خوانده و یا ارزشگذاری می‌کنند. پارامترهای ورودی و خروجی هر متد نیز نشان داده می‌شوند. نمونه‌ای از یک نمودار OPD در شکل شمارهٔ ۹ نشان داده شده است.

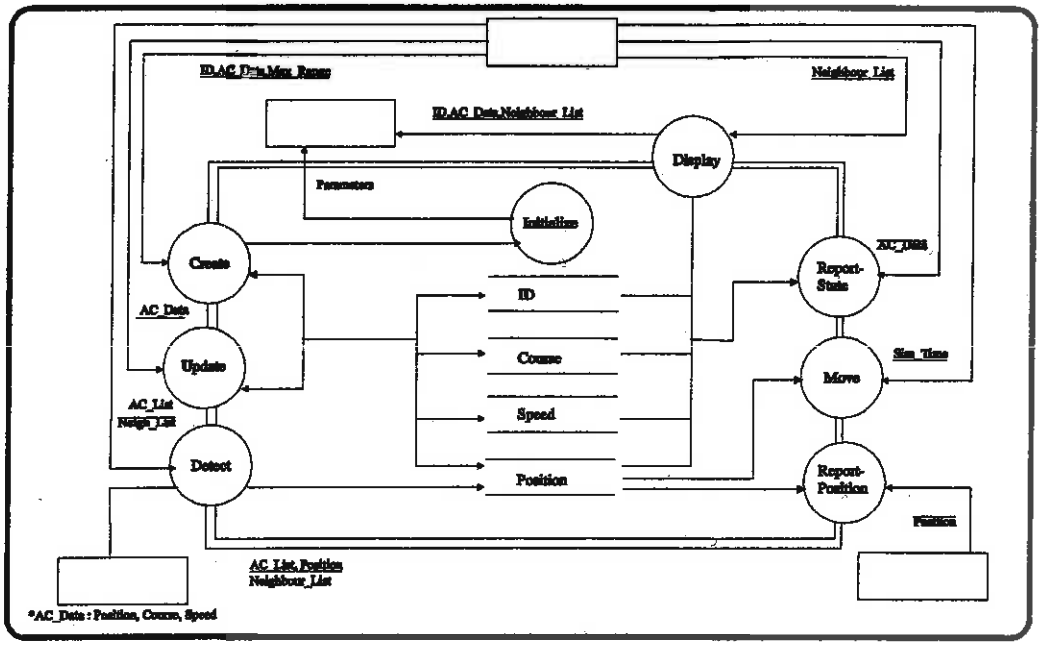
برای توضیح دادن اشیائی که الگوریتمهای پیچیده‌ای دارند، یا اشیائی که پیاده‌سازی آنها ابعاد پیچیده‌ای دارند، می‌توان از کد صوری استفاده نمود. تصمیم‌گیری در مورد استفاده یا عدم استفاده ازکد صوری به‌قضاوت طراحی بستگی دارد.

نتیجه‌گیری

متدولوژی پیشنهادی، روشی ساده برای تولید شیءگرای سیستمها ارائه می‌کند. این متدولوژی چهار مزیت عمده دارد. اولاً، تلاش در جهت تولید سیستم در فاز تحلیل متمرکز بوده و در فازهای بعد کاهش می‌یابد. به‌دلیل این ویژگی، انتظار می‌رود که هزینهٔ تشخیص و رفع اشتباهات غیر قابل اجتناب در سیستم کاهش یافته، و بدین ترتیب تلاش لازم برای کل سیستم کاهش یابد. ثانیاً، سیستم از منظرهای گوناگون (**کلّ سیستم، اشیاء تک و اشیاء فعل و انفعالی**) دیده شده و نمایش داده می‌شود. این کار منجر به‌ایجاد مستنداتی می‌شود که توسط افراد مختلف (**کاربر، تحلیل‌گران، طراحان، برنامه‌سازان و آزمونگران**) به‌سهولت قابل فهم هستند. ثالثاً، تلاشها و مستندات مراحل بعدی تولید عمدتاً بر تلاشها و مستندات قبلی استوارند. این ویژگی، قابلیت ردیابی را در سرتاسر چرخهٔ کامل حیات تولید مقدور می‌سازد. نهایتاً، چون این متدولوژی بسیاری از امور، بالاخص امور مراحل نهایی تولید را به‌قضاوت تحلیل‌گران و طراحان وامی‌گذارد، انعطاف لازم برای به‌کارگرفته شدن در سیستمها و قلمروهای مختلف از مسائل را داراست.

منظرهای چندگانه از سیستم در دست تولید می‌تواند فهم عمیقتری از محتوی و رفتار سیستم را فراهم کند. این منظرها فقط در صورت مناسبت بایستی استفاده شوند. البته می‌توان با استفاده از منظرهای چندگانهٔ سیستم که در فازهای ابتدایی فراروند تولید ایجاد شده‌اند، به‌طراحی و پیاده‌سازی واضحتر و سازگارتری از سیستم دست یافت. ضرورت وجود یک ابزار CASE نسبتاً پیچیده جهت تسهیل امر توضیح دادن و وارسی سیستم از جنبه‌های مختلف، و همچنین جهت فراهم آوردن چارچوبی برای برقرار نمودن سازگاری بین منظرهای چندگانهٔ سیستم، توسط متدولوژی پیشنهادی تأیید شده است.

36) Object Process Diagram



شکل۹: نمودار پردازشی شیء(OPD)

متدولوژی پیشنهادی همچنین بر نیاز به‌تصدیق فازهای مختلف فرارونـد تولید تأکید نموده و انعطافپذیری لازم را برای فراهم آوردن یک فرارونـد چرخه‌ای و تکراری داراست.

●

قدردانی

بدینوسیله از زحمات سرکار خانم دشتیانه و جناب آقای محتشمی که در تایپ و رسم اشکال بر روی کامپیوتر کمک شایانی کرده‌اند قدردانی می‌شود.

مراجع

[1] Schlaer, S., and Mellor, S. J., *Object Lifecycles: Modeling the World in States*, (Yourdon Press, Englewood Cliffs, New Jersey, 1992).

[2] Ward, P.T., *The CASE Real-Time Curriculum*, (Software Development Concepts, New York, 1989).

[3] Coad, P., and Yourdon, E., *Object-Oriented Analysis*, (Prentice-Hall, Englewood Cliffs, New Jersey, 1990).

[4] Booch, G., *Object-Oriented Design with Applications*, (Benjamin Cummings, Redwood City, California, 1990).

[5] Bailin, S., *An Object-Oriented Requirements Specification Method*, (Commun. ACM. 1989. **32**, (5), pp. 608-623).

[6] McMenamin, S. M., and Palmer, J. F., *Essential Systems Analysis*, (Yourdon Press, New York, 1984).

[7] Rumbaugh J., et als. *Object-Oriented Modeling and Design*, (Prentice Hall, Englewood Cliffs, New Jersey, 1991).

[8] Myers, G. J., *The Art of Software Testing*, (John wiley & Sons, New York, 1979).

[9] Cox, B. J., *Object-Oriented Programming: An Evolutionary Approach*, (Addison Wesley, Reading, Massachusetts, 1986).

[10] Gause, D. C., and Weinberg, G. M., *Exploring Requirements: Quality before Design*, (Dorset House, New York, 1986).

[11] Schlaer, S., and Mellor, S. J., *Object-Oriented Systems Analysis: Modeling the World in Data*, (Yourdon Press, Englewood Cliffs, New Jersey, 1988).

آیا ایده‌های قدیمی هنوز اعتبار دارند؟

رامین عرفانیان*

در سال ۱۹۹۲، در لندن میزگردی تحت عنوان «ایده‌های قدیمی هنوز معتبر هستند» با شرکت آقایان Tim Wolf، Michael Simpton، Richard Keith و رامین عرفانیان برگزار شد. در این میزگرد، Tim Wolf و رامین عرفانیان به‌عنوان مخالفین ایده‌های قدیمی در علم کامپیوتر صحبت می‌کردند. ذیلاً گزارشی از این میزگرد تقدیم می‌گردد.

در این گزارش سعی کرده‌ام بحثی دقیق و در عین حال مختصر و بر اساس ادعاهای مؤلفین ارائه دهم. قسمتهای انتخاب شده از مطالب را که یا با آنها موافق هستم و یا مخالف، به‌صورت معمولی نوشته‌ام و توضیحات خودم با حروف خوابیده مشخص شده است. هر مطلب به‌صورت جداگانه بازبینی و توضیح داده خواهد شد و در پایان نتیجه‌ای گرفته می‌شود.

توضیحاتی بر «لباسهای کهنه امپراتور[1]» اثر ریچارد هور[2]

هور چهار اصل اساسی را در طراحی زبان خود بیان می‌کند.

۱ـ امنیت. هر برنامه‌ای که از نظر نحوی اشتباه است باید توسط کامپایلر رد شود و هر برنامه‌ای که از نظر نحوی صحیح است باید نتیجه یا پیام خطایی را که قابل پیش‌بینی و قابل درک برحسب خود زبان برنامهٔ مبدأ است ارائه کند.

این یک قاعدهٔ خوش‌تعریف است که در کامپایلرهای زیادی در طی سالیان طولانی مورد غفلت واقع شده است. در حقیقت من دوست دارم قانونی را ببینم که می‌گوید از نظر منطقی غیرممکن است هر برنامهٔ به زبان مبدأ باعث شود که کامپیوتر در زمان اجرا بی‌هدف و خودسرانه عمل کند. اگر چه این یک فکر آرمانی می‌باشد، اما قطعاً حامل این پیام است که برنامه‌سازی خوش‌ترکیب معادل برنامه‌ای نیست که از نظر نحوی صحیح باشد.

۲ـ ایجاز و اختصار برنامه مقصد که توسط کامپایلر تولید می‌شود و فشردگی داده‌هایی که در زمان اجرا روی آنها کار می‌شود.

قاعده‌هایی که سی سال پیش وضع شده‌اند، با توجه به رشد بسیار اندیشه‌های جدید، نوآوریها و منابع تقریباً نامحدود، در حال حاضر خیلی معتبر نیستند. علم کامپیوتر یک جهش بسیار بزرگ تاریخی داشته است و این درست شروع عصر جدید است. برای مثال ماشینی بر مبنای پردازنده ۴۸۶ اینتل از تمام کامپیوترهای دههٔ ۱۹۵۰ که کنار یکدیگر قرار گیرند، قدرت محاسباتی بیشتری دارد. بدون تردید، واقعیت غیرقابل اجتناب امروز، جزئیات ناچیز فردا خواهند شد، ولی به هرحال هر قدر پیشرفتهای تکنولوژیک سریع هم باشد، هنوز قواعد جامعه‌شناسی باقی هستند.

۳ـ قواعد ورود و خروج رویه‌ها و توابع

در اینجا باید به نکته‌ای اشاره شود، یعنی اصل فوق منعکس‌کننده روشی است که در زمان ابداع، منطقی و اساسی بود، اما این اندیشه طراوت و تازگی خودش را از دست داده است و ما به‌تدریج هدف خود را ازکارایی و سرعت به مدیریت پیچیدگی تغییر می‌دهیم. این امر مستلزم این است که تجزیهٔ تابعی به اَشکال دیگری از تجزیه تغییر کند. همچنین باید گفت که اکثر برنامه‌های بیدرنگ از بازگشت به‌عنوان ابزار اصلی تکرار استفاده نمی‌کنند بنابراین، تأکید عمده باید بر روی تولیدکننده‌های کدهای فشرده‌تر باشد.

۴ـ کامپایلر باید فقط از تک‌گذر استفاده کند.

من مطمئن نیستم که عبارت «تک‌گذر» در مقابل چندگذر استفاده شده باشد، اگر چنین باشد با یک مسألهٔ اساسی مواجه هستیم. اصل دوم می‌گوید که مقصد باید تا حد ممکن کوتاه باشد در حالیکه استفاده از یک گذر واحد، مترجم را از استفاده از پرشهای کوتاه و بسیاری بهینه‌سازیهای دیگر، که در صورت استفاده از چندگذر انجام می‌شدند، محروم می‌کند.

