# به‌نام آن‌که جان را فکرت آموخت

# مهندسی نرم‌افزار مدل‌رانده

# اصول و مفاهیم

**نویسندگان:**

**دکتر بهمن زمانی**، دانشیار دانشکده مهندسی کامپیوتر دانشگاه اصفهان

**دکتر لیلا صمیمی دهکردی**، استادیار دانشکده فنی و مهندسی دانشگاه شهرکرد

**بهار ۱۴۰۰**

# فهرست مطالب

## سپاسگزاری

افراد متعددی در به‌ثمر نشستن این فعالیت نقش داشته‌اند که بر خود لازم می‌دانیم از همه آن‌ها تشکر نماییم. ابتدا از همسر و فرزندان خود که با صبوری و مهربانی ما را در این مهم پشتیبانی نمودند تشکر می‌نماییم. سپس از همکارانی که در گام‌های اولیه تدوین این کتاب همکاری نمودند، به‌ویژه خانم دکتر شکوفه کلاهدوز رحیمی، آقای دکتر علیرضا روحی و آقای مهندس محمدرضا شعرباف، سپاسگزاریم. در نهایت از اعضای گروه پژوهشی مهندسی نرم‌افزار مدل‌رانده دانشگاه اصفهان که با ارائه بازخوردهای مفید در خصوص محتوای کتاب، باعث غنی‌تر شدن مطالب گردیدند، قدردانی می‌نماییم.

**دکتر بهمن زمانی – دکتر لیلا صمیمی دهکردی**

**بهار ۱۴۰۰**

## پیش‌گفتار

علم کامپیوتر، و به‌تبع آن مهندسی نرم‌افزار، همواره به‌دنبال راهی برای بالابردن سطح تجرید بوده است. به‌عنوان نمونه، دانشمندان کامپیوتر، زبان‌های برنامه‌سازی سطح بالا را ابداع نمودند تا برنامه‌نویسان را از درگیرشدن در جزئیات زبان‌های سطح پایین‌تر، مثل زبان ماشین و اسمبلی، رهایی دهند. این تلاش برای بالابردن سطح تجرید همچنان ادامه دارد و معرفی روش «مهندسی مدل‌ْرانده[1]» (توجه کنید که حرف ل ساکن است) نیز دقیقاً در ادامه همان مسیر است. یعنی در این روش سعی می‌شود تا توسعه‌دهنده‌ی سیستم کمتر به برنامه‌نویسی مشغول باشد و به‌جای آن به مدل‌سازی بپردازد، با این امید که کد، به‌طور خودکار تولید شود.

گرچه تاریخ خیلی دقیقی را نمی‌توان برای معرفی روش مهندسی مدل‌رانده ذکر کرد، ولی از زمانی که این اصطلاح در مقالات علمی پدیدار شد، حدود ۱۵ سال می‌گذرد[2]. قاعدتاً همچون هر فناوری دیگری، این فناوری نیز مراحل چرخه عمر فناوری (معرفی، رشد، بلوغ، زوال) را طی خواهد کرد. امروزه این فناوری در مرحله بلوغ خود به‌سر می‌برد و انتظارات از آن به حد معقولی رسیده است. باتوجه به این واقعیت، آشنایی با اصول، مفاهیم و تکنیک‌های مهندسی مدل‌رانده برای کسانی که خواهان آشنائی با فناوری‌های روز در صنعت نرم‌افزار هستند می‌تواند بسیار مفید واقع شود.

هدف از نگارش این کتاب، به‌عنوان اولین کتاب در حوزه مهندسی مدل‌رانده در ایران، آن است که مفاهیم این حوزه به‌زبان ساده برای جویندگان علم، اعم از دانشجویان و شاغلان در صنعت، تبیین گردد. نویسندگان امیدوارند که در نیل به این هدف موفق عمل کرده باشند و مطالب کتاب برای خوانندگان مفید واقع شود.