روش کوتاه کردن برنامه استفاده از رویه‌هاست نه حذف اطلاعات ضروری توصیفی.

این امر هر قدر هم که درست باشد، اما باز هم اجازه بدهید از یاد نبریم که محیط و پارامترهایی که در اختراع این

*) اصل این مقاله به انگلیسی نوشته شده و توسط آقای سعید زین‌العابدین به فارسی ترجمه شده است.

1) The Emperor's Old Cloths 2) Richard Hoare

قاعده بکار رفته‌اند کاملاً با آنچه که اکنون وجود دارد متفاوت هستند. امروزه، برنامه‌های کوتاه آنهایی نیستند که طول کمتری داشته باشند بلکه برنامه‌هایی هستند که زمان کمتری برای تولید می‌گیرند، بنابراین، قابلیت استفاده مجدد و ایده‌های جدید بسیار دیگری مطرح می‌شوند.

یک روش این است که مسئله را آنقدر ساده کنیم که به‌طور واضح و آشکار هیچ کمبودی وجود نداشته باشد.

اینشتن می‌گوید «مسئله را تا حد ممکن ساده کنید، اما نه دیگر ساده‌تر از آن.» اگر چه درک ابزار ساده آسانتر است اما چیزهایی وجود دارند که طبیعتاً ساده نبوده و برای حل آنها به ابزارهای پیچیده‌تری نیاز داریم. به نظر من، همیشه راههایی برای برقرار کردن توازن وجود دارد. تنها مسأله این است که باید سخت فکر کنید.

توضیحاتی بر «برنامه‌ساز فروتن[3]» اثر ادسخر دایکسترا[4]

قبل از اینکه دهۀ ۱۹۷۰ تمام شود ما قادر خواهیم بود انواع سیستمهایی را طراحی و اجرا کنیم که با توانایی برنامه‌سازی کنونی نمی‌توانیم. هزینه ایجاد این سیستمها تنها چند درصد نفرسالهایی است که امروز صرف آنها می‌کنیم. علاوه بر آن، این سیستمها واقعاً از خطاها مصون خواهند بود. این دو پیشرفت به موازات هم به جلو می‌روند.

این سیستمها به تعداد زیاد به‌وجود می‌آیند و رشد می‌کنند. اسمال‌تاک یکی از مثالهای خوب است. برنامه‌ساز اسمال‌تاک مجبور نیست از صفر شروع کند، اما باید از مجموعه‌ای از رده‌های خوش تعریف که قابلیت عملیاتی دلخواه او را کمابیش برآورده می‌کند استفاده کند. در حالتی که یک قابلیت عملیاتی خاص را نداشته باشد، برنامه‌ساز می‌تواند قابلیت جدید را روی قابلیتی که قبلاً آزمایش و ثابت شده است، ایجاد کند. به دو روش می‌توان به سیستمهای عاری از خطا دست یافت:

- *با داشتن یک سیستم قوی که تمام مؤلفه‌های آن خوش‌تعریف بوده و آزمایش شده‌اند، بنابراین برنامه‌سازی عبارت است از فعالیت در کنار هم قرار دادن این مؤلفه‌ها، برای مثال زبان ایدا.*
- *ابداع روش تفکری که بدون توجه به صلاحیت برنامه‌ساز، خطاها را می‌یابد. واضح است که هنوز چنین سیستمی وجود ندارد اما یک تطابق نوع قوی[5] و روشهای مهندسی نرم‌افزار معتبر و صحیح بسیار به این هدف در حال نزدیک شدن هستند.*

3) Humble Programmer 4) Edsger W. Dijkstra 5) Strong type-checking

دایکسترا می‌گوید آنهایی که نرم‌افزار واقعاً قابل اعتماد می‌خواهند خواهند فهمید که برای شروع باید ابزاری را برای اجتناب از اکثریت خطاها بیابند.

۱ـ برنامه‌ساز فقط نیاز دارد که مسائل قابل مدیریت عملی را در نظر بگیرد.

۲ـ به محض اینکه ما تصمیم گرفتیم که خود را به زیرمجموعۀ برنامه‌هایی که به‌طور عملی قابل مدیریت هستند، محدود کنیم، یکبار و برای همیشه به کاهش مؤثر فضای راه‌حلی که در نظر گرفته می‌شود، دست یافته‌ایم.

من طرفدار بسیار سرسخت فرضیۀ منابع نامحدود انسان هستم و معتقدم اگر زمان و تجربۀ کافی در دسترس باشد، هر چیزی ممکن است انجام شود. به هر حال مهمترین روش مدیریت پیچیدگی تقسیم آن به قسمتهای کوچکتر و غلبه بر این قسمتها است. اینکه این کار چگونه و یا بوسیلۀ چه کسی انجام شود موضوعی است که می‌توان دربارۀ همۀ جوانب آن بحث کرد.

۳ـ آزمایش برنامه می‌تواند روشی بسیار مؤثر برای نشان دادن وجود خطاها باشد، اما این روش برای نشان دادن اینکه خطایی وجود ندارد اصلاً مناسب نیست. تنها راه مؤثر برای بالا بردن سطح اطمینان برنامه در حدّی چشمگیر، ارائه اثباتی متقاعدکننده از صحت آن است. اما نباید برنامه‌ای نوشت و بعد صحت آن را اثبات کرد، زیرا در این صورت ملزومات ارائۀ اثبات فقط به بار مشکلات برنامه‌ساز خواهد افزود.

یک نکتۀ عالی. چیزی که من دوست دارم ببینم، تحقیقات برای یافتن روشهای جدید اثبات غیر از اثبات ریاضی است.

۴ـ تنها ابزار فکری که بوسیلۀ آن یک استدلال خیلی محدود ممکن است تعداد زیادی حالت را شامل شود، «انتزاع» نامیده می‌شود. در نتیجه، استفادۀ کارا از این توانایی باید به‌صورت یکی از مهمترین فعالیتهای یک برنامه‌ساز شایسته در نظر گرفته شود.

۵ـ ابزاری که سعی می‌کنیم از آنها استفاده کنیم و نشانگذاری زبان که برای بیان و یا ضبط افکارمان استفاده می‌کنیم اصلی‌ترین عواملی هستند که تعیین می‌کنند اصلاً دربارۀ چه چیزی می‌توانیم فکر کنیم یا چه چیزی را می‌توانیم بیان کنیم! تحلیل تأثیری که زبانهای برنامه‌سازی بر عادات فکری استفاده‌کنندگان خود دارند، و اذعان به اینکه تاکنون مغز باارزش‌ترین منبع ما است. برنامه‌ساز شایسته از اندازۀ بسیار محدود جمجمه‌اش کاملاً آگاه است.

این نشانه‌ای از آگاهی ناقص یا فهم اشتباه نقش زبانهای برنامه‌سازی در مهندسی نرم‌افزار است. همانطور که بارها

اشاره کرده‌ام، زبانها ابزاری هستند که باید به ما برای رسیدن به اهدافمان کمک کنند. مثل استفاده از چاقو، اگر شما در حال دفاع از خانواده‌تان در مقابل یک جانور باشید، در این صورت کشتن با چاقو نه تنها منطقیترین راه است بلکه چاقو بهتر وسیله برای استفاده است. برعکس، اگر از سنگ برای زدن کسی استفاده کنید نه تنها روش تفکر شما اشتباه بوده است بلکه وسیلهٔ اشتباهی نیز انتخاب کرده‌اید. زبانها نباید به‌خاطر چیزهایی که مسئول آنها نیستند مقصر دانسته شوند. ما، به‌عنوان برنامه‌ساز باید روشهای بهتر تفکر را بکار گیریم و باید ابزار درست را بکار بریم.

به‌عنوان یک نکتهٔ جنبی مایلم یک اخطار به کسانی بدهم که دشواری وظیفهٔ برنامه‌سازی را با کوشش برای بهتر کردن ابزار نامناسب فعلی مربوط می‌دانند، زیرا ممکن است نتیجه بگیرند که وقتی ابزار ما بسیار مناسبتر شد، برنامه‌سازی دیگر به‌عنوان یک مسأله و مشکل مطرح نخواهد بود. ولی برنامه‌سازی همچنان بسیار مشکل باقی خواهد ماند، زیرا وقتی ما خود را از زحمات و مشکلات فعلی رها کردیم، برای درگیر شدن با مسائلی که اکنون خارج از ظرفیت برنامه‌سازی ما هستند، آزاد خواهیم شد.

توضیحاتی بر «چرا برنامه‌سازی تابعی مهم است»[6] اثر ج. هیوز[7]

مشخصات خاص و مزایای برنامه‌سازی تابعی اغلب کمابیش به‌صورت زیر جمع‌بندی می‌شود:

- برنامه‌های تابعی حکم انتسابی ندارند، بنابراین متغیرها، یکبار که مقداری گرفتند، هیچگاه تغییر نمی‌کنند.
- برنامه‌های تابعی هیچ نوع تأثیر جانبی ندارند. تابع ممکن است جز محاسبهٔ نتیجه‌اش، هیچ نوع اثر دیگری نداشته باشد. این ویژگی یک منبع اصلی خطا را از بین می‌برد و نیز ترتیب اجرا را بی‌اهمیت می‌کند. چون هیچ تأثیر جانبی نمی‌تواند مقدار یک عبارت را تغییر دهد، آن عبارت را می‌توان در هر زمانی ارزیابی کرد. این کار برنامه‌ساز را از مسئولیت تنظیم جریان کنترل آزاد می‌کند. از آنجا که عبارت را می‌توان در هر زمانی ارزیابی کرد، می‌توان به‌راحتی، متغیرها را با مقادیرشان جایگزین ساخت و برعکس. این آزادی برنامه‌های تابعی را از نظر ریاضی نسبت به نمونه‌های قراردادی آنها، بیشتر قابل ردیابی می‌کند.

مسائلی وجود دارد که فکر می‌کنم قبل از اینکه جلوتر برویم باید به آنها اشاره شود.

اول، مسئولیت نرم‌افزار این است که آنچه که قرار است انجام دهد را به روشی صحیح و با حداکثر سرعت ممکن انجام دهد.

دوم، علم کامپیوتر مبتلا به دخالت بیمورد ریاضیدانهایی است که دوست دارند برای همه چیز به روشی ریاضی دلیل بیاورند. اگر واقعاً می‌توانند لطفاً در مورد دنیای ما و معجزه‌های عجیب و غریب آن دلیل بیاورند.

سوم، تمایز قطعی بین مهندسی نرم‌افزار تولیدی (سودمند از نظر اقتصادی) و مهندسی نرم‌افزار آموزشی وجود دارد.

چهارم، نویسنده هیچ نظری دربارهٔ الگوهای دیگری مانند شی‌گرایی ندارد و یا نسبت به تجزیهٔ تابعی تعصب دارد.

پنجم، در بسیاری از موارد، ما به‌عنوان برنامه‌ساز به برخی از تأثیرات جانبی مانند چاپ بعضی نتایج و گرفتن ورودی نیاز داریم. این کارها را به دلایل واضحی، نمی‌توان به یک ترتیب دلخواه ارزیابی کرد.

اکنون عموماً پذیرفته شده است که طراحی واحدمند[8] کلید برنامه‌سازی موفق است و زبانهای جدید مانند ماجولا-۲، ایدا و ML استاندارد شامل ویژگیهایی هستند که به‌طور اختصاصی برای کمک به پیشرفت و بهبود واحدمندی طراحی شده‌اند.

```
sum []=0
sum (num: list)=num+sum list
```

محاسبهٔ sum را می‌توان با استفاده از کنار هم گذاشتن یک الگوی بازگشتی عمومی و قسمتهای بسته‌بندی شده واحدبندی کرد. این الگوی بازگشتی به‌طور قراردادی «reduced» نامیده می‌شود.

```
sum =reduce (+)0
```

آقای هیوز به توانایی برنامه‌سازی تابعی تأکید می‌کند و آنرا با استفاده از مثالهای بدی با نمونه‌های دیگر مقایسه می‌کند و هیچ کوششی برای نشان دادن عدم کارایی در زمان اجرا ندارد.