به بیان بسیار ساده هدف مهندسی مدل‌رانده کم کردن تلاش و بالابردن بهره‌وری است، و برای رسیدن به این هدف تمرکز را بر مدل‌سازی (به جای کدنویسی) قرار می‌دهد. مدل‌های ساخته شده، توسط برنامه‌های مبدل (یا تبدیل) به‌طور خودکار به کد تبدیل خواهند شد. لذا در مهندسی مدل‌رانده تمرکز از کدنویسی به مدل‌سازی منتقل شده است و قرار است مدل‌های با کیفیت ساخته شوند و کد به‌طور خودکار تولید شود. بدین‌ترتیب می‌توان گفت که این روش بر دو اصل کلیدی مهندسی نرم‌افزار مبتنی است: تجرید (انتزاع) و خودکارسازی. این‌که توسعه‌دهندگان نرم‌افزار به‌جای کدنویسی تمرکز خود را بر مدل‌سازی بگذارند، همان تجرید است؛ و این‌که کد به‌طور خودکار تولید شود همان خودکارسازی است. برای مدل‌سازی نیاز به زبان مدل‌سازی است و هر زبان مدل‌سازی، دارای ساختار نحوی، نمادگان و معناست. این سه جزء، و به‌ویژه ساختار نحوی، با مفهومی به‌نام فرامدل تبیین می‌گردند. هرچه زبان مدل‌سازی به

---

[1] در همین ابتدا ذکر این نکته را لازم می‌دانیم که ما دو اصطلاح «مهندسی مدل‌رانده» و «مهندسی نرم‌افزار مدل‌رانده» را معادل یکدیگر می‌دانیم و به‌جای یکدیگر نیز به‌کار می‌بریم.

[2] Douglas C. Schmidt, "Model-Driven Engineering," IEEE Computer, Feb. 2006, Vol. 39, No. 2, pp. 25-31.

دامنه‌ی موردنظر نزدیک‌تر باشد احتمال تولید مقدار بیشتری از کد نهایی، بالاتر می‌رود. به‌همین دلیل، زبان‌های مدل‌سازی خاص دامنه در حوزه‌ی مهندسی مدل‌رانده نقش به‌سزائی دارند. همان‌گونه که مشهود است، در قلبِ روش‌های مدل‌رانده، تبدیل‌ها (یا برنامه‌های مبدل) قرار دارند که مدل را به کد تبدیل می‌کنند. لذا این جمله مشهور است که «تبدیل روح و قلب روش‌های مدل‌رانده است». در فصول مختلف این کتاب سعی گردیده است که مطالب فوق به شکل اصولی و علمی و البته به بیان ساده معرفی گردند.

این کتاب می‌تواند به‌عنوان کتاب درسی برای یک درس پیشرفته در حوزه مهندسی نرم‌افزار مورد استفاده دانشجویان کارشناسی و نیز تحصیلات تکمیلی در رشته‌های مهندسی کامپیوتر (گرایش‌های مهندسی نرم‌افزار و مهندسی فناوری اطلاعات) قرار گیرد. همچنین مهندسان نرم‌افزار شاغل در صنعت می‌توانند از این کتاب جهت آشنائی با مفاهیم اولیه مهندسی مدل‌رانده استفاده نمایند. گرچه به‌طور مشخص پیش‌نیازی برای فهم مطالب این کتاب ذکر نمی‌شود، ولی دانستن مطالبی در حد درس مهندسی نرم‌افزار دوره کارشناسی رشته مهندسی کامپیوتر به درک سریع‌تر مطالب این کتاب کمک خواهد نمود.

محتوای این کتاب به شش فصل به‌شرح زیر تقسیم می‌شود.

فصل اول به مقدمه می‌پردازد و با مروری بر مهندسی نرم‌افزار، وارد بحث مهندسی نرم‌افزار مدل‌رانده و اهداف آن می‌شود.

فصل دوم مفاهیم پایه‌ای مانند مدل و مدل‌سازی را تشریح می‌نماید. در این فصل اشاره‌ای به کاربرد مدل در علوم و صنایع مختلف شده و به‌طور خاص انواع نمودارهای زبان مدل‌سازی یکنواخت (یوام‌ال) معرفی می‌گردند.

فصل سوم مفاهیم و اصول مهندسی نرم‌افزار مدل‌رانده را معرفی می‌کند. مفاهیمی چون فرامدل و اجزای زبان‌های مدل‌سازی نیز در این فصل مورد بحث و بررسی قرار می‌گیرند.

فصل چهارم به معرفی فرامدل، فرامدل‌سازی و اصول ساخت یک زبان مدل‌سازی خاص دامنه اختصاص دارد. معماری مدل‌رانده نیز در این فصل معرفی شده است.

فصل پنجم بر روی موضوع تبدیل مدل و زبان‌های تبدیل مدل تمرکز دارد. با توجه به نقش محوری تبدیل‌ها در مهندسی مدل‌رانده، در این فصل سعی شده است پوشش مناسبی بر موضوع تبدیل مدل ارائه گردد.

فصل ششم به نوع دیگری از تبدیل، یعنی تبدیل مدل به کد می‌پردازد. در این راستا، دو رویکرد ترجمه و تفسیر، در زمینه تولید کد از روی مدل، مورد بحث قرار می‌گیرند. در انتها نیز موضوع مدل‌های قابل اجرا مطرح شده است.

در طراحی فصل‌های این کتاب سعی شده است از یک ساختار یکسان و مناسب برای محیط‌های آموزشی استفاده شود. بدین معنی که هر فصل با یک خلاصه کوتاه شروع می‌شود، سپس مطالب فصل به‌صورت متوالی ارائه می‌گردند. در انتهای هر فصل نیز بخش‌هایی برای تمرین در نظر گرفته شده است که می‌تواند جهت ارزیابی درک مطالب از آن‌ها استفاده شود.

مطالب تألیف شده در این کتاب حاصل سال‌ها تدریس و تحقیق در حوزه مدل‌رانده توسط نویسندگان است. از این رو، نویسندگان معتقدند که چینش و تنوع مطالب به‌سرعت می‌تواند یک فرد مبتدی را با این حوزه آشنا سازد. البته بدیهی است، مراجع متعددی جهت تهیه این مطالب مورد استفاده قرار گرفته‌اند که در پاورقی‌ها و نیز در انتهای هر فصل کتاب از آن‌ها یاد شده است. بدیهی است شکل‌ها و جدول‌هایی که منبعی برای آن‌ها ذکر نشده است حاصل کار نویسندگان کتاب است.

در خاتمه از خوانندگان گرامی تقاضا دارد پیشنهادها یا اشکالات مشاهده شده را از طریق یکی از رایانامه‌های زیر به اطلاع نویسندگان برسانند.

bahmanzamani@gmail.com
l.samimi@gmail.com

# فصل ۱

# مقدمه

امروزه نرم‌افزار بخشی جدانشدنی از زندگی انسان مدرن است. هدف مهندسی نرم‌افزار، چه به‌عنوان یک علم و چه به‌عنوان یک حرفه[1]، تولید نرم‌افزار خوب و باکیفیت است. در طول نیم قرنی که از پیدایش علم مهندسی نرم‌افزار می‌گذرد، جامعه مهندسان نرم‌افزار نگرش‌ها[2] و متدولوژی‌های[3] مختلفی را برای توسعه نرم‌افزار معرفی کرده‌اند. مهندسی نرم‌افزار مدل‌رانده[4]، یکی از جدیدترین نگرش‌ها در توسعه نرم‌افزار است که هدف نهایی آن تولید خودکار[5] کدِ نرم‌افزار از روی مدلِ طراحی است. این فصل به معرفی مفاهیم اصلی مهندسی نرم‌افزار به‌طور عام، و مهندسی نرم‌افزار مدل‌رانده به‌طور خاص می‌پردازد.

[1] Profession
[2] Paradigm
[3] Methodology
[4] Model-Driven Software Engineering (MDSE)
[5] Automatic

## اهداف فصل

- معرفی مهندسی نرم‌افزار و اهمیت آن
- آشنائی با متدولوژی‌های توسعه نرم‌افزار
- معرفی مهندسی نرم‌افزار مدل‌رانده
- آشنایی با اصطلاحات رایج در مهندسی مدل‌رانده
- معرفی اصول مهندسی مدل‌رانده