نکتهٔ دیگر امکان تبدیل برنامه‌های امری[9] به برنامه‌های تابعی، در صورت نیاز است یعنی اگر مناسب مسئله‌ای مانند تجزیه‌گر بازگشتی کاهشی باشد[10]

توانایی ما برای تجزیه یک مسأله به بخشها، مستقیماً به توانایی ما برای کنار هم قرار دادن راه‌حل بستگی دارد. یک زبان برای کمک به برنامه‌سازی واحدمند باید این امکان را بخوبی دارا باشد. زبانهای برنامه‌سازی تابعی دو شگرد جدید برای این کار فراهم می‌کنند: توابع از مرتبهٔ بالاتر و ارزیابی تنبل[11]. بااستفاده از این امکانات می‌توان برنامه‌ها را به روشهای جدید و جالب واحدمند کرد که مثالهای زیادی از این مطلب را نشان داده‌ایم.

6) Why functional programming matters 7) J. Hughes

8) modular design 9) imperative programs 10) recursive descendent parser 11) lazy evaluation

واحدهای کوچکتر و عمومیتر را می‌توان به‌طور گسترده‌تری به‌کار برد، که این عمل برنامه‌سازی بعدی را تسهیل می‌کند. به این دلیل است که برنامه‌های تابعی در مقایسه با برنامه‌های رایج، بسیار کوچکتر بوده و نوشتن آنها خیلی ساده‌تر است.

برای من برنامه‌سازی واحدمند روشی است که در آن می‌توان قسمتهای مسأله را از هم جدا کرد، برای اینکه آزمایش شده، اشکال‌زدایی شود و بسادگی جمع شود. من معتقدم برای اینکه تجزیهٔ ما کار کند، ما نیاز به هیچ نوع شگردی نداریم. آنچه که ما نیاز داریم روشی بسیار متفاوت برای تجزیهٔ مسائلمان است. برای مثال تجزیه شیء. قابلیت استفاده مجدد، مسأله‌ای است که می‌توان در مورد آن بسیار بحث کرد. آنچه که ما دربارهٔ قابلیت استفاده مجدد می‌فهمیم این نیست که فقط قسمتی از برنامه را کپی کرده و بچسبانیم بلکه به این مفهوم است که بتوانیم برنامه را به موازات کاربرانش نگهداری کرده و توسعه دهیم. چیزی که برنامه‌سازی تابعی به آن نزدیک نشده است. برنامه‌های تابعی به این دلیل کوتاه هستند که بر اساس بازگشت‌پذیری نوشته شده‌اند و هر برنامه‌سازی باید به این حقیقت گواهی دهد که اشکالزدایی چنین برنامه‌هایی بسیار دشوار و کارایی آنها بسیار کم است. بعضیها ممکن است بگویند استدلال ریاضی برنامه‌ای که به زبانهای تابعی نوشته شده است ساده است. در این صورت نظر من این خواهد بود که نه همگی ما ریاضیدان هستیم و نه می‌خواهیم که باشیم. اگر به عهدهٔ ریاضی‌دانان بود باید هنوز به زبان اسمبلی برنامه می‌نوشتیم و علم کامپیوتر مانند فیزیک هسته‌ای می‌شد. و بالاخره اگر کسی بتواند یک سیستم پنجره‌ای یا یک سیستم بلادرنگ مانند کنترل ترافیک هوایی را به من نشان دهد که کاملاً به زبان تابعی نوشته شده باشد، من متقاعد می‌شوم که برنامه‌سازی تابعی پاسخ نهایی به تمام سؤالات ماست.

توضیحاتی بر «برنامه‌سازی: جادو یا علم»[12] اثر سی. هور[13]

یکی از مسائل عمدهٔ حرفهٔ برنامه‌سازی آن است که تصمیمات تکنیکی و ساختاری ما تقریباً نمایان هستند، چیزی را که در برنامهٔ پایان‌یافته قابل مشاهده باشد نمی‌توان از قبل با استفاده از تصاویر نشان داد.

«برنامهٔ پایان‌یافته» عبارتی است که من فکر می‌کنم باید بسیار بااحتیاط تعریف شود. برنامهٔ پایان‌یافته نه تنها تعدادی پرونده مبدأ و پرونده قابل اجرای کامپایل شدهٔ آنهاست، بلکه مجموعه‌ای از اوراق طراحی، مستندات توصیفی و بعضی توضیحات قوی ادغام شده در برنامه را نیز در برمی‌گیرد. من معتقدم آقای هور وقتی مطالبش را می‌نوشته از نوشتهٔ گردی بوچ خبر نداشته است. مانند هر چیز دیگر، علم کامپیوتر به‌وسیلهٔ واقع‌گرا و قابل فهم بودن سعی در توسعهٔ خود دارد. بوچ دو روش را برای فائق آمدن بر مشخصات سیستم مورد سؤال پیشنهاد می‌کند: اول، نمودار ساختار رده‌ها یعنی ارتباطی کامل بین قالبها و موجودیتها و دوم، نمودار شیء‌که تصویر لحظه‌ای سیستم در زمان اجراست.

این حقیقت که برنامه‌ای برای مقدار صفر و ۶۵۵۳۵ کار می‌کند، هیچ اطمینانی نمی‌دهد که برنامه برای هر مقداری بین آنها نیز کار خواهد کرد، مگر اینکه این حقیقت با استفاده از دلایل منطقی که بر پایهٔ متن برنامه است ثابت شود. اما اگر این استدلال منطقی صحیح باشد دیگر نیازی به آزمایش نداریم. به این دلیل است که یکی از شرایط اولیه اساسی برای پیشرفت تجربیات حرفه‌ای ما این است که بیاموزیم چگونه دربارهٔ برنامه‌هایمان استدلال کنیم، صحت برنامه‌ها را اثبات کنیم قبل از اینکه آنها را بنویسیم، به این منظور که بدانیم برنامه‌ها نه تنها آزمایشها را خواهند گذراند، بلکه بعد از آن برای همیشه به‌صورت صحیح به‌کار ادامه خواهند داد.

توضیحاتی بر «آیا برنامه‌سازی را می‌توان از شیوهٔ فون نویمان رهانید؟ شیوهٔ تابعی و جبر برنامه‌های آن.»[14] اثر بکوس[15]

به‌نظر می‌رسد هر زبان جدید با مشکل مواجه است. زبان بعدی ـ با کمی بهبود ـ تمام ویژگیهای زبانهای قبل از خود را شامل می‌شود بعلاوهٔ تعدادی ویژگی جدید بیشتر.

یک نکتهٔ عالی. ولی من عیب عمده‌ای در این فرارونده نمی‌بینم. انسان ایده‌هایش را به‌وسیلهٔ روشهای تکراری و افزایشی توسعه داده است، یعنی ما با آزمون و خطا یاد می‌گیریم. نکته دیگر آن است که انسان طبیعتاً در حال تکامل است و با این تکامل، نیازها و احتیاجاتش نیز باید متکامل شوند.

هر زبان جدید مدعی ویژگیهای جدید است مانند گونه‌بندی قوی یا احکام کنترل ساختارها اما حقیقت روشن و صریح این است که زبانهای کمی وجود دارند که برنامه‌سازی را ارزانتر و قابل اعتمادتر سازند به‌نحوی که بتوان هزینهٔ تولید و آموزش آنها را توجیه کرد.

نکته خوب انتخاب شده است، آنچه که اساساً نیاز داریم یک نمونهٔ جدید است که ما را قادر کند از همان نوع ساختار درونی و پایه‌ای استفاده کنیم که در محیط پیچیدهٔ ما برای هزاران سال مورد استفاده قرار گرفته است. توانایی استفادهٔ مجدد از هر پیچیدگی بعد از اینکه خلق شد و توانایی ادغام

12) Programming: Sorceny or Science 13) C. Hoare

14) can programming be liberated from the von neumen style? A Functionalstyle and its Algebra of programs 15) Backus

چنین پیچیدگیها باید مهمترین هدف ما باشد. اکنون، کافی است بگوئیم که چنین نمونه‌ای وجود دارد و آن نمونهٔ شیءگرا می‌باشد.

بقیهٔ مطلب آقای بکوس تمام نکاتی را که ما قبلاً عنوان کرده‌ایم ذکر می‌کند. بنابراین، تمام ایرادهایی که من گرفته‌ام هنوز معتبر است. این نکته هرگونه توضیح روی باقیماندهٔ مطلب را تکراری و خسته‌کننده خواهد کرد. بنابراین از بازبینی بیشتر مطلب صرفنظر کردم.

دیدگاههای من

من مرعوب قدمت برخی از این مقاله‌ها شده بودم. با در نظرگرفتن سرعت پیشرفت در علم کامپیوتر، ۱۰ سال زمان زیادی است و علم، قدیمی و از رده خارج در نظرگرفته می‌شود. نکتهٔ دیگر من این است که این مقالات هر چند بسیار ارزشمند هستند ولی به مسائل جدیدی که اکنون رویاروی ما قرار دارد نمی‌پردازند. صرف‌نظر از اینکه شما چقدر باهوش و عاقل هستید، اگر به مسائل پرواز بپردازید و مسائلی که با آنها مواجه هستید مربوط به صدها سال پیش باشد، راه‌حل شما از نظر دانشمندان علم فضانوردی زمان حاضر شایستگی توجه و علاقه را نخواهد داشت.

تمام نویسندگان تحت تأثیر مطالبی هستند که در مراحل اولیه آموزش خود یعنی دهه ۱۹۶۰ با آنها سر و کار داشته‌اند. از مسائل امروزه کاملاً آگاه نیستند. مسائلی که ارائه شد هرکدام به طریقی قدیمی و یا نامعتبر هستند، برای مثال برنامه‌سازی ساختیافته، برنامه‌سازی واحدمند وغیره.

من معتقدم برنامه‌سازی تابعی بیش از حد لازم مورد توجه قرارگرفته است و دو دلیل برای این وجود دارد:

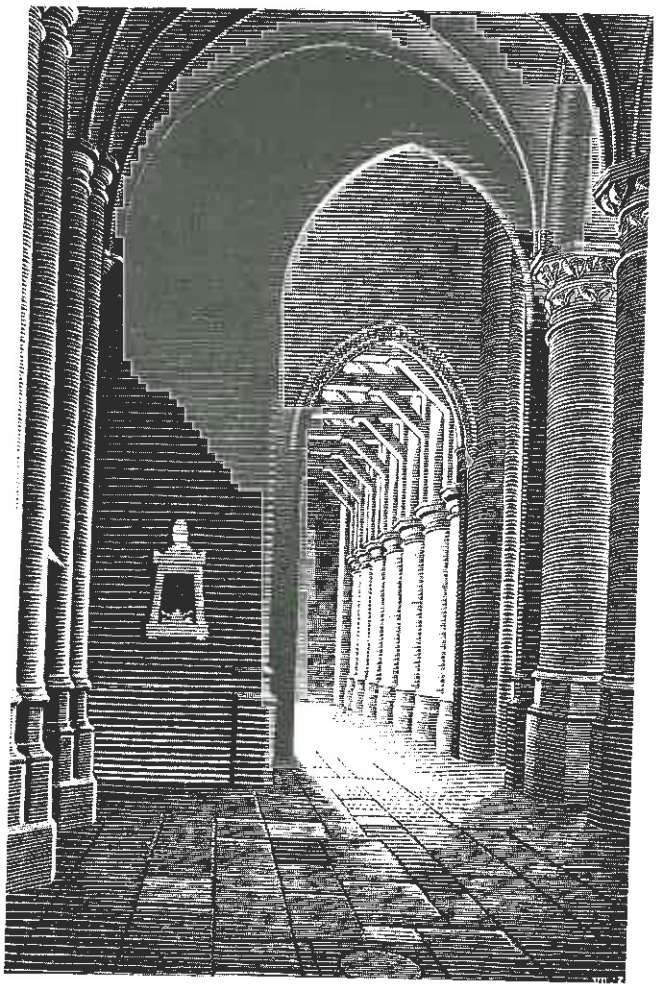
- در اواخر دههٔ ۱۹۶۰ و اواسط دههٔ ۱۹۷۰ ما با مسأله‌ای به‌نام بحران نرم‌افزار مواجه بودیم. بحران نرم‌افزار نتیجهٔ بسیاری از مسائل انباشته شده از جمله استفاده از gotoها و متغیرهای سراسری بود. برنامه‌سازی تابعی این مسأله را به این صورت حل می‌کند که در برنامه اجازهٔ دستورالعملهای جایگذاری نمی‌دهد و ورود و خروج برنامه را به نقاطی مشخص محدود می‌کند.
- دلیل دیگری برای اینکه برنامه‌سازی تابعی به‌عنوان راه‌حل شناخته شد این حقیقت است که این روش با تجزیهٔ تابعی و روشهای از بالا به پائین مطابقت دارد.