# ۱-۱ مهندسی نرم‌افزار

در دنیای مدرن امروز تصور زندگی بدون نرم‌افزار امکان‌پذیر نیست. در اکثر وسایلی که ما به‌طور مستقیم یا غیرمستقیم از آن‌ها استفاده می‌کنیم، این نرم‌افزار است که کنترل تجهیزات (یا همان سخت‌افزار) را به‌دست داشته و منجر به استفاده مفید از آن تجهیزات می‌شود. از خودرو و لوازم خانگی گرفته تا سیستم‌های مخابراتی و کارخانجات صنعتی، همه توسط سیستم‌های کامپیوتری کنترل می‌شوند و در قلب آن سیستمِ کامپیوتری نیز، نرم‌افزاری قرارگرفته است که عملیات را انجام می‌دهد. در برخی موارد مانند سیستم خلبان خودکار هواپیما[1] یا خودروهای بدون راننده، نرم‌افزار مسئولیت سنگینی را برعهده می‌گیرد به‌گونه‌ای که هرگونه اشتباه ممکن است منجر به حادثه ناگوار و جبران‌ناپذیری شود. اما برای ساخت یک نرم‌افزار، آیا پیروی از اصول و انتخاب یک متدولوژی مشخص ضروری است؟ مهندسی نرم‌افزار به این پرسش پاسخ می‌دهد که در ادامه به آن می‌پردازیم.

## ۱-۱-۱ مهندسی نرم‌افزار چیست؟

ساخت یک نرم‌افزار در واقع مانند حل یک مسئله[2] است. برای حل هر مسئله، یک فرآیند[3] عمومی را می‌توان ذکر کرد که چهار مرحله اصلی دارد: ۱) فهم مسئله، ۲) طراحی راه‌حل، ۳) اجرا یا ساخت راه‌حل و ٤) آزمایش یا آزمون[4] راه‌حل. برهمین اساس، برای ساخت یک نرم‌افزاز نیز می‌توان یک فرآیند ساده‌ی چهار مرحله‌ای پیشنهاد داد که با استفاده از واژگان مختص توسعه نرم‌افزار بیان شده باشد. این چهار مرحله عبارتند از: ۱) تبیین نیازمندی‌ها[5]، ۲) طراحی نرم‌افزار، ۳) ساخت نرم‌افزار و ٤) آزمون نرم‌افزار.

---

[1] Auto-Pilot
[2] Problem Solving
[3] Process
[4] Test
[5] Requirements Specification

به‌طور خلاصه، این مراحل را می‌توان بدین شکل تعریف نمود. در مرحله تبیین نیازمندی‌ها، نیازمندی‌ها یا قابلیت‌هایی که نرم‌افزار باید داشته باشد مشخص می‌گردند. در مرحله طراحی نرم‌افزار، محصولِ[1]نرم‌افزاریِ موردنظر که بتواند نیازمندی‌های فوق را برآورده کند طراحی می‌شود. در مرحله ساختِ نرم‌افزار نیز محصول نرم‌افزاری مورد نظر پیاده‌سازی می‌شود. نهایتاً، در مرحله آزمونِ نرم‌افزار نیز سعی می‌شود درستی عملکرد نرم‌افزار مورد آزمایش قرار گیرد.

حال که با مراحل ساخت نرم‌افزار آشنا شدیم، جادارد به تفاوتی که ساخت نرم‌افزار با حل یک مسئله‌ی مشخص، برای مثال یک مسئله ریاضی، دارد اشاره کنیم. درواقع، ساخت نرم‌افزار جزو مسائل بدرفتار[2] است، در حالی‌که حل یک مسئله ریاضی جزو مسائل خوش‌رفتار[3] است. از جمله تفاوت‌های مهم میان مسائل بدرفتار و مسائل خوش‌رفتار می‌توان به این موارد اشاره نمود [۱].