امروزه استفاده از این روشها برای پرداختن به پیچیدگیهایی که در بعضی از کاربردها وجود دارند، بسیار پر زحمت است. استفاده از این روشها ماهیتاً به‌جای اینکه مسأله را قابل استفادهٔ مجدد کند باعث بزرگتر شدن و پیچیده‌تر شدن آن می‌شود. اجازه بدهید روی دو نکته نتیجه‌گیری کنیم.

۱ـ ما نیازمند روشهای بهتری برای تعریف مسائل موجود هستیم و نباید به هر مسأله‌ای فقط به‌خاطر اینکه ریشه‌های تاریخی دارد بپردازیم. همه چیز با زمان تغییر می‌کند و مهندسی نرم‌افزار هم همینطور است. بنابراین باید هوشیار بوده و در انتظار گوش دادن به کسی باشیم که رهیافت جدیدی را مطرح خواهد کرد. (برای مثال رهیافت انسان‌گرا)

۲ـ زبانها و ابزاری که در طی سالیان متمادی از آنها استفاده کرده‌ایم دیگر کارایی ندارند. این یک جملهٔ گستاخانه نیست. ما از بعضی از زبانهای از میان رفته مانند فرترن و کوبول استفاده می‌کنیم زیرا میلیونها خط برنامه به این زبانها نوشته شده است در حالیکه زبانهایی مانند self و clos و بسیاری دیگر که برای من ناشناخته هستند، فقط در گروههای کوچکی به‌کار رفته‌اند که ارزش آنها را درک می‌کنند. اجازه بدهید به یاد داشته باشیم این محدودیت خودمان نیست بلکه محدودیتهای دیگر مانند محدودیتهای اجتماعی هستند که بر ما تحمیل شده و اجازه پیشرفت را به ما نمی‌دهند، مگر اینکه تمام آنها را کاملاً کنار بگذاریم.

●



وراثت و گوشه‌ای از کاربرد آن در C++

محمودرضا زارع

دانشجوی نرم‌افزار دانشگاه شهید بهشتی

ط

چکیده

در این مقاله سعی می‌کنیم چگونگی استفاده از ویژگی وراثت برای قراردادن اشیائی از گونه‌های مختلف را در یک لیست پیوندی تشریح کنیم.

کلمات کلیدی: رده[1]، وراثت[2]، ردهٔ پایه[3]، نمونه[4]، لیست پیوندی[5]، گره[6]، شیء[7]، شیءگرا[8]، اشاره‌گر[9]، گونه[10]

مقدمه

یکی از مهمترین ویژگیهای یک زبان برنامه‌سازی شیءگرا وراثت است. وراثت در واقع مکانیسمی است برای آن که بتوانیم بهتر و ساده‌تر دنیای واقعی را در قالب یک برنامه بیان نموده و در میزان برنامه‌سازی و نیز اندازهٔ برنامه‌ها صرفه‌جویی کنیم. بنابراین یک برنامه‌ساز شیءگرا باید از همان مراحل اولیه طراحی برنامه به‌استفاده از این ویژگی توجه کافی داشته باشد.

هدف ما از این مقاله آشنایی با مفاهیم شیءگرایی و یا شرح مزایای وراثت نیست، بلکه فرض می‌کنیم خواننده با این مفاهیم و نیز پیاده‌سازی آنها در زبان C++ به اندازه کافی آشنایی دارد و سعی می‌کنیم جنبه‌هایی ازکاربرد وراثت در C++ را بیان کنیم که معمولاً درکتابهای مقدماتی این زبان به‌این شیوه یافت نمی‌شوند ولی دانستن آنها برای هر برنامه‌ساز C++ اگر الزامی نباشد خالی از فایده نیست.

مسائل لیست پیوندی

یکی از جنبه‌های قوی و سهل‌الاستفادهٔ زبان C++ (که در واقع از C به‌ارث رسیده است) استفاده از اشاره‌گرها می‌باشد. به‌عنوان مثال برنامهٔ ۱ را در نظر بگیرید. در این برنامه نخست یک رده به‌نام circle (که نشان‌دهندهٔ یک دایرهٔ ساده است) معرفی شده و سپس در ابتدای ()main یک لیست پیوندی با دوگره از نمونه‌های آن ساخته می‌شود. در پایان نیز یک اشاره‌گر

1) class 2) Inheritance 3) base class 4) instance
5) linked list 6) node 7) object 8) object-oriented
9) pointer 10) type

برنامه ۱

```
# include <stdlib.h>
class circle
{
    int x , y ; // centre coordinates
    float radius ;
public :
    circle *next ;
    circle(int c_x , int c_y , float r)
    {
        x = c_x ;
        y = c_y ;
        radius = r ;
        next = NULL ;
    }
    void draw()
} ;
void main()
{
    circle *first ;
    // build list
    first = new circle(3,3,4.2) ;
    first → next = new circle(3,4,3.2) ;
    // draw list
    circle *t = first ;
    while(t)
    {
        t → draw() ;
        t = t → next ;
    }
} // main
```

به‌نام t لیست ساخته شده را پیموده و تابع draw() مربوط به هر گره را فراخوانی می‌کند. در اینجا تمرکز ما روی چگونگی رسم دایره نیست. لذا از جزئیات پیاده‌سازی تابع circle::draw() چشم‌پوشی کرده‌ایم.

همانطورکه ملاحظه می‌کنید در برنامه ۱ همهٔ گره‌های لیست پیوندی از جنس circle هستند و بنابراین متغیر next از ردهٔ circle و نیز متغیرهای first و t در تابع main() اشاره‌گر به‌همان ردهٔ circle معرفی شده‌اند. ولی همیشه وضعیت به‌همین سادگی نیست، بلکه گاهی نیاز داریم گره‌های لیست پیوندی از گونه‌های متفاوت باشند. به‌عنوان مثال یک برنامه CAD[11] را درنظر بگیرید که همه اشیاء آن در یک لیست پیوندی قرار دارند. در این لیست امکان دارد اشیائی از جنس خط، دایره یا مربع به‌هر ترتیبی وجود داشته باشند.

برای درک بهتر مطلب به‌برنامه ۲ نگاهی بیاندازید. در این برنامه نخست ردهٔ جدیدی به‌نام rect معرفی شده و سپس در تابع main() سعی می‌شود یک لیست پیوندی با گره‌هایی از جنس circle یا rect ایجاد شود. ولی همانطورکه در متن برنامه نیز مشخص شده، در خط زیر خطا وجود دارد.

```
first ->next = new rect(3,3,4,4); // error
```

زیرا متغیر next از ردهٔ circle اشاره‌گر به circle معرفی شده و نمی‌تواند به یک نمونه از ردهٔ rect اشاره کند. البته علی‌الظاهر می‌توان این اشکال را با استفاده از type cast به‌صورت زیر برطرف نمود.

```
first ->next = (circle*) new rect(3,3,4,4);
```

ولی دقت داشته باشید که در یک برنامهٔ واقعی مانند CAD لیست پیوندی در زمان اجرا ساخته می‌شود و هنگام برنامه‌سازی مشخص نیست که عنصر بعدی لیست از چه جنسی خواهد بود تا بتوان از type cast استفاده نمود*

اشاره‌گر به ردهٔ پایه

تا اینجای کار دیدیم که چگونه قراردادن اشیائی از رده‌های مختلف در یک لیست واحد پیوندی ایجاد اشکال می‌کند. در اینجا راه حلی که C++ برای حل مشکل فوق در اختیار می‌گذارد را معرفی می‌کنیم. وراثت، راه حل C++ برای حل این مشکل است. این راه حل از آنجا آغاز می‌شود که اشاره‌گر به‌ردهٔ پایه می‌تواند برای اشاره کردن به یک رده مشتق شده[12] از آن نیز به‌کار رود. به‌این ترتیب می‌توان اشکال برنامه ۲ را به‌صورت برنامه ۳ مرتفع نمود.

برنامه ۲

```
# include <stdlib.h>
class circle
{
    int x , y ; // centre coordinates
    float radius ;
public :
    circle *next ;
    circle(int c_x , int c_y , float r)
    {
        x = c_x ;
        y = c_y ;
        radius = r ;
        next = NULL ;
    }
    void draw()
} ;
class rect
{
    int x1 , y1 , // up left corner
        x2 , y2 ; // down right corner
public :
    rect *next ;
    rect(int x1 , int y1 , int x2 , int y2)
    {
        rect::x1 = x1 ;
        rect::y1 = y1 ;
        rect::x2 = x2 ;
        rect::y2 = y2 ;
        next = NULL ;
    }
    void draw()
} ;
void main()
{
    circle *first ;
    // build list
    first = new circle(3,3,4.2) ;
    first → next = new circle(3,3,4,4) ; // error
    // draw list
    circle *t = first ;
    while(t)
    {
        t → draw() ;
        t = t → next ;
    }
} // main
```

11) Computer Aided Design

*) حتی‌اگر برنامه ۲ را به‌صورت فوق و با استفاده از type cast تصحیح کنید اشکالات کامپایل آن برطرف خواهد شد ولی ـ در صورت پیاده‌سازی کامل توابع draw() ـ در زمان اجرا مشکلاتی بروز خواهد کرد که به‌امتحان آن می‌ارزند.

12) derived class

برنامه ۳

```
# include <stdlib.h>
class base
{
    public :
        base *next ;
        base () next = NULL ;
        void draw()
} ;
class circle : public base
{
        int x , y ; // centre coordinates
        float radius ;
    public :
        circle(int c_x , int c_y , float r)
        {
            x = c_x ;
            y = c_y ;
            radius = r ;
        }
        void draw()
} ;
class rect : public base
{
        int x1 , y1 , // up left corner
            x2 , y2 ; // down right corner
    public :
        rect(int x1 , int y1 , int x2 , int y2)
    {
            rect::x1 = x1 ;
            rect::y1 = y1 ;
            rect::x2 = x2 ;
            rect::y2 = y2 ;
        }
        void draw()
} ;
void main()
{
    base *first ;
    // build list
    first = new circle(3,3,4.2) ;
    first → next = new rect(3,3,4,4) ;
    // draw list
    base *t = first ;
    while(t)
    {
        t → draw() ;
        t = t → next ;
    }
} // main
```

در برنامه ۳ نخست یک ردهٔ پایه به‌نام base معرفی شده، آنگاه دو ردهٔ circle و rect از آن مشتق می‌شوند. آنگاه در تابع ()main یک لیست پیوندی که گره‌های آن از جنس circle یا rect هستند ساخته می‌شود. همانطورکه ملاحظه می‌کنید متغیر next از ردهٔ base و نیز متغیرهای first و t در تابع ()main همگی اشاره‌گر به‌ردهٔ base معرفی شده‌اند و بنابراین می‌توانند به‌رده‌های مشتق‌شده از آن یعنی circle و rect نیز اشاره کنند. به‌این ترتیب حتی در یک برنامه واقعی مانند CAD که جنس گره بعدی از قبل مشخص نیست نیز اشکالی بروز نمی‌کند.

●

فردی به عنوان اپراتور کامپیوتر ترجیحاً لیسانس مرتبط، برای مسئولیت نگهداری یک سیستم سادهٔ کامپیوتری نیازمندیم. متقاضیان می‌توانند با صندوق پستی شماره ۱۹۳۹۵/۳۵۶۴ مکاتبه فرمایند.

OOPG;

The Object-Oriented Practitioner's Group

Announcement. The Informatics Society of Iran would like to announce the creation of yet another interest group in the Society, entitled the "Object-oriented Practitioner's Group", under the supervision of Mr. Ali Arsanjani and Mr. Ramin Erfanian. The Group has had two gatherings of 38 and 28 people during its first two montly sessions starting from November, 1994 (Azar, 1373). The primary aims of this subgroup are outlined below:

Aims.

1. Elevating the technical awareness of enthusiasts and practitioners of OO

1.1. Application (Practical) Aspect: The benefits and problems of employment of object-oriented methods; correct use of the technology and refraining from its misuse.

1.1.1. Facilitating the introduction of concepts, techniques, tools, methodologies on begginer, intermediate and advanced levels, as proposed and utilzed by practitioners and researchers.

1.1.2. Promoting the usage of OO model and methodologies, techniques, environments and languages in an effort to elevate the quality of software developed and produced in this country.

1.2. Theoretical (Conceptual) Aspect: promotion of the meaning and concepts of the object-oriented paradigm.

1.3. Creation of a goal-oriented trend for the cultivation of a software engineering object-orientation culture for the understanding and usage of OO methods.

2. Creating an academic atmosphere conductive to the exchange of software experiences, ideas and methods pertaining to object-orientation:

2.1. Motivating the interaction between practitioners and between enthusiasts in an effort to give this technology wider popularity.

2.2. Creating an atmosphere that is conductive to research effort in the areas of the object paradigm, and motivating innovation and new methods of usage.

2.2.1. Creating a repository of OO resources and sevices: films, software, articles, books, experience logs, ets.

3. Acting as the contact point with internationally renowned object-orientation groups.

Agenda. There is a monthly speech of about fourty minutes on a topic of interest according to the priorities set by the participants via an initial questionnaire. The last two topics have been "The Benefits of OO" and "General Concepts of Object-orientation" by Ramin Erfanian and Ali Arsanjani respectively. There will be a monthly, fifteen to twenty minute talk on general concepts of object-orientation in question and answer form. New members are welcome to join in for papers, seminars, email conferencing, posing questions, ets.

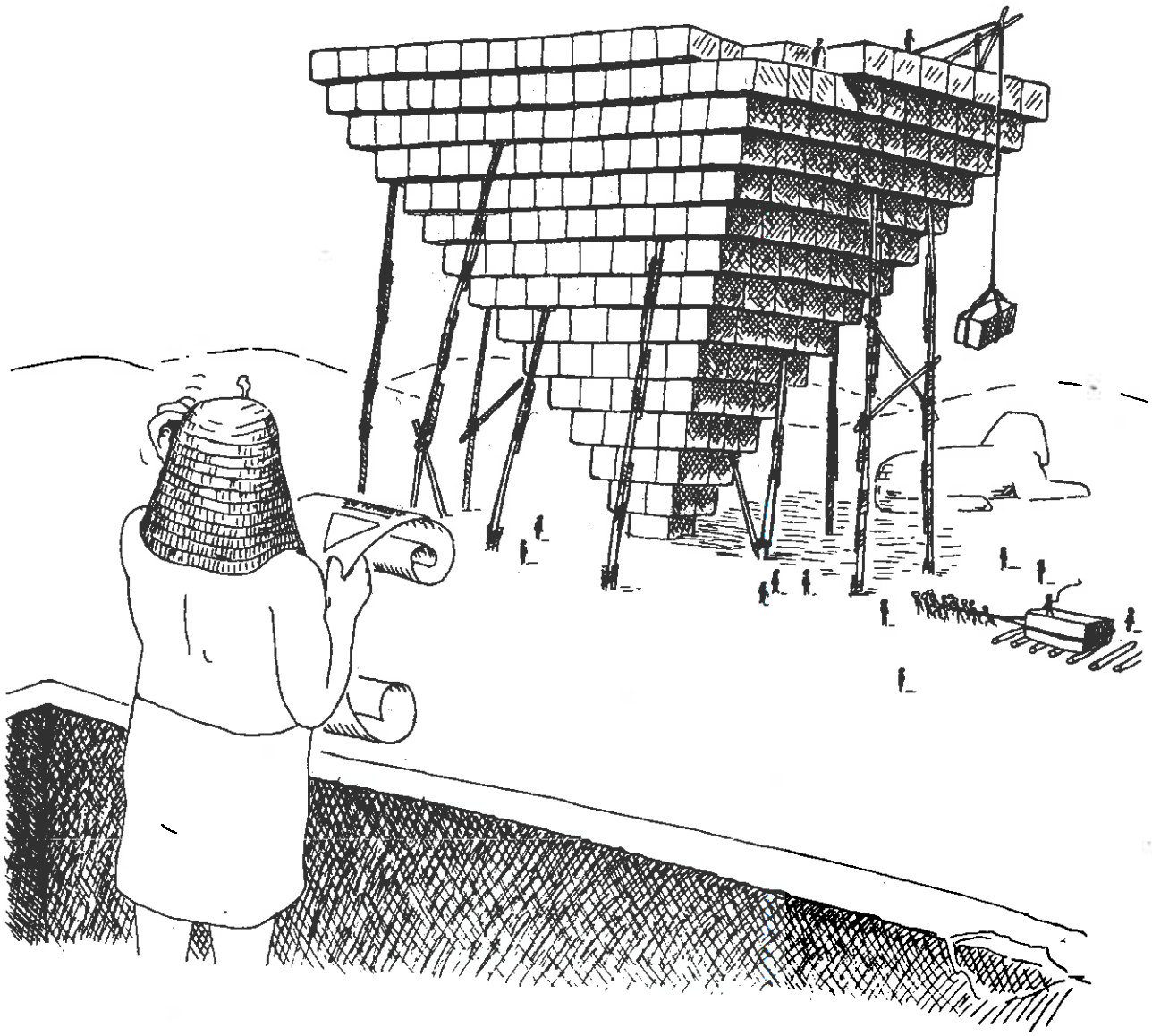
quote the "round-trip gestalt" process of the OO life-cycle. It appears that there is (or I haven't discovered!) no precise formulation of this matter in the literature. I would to have your viewpoints on what would constitute a more precise, pragmatic and usable refinement or elaboration of the OO development process model.

This is what I'm trying to write about in my next book, "object solutions." my goal is to identify a process that is certifiable by iso 9000 processes, as well as moves up to at least level 3 in the sei cmm.

7) What are your views on the future directions of software development in general and Object-Orientation in particular? Do you believe that Object-Orientation will be a permanent software engineering phenomenon?

These are interesting times for software developers; the increasing, world wide demand for software, plus the pervasiveness of computing, plus the price/perofrmance of distributed sytems creates an insatiable demand for software. software development is hard. I expect it to get harder, but oo helps make it somewhat easier. I predict OO will be around for a long, long time.

Thanks again for your time.



INTERVIEW

An Interview with
Grady Booch

Chief Scientist
Rational Software Corporation

This interview has been conducted through e-mail.

1) Mr.Booch, can you please recount for us the history of your involvement with and pioneering efforts on Object-Orientation; the languages and projects you were initially involved in. What are your current research/work interests and what type of projects with which you are involved?

I published perhaps the first paper on Object-Oriented design in 1982. at that time, I was in the US Air Force, working on a variety of large,complex telemetry systems in support of the shuttle program. shortly thereafter, I became involved in the Ada program, wherein I helped devise methods to effectively use the language. in the mid 80's, I got involved in C++ and smalltalk, and found my work was directly applicable to developing systems in those languages. in 1990, I published one of the first books on object-Oriented analysis and design; this book was revised to a second edition in 1994.

My current work centeres primarily on complex systems. much of my time these days is in helping large organizations architect their OO systems

The booch method is perhaps the most wiedely-accepted method world-wide it represents a seocnd generation method, which embraces the work of many other methodologies

2) Many Object-Oriented methodologies have evolved from structured techniques and entity-relationship notions. What are your views on the relationship between structured techniques and methods, entity-relationship modeling and Object-Orientation? How much do you think Object-Orientation is influenced by other paradigms and techniques?

It really is evolving on its own. There is a distinct split between oo and structured techniqeus, less so between er modeling and oo. in the former case, there are fundamental differences in philosophy. in the latter case, it is fair to say that er modeling has influenced OO methods, but OO methods go far beyond this, by adding issues of inheritence and behavior.

3) The Object-Oriented learning curve is somewhat steep. But the prospects and benefits of OO are quite attractive for many organizations. What are your suggestions for these organizations that want to migrate to Object technology and Object-Oriented techniques? How best should they go about doing this, and in your experience, how long do you think this usually takes?

Migrating to OO techniques takes an individaul perhaps 6–9 months to really understand the paradigm. this is best done through pilot projects.

4) You have a rich store of experience in being involved in lrage-scale Object-Oriented projects. Each of these projects teaches many things, as one encounters new problems, and new ways of solving them through robustness and flexibility in applying the Object-Oriented paradigm and its techniques. Can you share some of your experiences with us on correct and incorrect applications of Object-Oriented techniques and methods?

At the moment, I'm working on a book, titled "object solutions: managing the Object-Oriented project" and have in it some 300 stories from real projects, so it's hard to select just one. Some of the stories I tell are of succesful projects; in all fairness, some of the stories I tell are of failures — but peope can learn from their failures as well as their successes.

As for failures: I'm reminded of one large telephony project that was using C++ for the first time; I examined their code — they had written several hundred thousand lines of C++ and didn't have a single class in the entire system! Clearly, they had missed the point of oo.

As for successes: I'm curently involved in one of this country's largest smalltalk applications — it's an international effort, to be deployed world wide, dealing with the transporation industry. I've seen the group migrate from cobol to smalltalk and C++ — they simply could not have done what they've achieved otherwise.

5) There are a vast number of Object-Oriented methodologies poping up in the literature. What are your views on this proliferation and which ones would you recommend to be advocated? Which ones are more practical and have been known to be usable throughout many projects?

There are too many OO methods. in my experience, there are only a few really interesting and world class ones: booch, omt, shlaer-mellor, and jacobson are on my top list. I'm working on achieving a unification of these best methods, and my current work reflects taht

6) There is a repeated mentioning and emphasis, by most authors on OO, about the"iterative" and "incremental" nature of the software development process model. In your excellent book, OO Design with Applications, you

(early 1963), it was fairly easy to submit programs as card decks for the IBM 7090 and get printed output back, usually within half a day. I bought a book on FORTRAN and quickly taught myself the basics of programming. That was enough background for me to convince the Cyclotron Group that I should write programs over the summer of 1963 to solve their data-reduction problems. I threw myself into learning about numerical analysis, curve fitting, drawing graphs, and who knows what else. I ended up working for them year round until the day I graduated. (And that experience led me to join a general migration to Michigan State – where there was a new cycltron and a new on-line data-acquisition computer - - to do more of the same.)

3) How are things in C Journal?

The C Users Journal is doing quite well. We are about to change the name to the C/C++ Users Journal, in simple recognition of the mix of coverage we now provide. I believe that we are now a principal source of pragmatic information on programmin in C++ (as opposed to sermons preaching the superiority of object oriented programming and design).

4) Why is that you are member of both C++ and C ANSI standardization committees?

Both are important languages, with overlapping but different areas of application. And both are languages where the standardization effort is central to the future direction of the language. I like to be where the action is.

5) You always emphasize on programming for a purpose vs programming for fun, is it because you do not like programming or it has a more profound reason?

I like programming very much. I just feel there is so much to do that I begrudge any time spent purely on play. If others want to "hack," that's fine with me. But I don't want to and I see no need to help hackers or to glorify their pastime.

6) Tell us your nicest story about being a computer guru?

I recently got to help out as an expert withness in a copyright infringement case. Other experts hired by the plaintiff (whose software was misappropriated) had done a fine job of proving infringement, but the defendant managed to hire still other experts to muddy the waters. I was able to apply my knowledge of C and of software tools – with just a few hours of work – to provide independent supporting evidence that the software was indeed copied. The defendants gave up halfway through the trial and agreed to meet all the terms of the plaintiff. And I only had to spend five minutes on the witness stand to make a difference.

7) Is there any feature in either C or C++ languages that you could claim their invention?

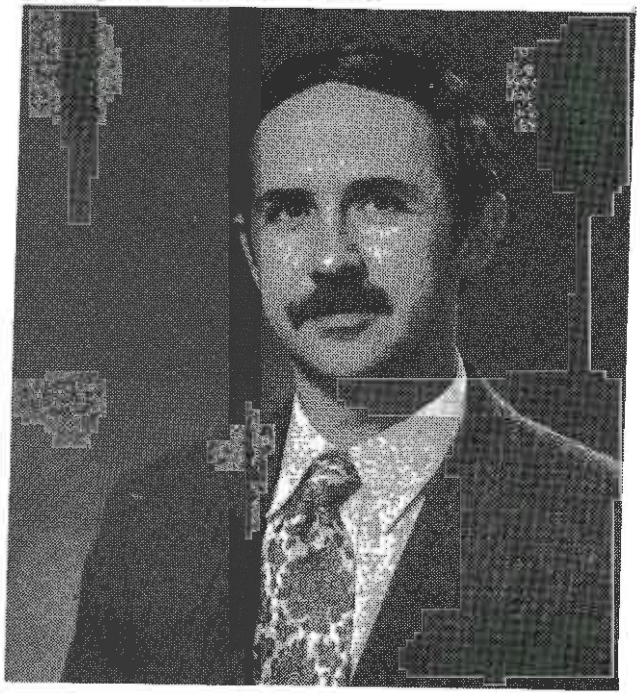
Only small things. And I'm not sure thery're my invention so much as my clarification. I insisted, for example, that the address - of operator apply to array and incomplete types, so & a has type "pointer to array of . . . " It took a lot of arguing to explain the need for such orthogonality, but now people accept it as a commonplace.

I invented the terms "incomplete type" and "translation unit," among others, to better explain how C is processed. And as far as I know, I worte the precursor to what is now the atexit function.

Within C++, all I've managed to do is introduce some functionality into exception classes for programming embedded systems (a feature that is now under attack), and get some of the larger headers broken up into smaller ones. My more ambitious proposals have uniformly met with heated opposition from one or two quarters, so I have stopped trying to innovate. My major contribution to the draft C++ standard has been to write out the entire library clause in uniform notation that is also reasonable "standardese."

8) You have written a few books. What were the motivations for doing so?

I like explaining things to people. I also like writing books as a way to earn money – you work hard for most of a year, then watch money come dribbling in for years afterward, if you get ahead enough, you can coast on past royalties from time to time.



9) Software industry in Iran is in its primary stages, what would be your wisest programming advice to the newcomers to this professions?

As programmers, concentrate on learning how to produce high quality custom software reliably.

continued on Page 5

INTERVIEW

An Interview with

P.J. Plauger

Senior Editor of the C Users Journal

This interview has been conducted through e-mail.

1) Some of our readers may not know you, could you introduce yourself, your academic and industrial background?

I got involved with computers as a sophomore at Princeton University. Even though I majored in physics (AB, 1965), I spent much of my time programming computers for the Princeton University Cyclotron Group. I then earned a Ph.D. in Nuclear Physics (Michigan State University, 1969), but once again I put at least as much energy into programming as I did into learning physics. So I was recruited by AT& TBell Labs as a computer scientist. I began work at Bell Labs, Holmdel, New Jersy, in late 1969.

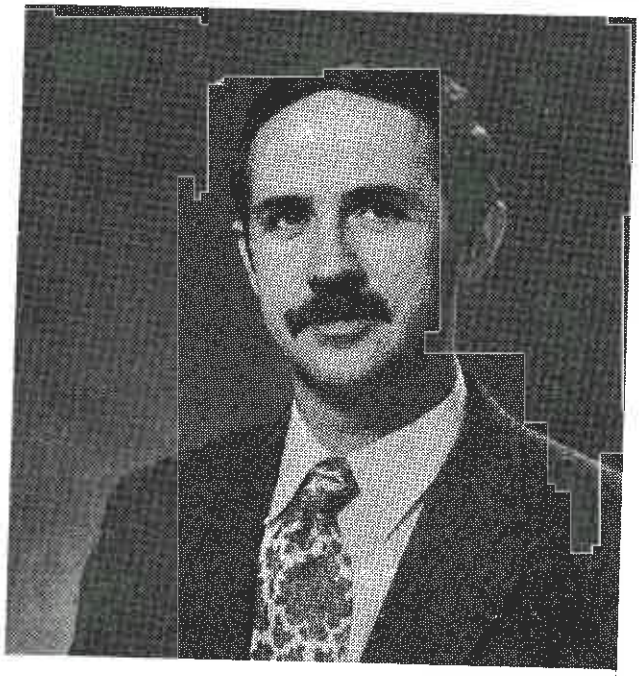
I was soon transferred to Murray Hill, New Jersey, however, where I found myself surrounded by the originators of the UNIX operating system and the C programming language. Brian Kernighan and I had offices next door to each other. That led to our writing our first book, "The Elements of Programming Style" (McGraw Hill 1974). It became an instant hit and gave us both a taste for writing that continues today. Brian and I went on to write "Software Tools" together, even as I was leaving Bell Labs in mid 1975.

I then joined Yourdon Inc., a seminar and consulting firm founded by Ed Yourdon. There, I discovered I had good skills as a seminar leader for the next several years. I lectuced on structured programming, structured Design, etc. all around the world. I became Vice president at Yourdon within about six months of joining the company.

At Yourdon Inc., I also wrote my first commercial C compiler, for a company in Concord, Massachusetts who wanted C under RSX-11M on a PDP-11, but not under UNIX. Yourdon had some success selling this compiler as a product, but I saw greater potential. I left Yourdon Inc. in mid 1978 to found Whitesmiths, Ltd.

At Whitesmiths, I quickly produced a new PDP-11 C compiler. It sold well and was sufficiently portable that I got it running on the Intel 8080/8085 (Zilog Z80) and on the VAX by the end of 1979. Eventually, that compiler was ported to the Motorola MC68000 family, the Intel 8086 family, and IBM System/370 by me and others at Whitesmiths. We also produced a pascal-to-C translator and the first UNIX-equivalent operating system, called Idris.

Our early and rapid successes soon led to infighting within the company, however. By the time I settled the differences and bought out my partners, competition had caught up with us. I spent most of the mid 1980s struggling to regain our earlier growth , without



success. I soon determined that running a software company of 30-40 people was too much of a drain on me. So in 1988, I seized the opportunity to sell Whitesmiths to Intermetrics, Inc. of Cambridge, Massachusetts. After working for the buyers for a year, as agreed at the sale, I left to become an independent writer.

Since 1989, I've spent one year as a visiting lecturer in Australia (University of New South Wales, 1991). Otherwise, I work at home and travel approximately one week a month. I've long been active in software standardization – first POSIX, then C, and now both C and C++. I write about 30 articles per year for several computer magazines and am Senior Editor of the C Users Journal."

2) How did you start your career as a programmer?

Princeton University had half a dozen computers around campus when I was there. By late in my sophomore year

The bit manipulation error took the 'Windows' logo ASCII text screen and changed two bits to make it 'Sindogs'. This became known as the 'Sindogs' bug and we named one of our development servers in honor of that bug.

Is there any feature in either C or C++ languages that you could claim their invention?

No – I'm not a language lawyer type. The C++ language already has enough cooks. My claim to fame is that we took the complex C++ language and came up with what we call a 'sane subset'. This is a convention of using the main features of the language that support object orientation (classes for inheritance, single inheritance rooted class hierarchy and virtual functions and message maps for polymorphism). Of course for people who want to use more complicated features of the C++ language they can, but they don't have to read a thick language spec to get up and running with Visual C++ and MFC.

Have you written a book, what were the motivations for doing so?

No – I've been asked and been tempted several time. I haven't been able to find the time.

Software industry in Iran is in its primary stages, what would be your wisest programming advice to the newcomers to this professions?

My suggestion would be to get up to speed on C++ and MFC. That's the most leveraged use of your time since C++ will be around for quite a while, and MFC is evolving all the time. Using C++ lets you build robust, fast and powerful applications, and building on top of MFC gives you a lot of functionality for free or for a very low cost.

With the heated war in OS and computer applications, do you think C, C++ are the final choices in computer languages and object oriented final programming paradigm?

C++ is certainly the primary object oriented language for type safe, class oriented, high performance development. For less formal programming, more dynamic languages, such as Visual Basic (and its object oriented extensions) are called for.

The object oriented paradigm will stay around for a while, not necessarily in a classic Smalltalk form – but in more robust and powerful systems like OLE.

Are these the final choices? – well probably not since everything evolves. They will be here for some time, but it is hard to predict what we will all be doing a decade or two from now.

What is the current project that you are working on?

We are just finishing up Visual C++ 2.0 (that includes MFC 3.0). This is a multi-platform product, supporting not only the 32-bit versions of Windows on Intel architectures, but also RISC platforms (like the MIPS) and we allow cross targeting to the Apple Macintosh.

Thanks again for your time.

Continued from page 7

Interview with P.J. Plauger

That will be your quickest entry into the more developed western markets. As consumers, resist the temptation to pirate large quantities of western software. You will be more quickly welcomed into the world marketplace if your domestic market is under control. (I know that neither of these goals are easy to achieve.)

10) With the heated war in OS and computer applications, do you think C, C++ are the final choices in computer languages and object oriented final programming paradigm.

The Standard C of today differs in important respects from the "K & R" C of ten years ago. The draft Standard C++ of today differs even more markedly from the "cfront" C++ of five years ago (it's a far more malleable language). We can expect both languages to continue to evolve to meet changing needs. And even so, another language can come along at any time and displace either or both of these market leaders. But all we can safely predict now is that rapid advances will continue in software, and slower but steady advances will continue in the most used programming languages.

11) What is the current project that you are working on?

I'm writing a book called The Standard C++ Library, as an obvious companion to my earlier book, The Standard C Library. I've just completed writing and debugging all the code. My friend Tom Plum, of Plum Hall Inc. (Kamuela, Hawaii), has already licensed this code to some companies for redistribution. My hope and plan is that we will do at least as well licensing the C++ library as we have the Standard C library. There are, after all, essentially *no* implementations of the C++ library that match the current draft C++ standard.

Thanks again for your time.

INTERVIEW

An Interview with

Scott Randell

Manager of the AFX group at Microsoft

This interview has been conducted through e-mail.

Some of our readers may not know you, could you introduce yourself- your academic and industrial background?

My name is Scott Randell, and I am a Software Design Engineer and the manager of the AFX group here at Microsoft. AFX stands for Application Framework X.

We are still trying to figure out what the 'X' stands for.

I have a BSc degree from the University of Waterloo in Canada. I don't really use my degree in Applied Physics all that much on a day-to-day basis.

How did you start your career as a programmer?

I began working at Microsoft right after graduating from Waterloo.

Before AFX, I worked in a variety of development teams here at Microsoft. I spent a lot of time working on internal tools and shared components that were used in Microsoft operating systems and applications such as Microsoft Excel and Word.

My group, the AFX group, began in early 1990. We began by spending over a year doing research, design, implementation and usability testing of a prototype system. We then threw away the prototype and started work on MFC in the summer of 1991. After several external design reviews we finally released MFC version 1 in the spring of 1992.

How are things in Microsoft-developments, researches?

Microsoft is a great place to work. There are many new developments and new technical areas Microsoft is tackling all the time. They span from home entertainment titles all the way to server technology paving the ' information highway'.

Of course my focus is on development tools. Even within the development tool area, there are many new innovative things going on all the time. In our area the success of MFC has certainly driven a lot of this change.

MFC version 2 (which released 10 months after MFC 1) and MFC version 2.5 (which was released 10 months after that) both greatly increased the number of people in the world who are using MFC.

We are now working on MFC version 3 to be released soon. we continue to constantly improve and evolve the Microsoft Foundation Classes to support new features of applications and operating systems.

What are the Microsoft's plans for dominating the compilers market?

Dominating isn't exactly the word I would choose. We are working hard to make the C++ language and the MFC class library the standard. These tools are what the majority of leading edge application developers, across the planet, use in their day to day work to help solve their problems.

This applied not only to the Windows platforms (Windows 3.x, Windows NT, future versions of Windows), but also to the Macintosh and through our work with third parties, Unix.

Tell us what makes Microsoft the biggest and most innovative software company in the world and what are the implications on you personally?

Well I would agree with you that Microsoft is the most innovative company out there (even though we aren't the biggest). Working at Microsoft is a dream come true for any hard-core programmer. The company is filled with very technical, highly charged people. The atmosphere here is great, and the products and technologies we are working on are really amazing.

Working for Microsoft also has the advantage that we do define and drive a lot of standards. Take MFC for example. The fact that we've come up with a technically strong, multi-platform, powerful programming API is one thing – but being part of Microsft makes us treat that like a systems API, support it well, drive it as an industry standard and even license it to our competitors in the industry – all so that it becomes the standard that developers choose.

Tell us your nicest story about being a computer guru?

Well my favorite story is about debugging the first version of Windows Excel (back in the days of real - mode Windows and Excel 2.0).

There was a very hard to reproduce bug in a specific part of GDI (long since fixed of course) that erroneously set and reset bits of a in-memory bitmap – but in the wrong part of memory.

Beginning with basics, we have a very readable article entitled the "A Layman's Introduction to Object-orientation", by Ramin Erfanian. A conceptual background for the topic is first presented. Next, prevalent problem-solving methods in computer science are pointed out to be inadequate. Then a solution in the guise of the concepts of object-orientation is gradually unfolded. Fundamental concepts of object-orientation are subsequently explained in a question and answer form, giving everyday examples that make the more complex concepts of object-orientation more understandable and accessible to the novice reader. Object Modeling is addressed and a summary on software quality factors is presented and a recapitulation of the benefits of object-orientation.

A translation of "Object Wars" (BYTE, May 1994) provides a concise overview of the current situation in terms of issues, conflicting opinions, cometing OO camps and problems encountered in trying to propagate the object-oriented paradigm by developing standard-based (e.g., CORBA) tools, environments, operating systems and languages. This article shows how the leading competitors in the OO software industry are trying to rally their own concepts and possibly market their own efforts. It is a good example of how aspects of computer science can be transormed into conflicting fiscal aims, when we endeavor to bring it down from the ivory towers of the intellectual pursuits and academic circles, into the trials and tribulations of teams struggling to develop a product that can be marketed in time to make profit and create the basis for a long-term standard in the industry.

In "A Model for Object-oriented Software Development", Ali Arsanjani presents a new object-oriented model, beginning with highlighting the importance of creating a fundamentally object-oriented software process model in order to exploit the capabilities provided by the paradigm in different methodologies. A brief review of fundamental software engineering principles is made, followed by the fundamental concepts of OO and a categorization of prevalent methodologies While using traditional life-cycle models such as the Waterfall or Evolutionary models, will not only hinder the more complete utilization of the power and benefits that OO truly offers, they usually will result in an increase in entropy in development activities which causes the fallbacks, problems and inadequacies experienced in many of OO-based projects in the world today. The author points out that there currently appears to be no model in existence that has been created from fundamentally object-oriented principles, and goes on to propose such a model of "Creative Evolution", as a solution, where *all* aspects of the development process, even management perspectives and development interactions are intrinsically object-oriented. Finally, the pros and cons of using OO are presented.

A concise history of an OOPL (Object-oriented Programming Language) is given in "A Brief History of Smalltalk", compiled by Ramin Erfanian.

A synthesis of some object-oriented methodologies is proposed to constitute "A Proposed Methodology for Object-oriented Construction" by Mohsen Sharifi. In this article, it is pointed out that the object-oriented paradigm has more promise in reconciling the activites of the software construction process. It is pointed out that most methodologies address or are concentrated on only one of the development activities, while an effort has been made in this article to look upon the development process in its entirety. Many notations are utilized together, in order to demonstrate a simulation of an air traffic control system as an example of how to apply the methodology and to simplify the complexities inherent in the example.

Ramin Erfanian records his experiences in a panel discussion with some of the finest minds in computer science. Here, there is a clash of older ideas in software engineering with newer ones.

"Inheritance with a Sample Application in C++" by Mahmoud Reza Zare introduces the topic of inheritance as applied to a segment of a hypothetical CAD application. The examples provide a step-by-step unfoldment of the practical utilization of inheritance in C++.

In this issue, we have email interviews with Grady Booch, one of the pioneers in Object-Orientation and Chief Scientist at Rational Software Corporation; P.J. Plauger, the senior editor of the C Users Journal; Scott Randell, Manager of the AFX group at Microsoft.

A Glossary of Farsi/English OO terminology has been compiled by Mohsen Sharifi. It is a commendable effort that can serve as the basis for further pursuit of the issues of standardization and localization of OO concepts in Iran.

In "The Object-orientation Practitioner's Group Report", Ali Arsanjani announces the creation of a new subgroup under the Informatics Society, which is founded with the intention of promoting the object paradigm; to the elevation of object-orientation awareness and correct practices, in an effort to make the technology more accessible to software developers. The aim is to propagate a sound software engineering object-orientation background, in order for users, managers and developers to be better informed of the vast potentialities of OO, as well as the pitfalls and tradeoffs incurred in utilizing object-oriented methods (models, methodologies, tools, and languages).

I hope you will enjoy this issue.

Ali Arsanjani

EDITORIAL

Object-Orientation:

A Software World-view

Whatever has been attributed to it, object-orientation in an inseparable trend of our times and it is well to realize that it is, basically, *a software world-view*. It is a way the developer looks at his/her system under consideration, forms abstractions, finds a model of the real-world problem in a conceptual "problem space", and then creates a series of solutions that are systematically eliminated and refined in "solution-space". The correspondence between the problem domain and solution elements of the solution domain is a convenient mental construct which aids in decomposing complexity. But, complexity management is not merely realized in problem decomposition; rather, it is **how** a problem is decomposed, and the **representation** of the decomposition that makes it amenable to a solution that may be categorized as simpler or more complex in the spectrum of relative complexity engulfing all conceptual realms in computer science.

We solve real-world problems in computer science and engineering by creating a model of the problem in what is called problem space. We then use the tools of our trade (mental constructs, concepts, notions and techniques) to create a model of the solution, in what is called "solution space"; which in fact is aimed to have as much a coorespondence with the problem space as is possible in order for the solution to be useful, pertinent and intelligible.

Object-orientation separates itself from other paradigms, in the seamless way it glides from Object-oriented Analysis to Design and then Implementation and Testing. The phase transition from one step to another does not require an alteration of conceptual tools; the same conceptual contructs that have been created in a previous step will only be further refined and elaborated in the new phase. The gracefulness by which this transition is accomplished makes the mapping of the problem space into the solution space and back onto the problem space (real world phenomena) one with a very narrow "conceptual gap".

The iterative and incremental nature of the development process has not been well-defined, though it has been emphatically acknowledged. Insight into this nature will lead to rapid advances in overall software quality and the realization of the goals of software engineering to a much larger extent. On the other side, the actual users or customers of the systems will be more satisfied in that the solutions coded for them in the embodiment of their system will be supportive to their business or scientific processes and will aid in making software an asset to mankind, rather than, in some cases, a liability.

The object-oriented paradigm represents an evolutionary step in that direction. There are, however, further steps to be taken in order to arrive at a wider integration of reusable components in heterogeneous platforms with rapid and intelligent object transparency. The next step in the evolution of software engineering concepts, will achieve that aim. And one pillar of that new edifice will definitely be the research and products based on the Object Paradigm, based on Object-oriented Technologies.

* * *

Editor's Note

There has been a profuse growth of interest in object-orientation (OO) during the past decade, due to the new and vast array of opportunities and advantages that this software engineering technology promises to offer. New products and standards, languages and software platforms on the one hand, and new research directions producing a prolific number of articles and books on the other hand have led the software communities throughout the world to turn their heads and behold that a concept that was thought to be in its teens, is indeed becoming mature. The entire concept of viewing the software world as the intelligent interaction of objects that simulate real-world phenomena had led to a new software engineering world-view; the **Object-oriented Paradigm**. This special issue of *Computer Report* of the Informatics Society of Iran is devoted to issues and ideas relating to this paradigm.

There are a wide variety of articles in this issue; covering quite a varied spectrum of interests and topics. I will try to give a flavor of each one in the following paragraphs. The articles fall into one of the following categories listed below:

- Email Interviews with three software gurus (Grady Booch, P.J. Plauger, Scott Randall)
- A Layman's Introduction to OO
- An Object-Oriented Software Model
- OO Methodologies
- OO Trends and Issues
- A Proposed OO Farsi/English Glossary
- OO News in Iran: the Object-orientation Practitioner's Group (OOPG)
- OO Languages
- Inheritance in C++

GOZARESH-E-COMPUTER
(*Computer Report*)
Vol. 16, No. 126
Jul-Aug 1994
Publisher
Informatics Society of Iran (ISI)
Managing Editor
Ebrahim N. Mashayekh

Circulation: CR is published bimonthly by ISI. Please address your subscription requests to: Anoosh Hosseini, P.O.Box 61622, Sunnyvale, CA 94088 USA. anoosh@sgi.com.
Annual subscription is included in membership fee. Non-member price: US$ 20 per year (6 issues).
CR features original and translated articles, news and reviews on all aspects of computing in Iran and abroad.

Submissions: Submit your articles to: The Editor, *Computer Report*, P.O.Box 1196, Tehran 14155, IRAN, isi@irearn.bitnet.
All submissions are subject to editing for style, clarity and space consideration.

Editorial: Unless otherwise stated, articles and reports reflect the author's opinion. Inclusion does not imply endorsement by ISI.

Mailing List Rental: ISI lists are available for computer-related products and services.

Copyright ©1994by Informatics Society of Iran. Copying without fee is permitted with credit to the source.

CR's camera-ready copies are produced using TEX-e-paarsi typesetting system.

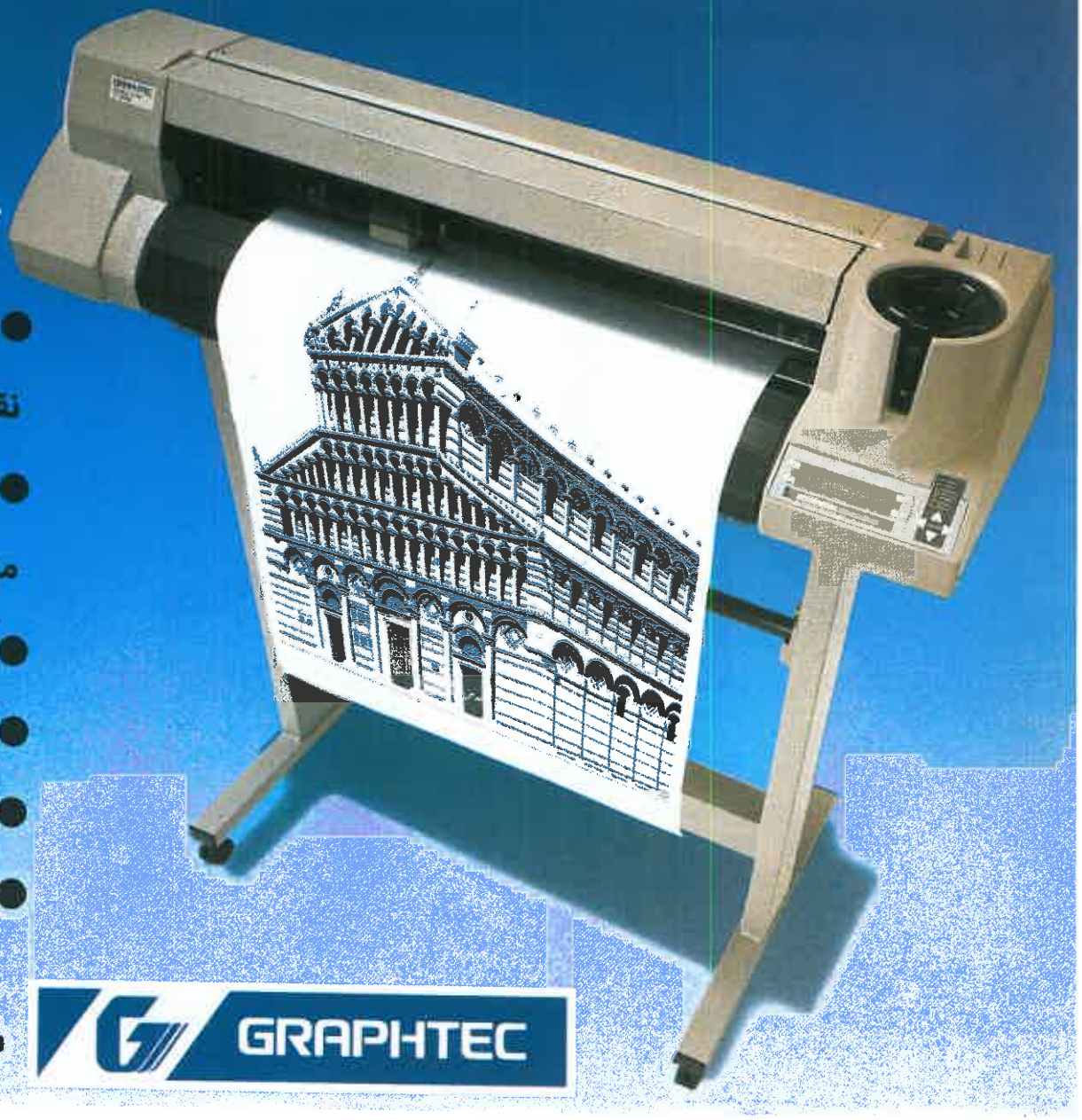
Contents of Persian Section

Informatics Society of Iran

Board of Directors
President: Ebrahim N. Mashayekh
Vice-President: Ali Parsa
Secretary: Dr. Mehdi Beheshtian
Treasurer: Mohammad M. Abdollahi
Committees
Publishing: Ebrahim N. Mashayekh
Science and Technology: Mohammad M. Abdollahi
Education: Ebrahim Abtahi
Public Relations: Dr. Mohammad Sanati
Membership: Mohammad-Hassan Mehvari
(All activities done by the Board members are voluntarily)

از پلاتر
چه انتظاری دارید؟

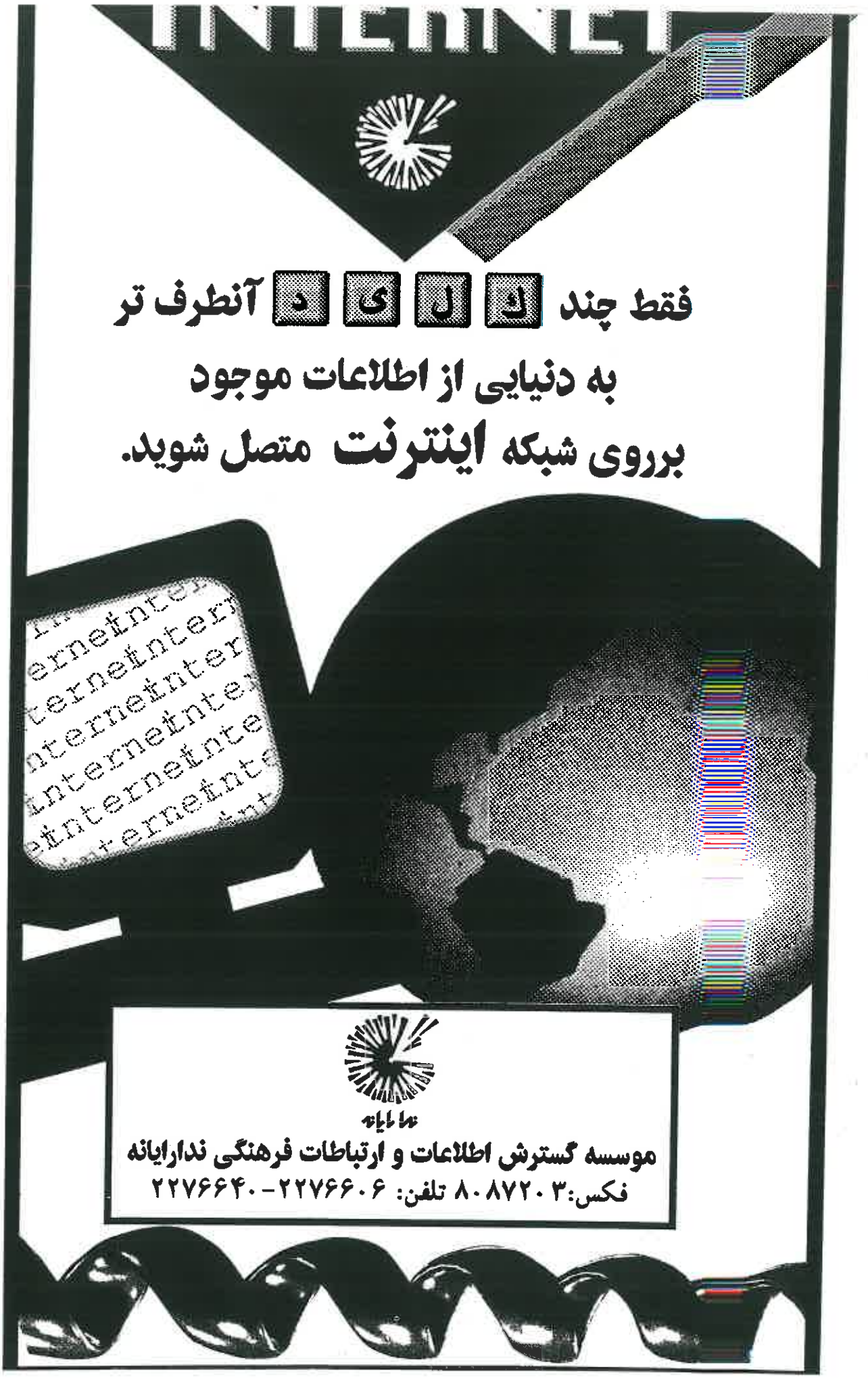
- قابلیت استفاده از کاغذ رول برای نقشه‌های طولانی
- مدیریت اعلام اتمام جوهر قلمها همراه با مسافت شمار
- آشنایی با پنج زبان زنده دنیا
- قابلیت رسم با مداد، ماژیک، خودکار، راپید
- تنظیم اتوماتیک سرعت و فشار قلمها
- و...



تلفن: ۸۹۶۶۶۲

شرکت مهندسین مشاور کامپیوتر وار تباطات

GRAPHTEC


INTERNET
فقط چند کلیک آنطرف تر
به دنیایی از اطلاعات موجود
برروی شبکه اینترنت متصل شوید.
موسسه گسترش اطلاعات و ارتباطات فرهنگی ندارایانه
فکس:۸۰۸۷۲۰۳ تلفن: ۲۲۷۶۶۰۶-۲۲۷۶۶۴۰

راه حل نهایی کامپیوتر

ارتباطات

LAN - WAN ...

LONGSHINE

مرکز کامپیوتر سازگار

واردات و توزیع انواع لوازم‌جانبی کامپیوتر

□ آدرس: خیابان دکتر شریعتی ـ نرسیده به سه راه طالقانی ـ جنب بیمارستان پاسارگاد پلاک ۱۴۲
تهران ۱۵۶۱۹ ایران
□ تلفن: ۷۶۲۲۸۸ ـ ۷۶۸۵۸۴ ـ ۷۵۲۲۳۲۳
□ فاکس: ۷۶۸۵۸۴

رنگهای زیباوجادوئی

قدم بگذارید

جدیدترین تکنولوژی + بالاترین کیفیت + مناسبترین قیمت

فقط با

hp DESKJET PRINTERS

تمام رنگی
مدلهای متنوع

EVERTECH SA
p.o Box 88. Sharjah. United Arab Emiarates: Tel:(+9716)352444/524003 Fax:(+9716)523937
Tlx:68033 Burkat EM Headoffice:6 Rue charles-Bonnet.1206 Geneva.
Switzerland.Tel:(022)839 40 00. Fax:(022)839 40 40

- تنها مراکز مجاز فروش و خدمات پس‌ازفروش:
- پویا (تلفن ۹-۸۷۳۴۴۲۵) (فاکس ۸۷۳۴۴۳۰)
- تراویس (تلفن ۸۵۶۵۲۵ و ۸۵۶۵۲۴) (فاکس ۸۷۰۳۳۵)
- داده‌پردازی ایران (تلفن ۹-۸۹۳۲۵۱) (فاکس ۸۹۱۷۱۰)
- میکرونرم‌افزار (تلفن ۶-۶۴۲۷۱۹۵) (فاکس ۹۳۹۶۵۶)
- بهینه‌پردازی سیستم‌های مهندسی (تلفن ۸۵۲۰۹۲ و ۸۵۹۰۰۰) (فاکس ۸۴۶۴۷۰)

راه حل
نهایی
کامپیوتر

ارتباطات

LAN - WAN ...

LONGSHINE

مرکز کامپیوتر سازگار

واردات و توزیع انواع لوازم‌جانبی کامپیوتر

□ آدرس: خیابان دکتر شریعتی ـ نرسیده به سه راه
طالقانی ـ جنب بیمارستان پاسارگاد پلاک ۱۴۲
تهران ۱۵۶۱۹ ایران
□ تلفن: ۷۶۲۲۸۸ ـ ۷۶۸۵۸۴ ـ ۷۵۲۲۳۲۳
□ فاکس: ۷۶۸۵۸۴

رنگهای زیباوجادوئی

قدم بگذارید

جدیدترین تکنولوژی + بالاترین کیفیت + مناسبترین قیمت

فقط با

hp DESKJET PRINTERS

تمام رنگی
مدلهای متنوع

EVERTECH SA
p.o Box 88. Sharjah. United Arab Emirates: Tel:(+9716)352444/524003 Fax:(+9716)523937
Tlx:68033 Burkat EM Headoffice:6 Rue charles-Bonnet.1206 Geneva.
Switzerland.Tel:(022)839400, Fax:(022)839440

- تنها مراکز مجاز فروش و خدمات پس‌ازفروش:
- پویا (تلفن ۹-۸۷۳۴۴۲۵) (فاکس ۸۷۳۴۴۳۰)
- تراویس (تلفن ۸۵۶۵۲۵ و ۸۵۶۵۲۴) (فاکس ۸۷۰۳۳۵)
- داده‌پردازی ایران (تلفن ۹-۸۹۳۲۵۱) (فاکس ۸۹۱۷۱۰)
- میکرونرم‌افزار (تلفن ۶-۶۴۲۷۱۹۵) (فاکس ۹۳۹۶۵۶)
- بهینه پردازی سیستم‌های مهندسی (تلفن ۸۵۲۰۹۲ و ۸۵۹۰۰۰) (فاکس ۸۴۶۴۷۰)