- مسئله بدرفتار را نمی‌توان به‌طور کامل تبیین نمود؛ بیان کامل مسئله، عملاً مستلزم درگیرشدن در حل مسئله است. ولی در مسائل خوش‌رفتار بیان مسئله به‌طور دقیق و کامل قابل انجام است و طراحی راه‌حل را می‌توان به‌خوبی از بیان مسئله جدا نمود. به‌عنوان مثال، مسئله‌ی حل معادله درجه دو در ریاضیات را در نظر بگیرید. به‌سادگی می‌توان صورت مسئله را تعریف کرد: «با داشتن ضرایب $a$ $(a\neq0)$ و $b$ و $c$ در معادله‌ی $ax^2+bx+c=0$ مقدار مجهول $x$ را پیدا کنید». در مقابل، یک مسئله نرم‌افزاری، مثلا طراحی یک سیستم خرید آنلاین را در نظر بگیرید. هیچگاه نمی‌توان به‌طور دقیق بیان کرد که مشتری چه انتظاری از سیستم دارد. عملاً اگر بخواهیم به‌طور دقیق بیان کنیم باید یک نمونه از سیستم ساخته شده را نشان دهیم.
- حل یک مسئله بدرفتار را نمی‌توان کامل آزمود و گفت راه‌حل درست یا غلط است؛ فقط می‌توان گفت یک راه‌حل بهتر یا بدتر از دیگری است. ولی در مسائل خوش‌رفتار به‌خوبی می‌توان راجع به درست یا غلط بودن یک راه‌حل قضاوت نمود. باز مسئله حل معادله درجه دو را در نظر بگیرید. وقتی حل شد، جواب‌ها را در معادله قرار می‌دهیم و به‌طور قطع می‌توانیم بگوییم جواب‌ها درست هستند یا خیر. در مقابل یک سیستم ساخته شده‌ی فروشگاه آنلاین ممکن است امروز درست کار کند، ولی فردا خطایی در آن کشف شود. در واقع هیچگاه نمی‌توان ثابت کرد که یک نرم‌افزار عاری از خطاست.
- در مسائل بدرفتار نمی‌توان به روش «سعی و خطا[4]» عمل نمود، چون گاهی خطا موجب فاجعه می‌شود. در حالی‌که در مسائل خوش‌رفتار به‌سادگی می‌توان اگر راه‌حل غلط بود، دوباره سعی نمود و راه‌حل دیگری پیدا کرد. اگر به مسئله‌ی حل معادله درجه دو برگردیم، در صورتی که

[1] Product
[2] Wicked Problems
[3] Tame Problems
[4] Trial and Error

در راه حل خود اشتباه کرده باشیم و جواب‌ها در معادله صدق نکنند، به‌سادگی برمی‌گردیم و دوباره حل می‌کنیم. ولی در فروشگاه آنلاین اگر وقتی نرم‌افزار را ساختیم، مشتری از آن راضی نبود، منجر به از دست رفتن زمان و هزینه شده است. حتی ممکن است مشتری کسب‌وکار خود را به‌دلیل بدعمل کردن نرم‌افزار از دست بدهد.

- در مسائل خوش‌رفتار، راه‌حل را می‌توان برای مسائل مشابه به‌کار برد، در حالی‌که هر مسئله بدرفتار در واقع یک مسئله یکتاست و راه‌حل خودش را دارد. در مثال حل معادله درجه دوم، همان‌گونه که در کتاب‌های ریاضی آموزش داده می‌شود، حل معادله درجه دوم یک روش مشخص دارد که اگر از آن تبعیت کنیم به حل خواهیم رسید. ولی هرگز نمی‌توان دو فروشگاه آنلاین را صددرصد مشابه هم دانست و یک روش کاملاً یکسان را برای تولید هردو اجرا نمود. حتی درحد بسیار کمی ممکن است دو کسب و کار با هم متفاوت باشند.

به‌منظور مرور مطالب، تفاوت‌های مسائل خوش‌رفتار و بدرفتار در جدول ۱-۱ خلاصه شده است.

جدول ۱-۱- مقایسه ویژگی‌های مسائل خوش‌رفتار و بدرفتار (برگرفته از [۱])

| مسائل خوش‌رفتار | مسائل بدرفتار |
|---|---|
| بیان مسئله به‌طور دقیق و کامل قابل انجام است. | مسئله ساختار مشخصی ندارد و به‌طور دقیق قابل تبیین نیست. |
| تبیین مسئله و راه‌حل را می‌توان از هم جدا نمود. | تبیین مسئله همان حل مسئله است و برعکس. |
| جواب مسئله را می‌توان از نظر درستی یا نادرستی آزمود. | جواب یک مسئله بدرفتار را نمی‌توان کامل آزمود. |
| اگر مسئله حل نشد، دوباره سعی می‌کنیم. | حل‌کننده مسئله حق ندارد اشتباه کند، چون ممکن است منجر به فاجعه شود. |
| راه‌حل را می‌توان برای مسائل مشابه به‌کار برد. | هر مسئله‌ی بدرفتار یک مسئله یکتاست. |

با توجه به موارد گفته شده، به‌خصوص وقتی با نرم‌افزارهای بزرگ و پیچیده سروکار داریم، ناچاریم از یک روش علمی و مهندسی برای تولید نرم‌افزار استفاده کنیم. اینجاست که علم مهندسی نرم‌افزار معرفی می‌شود.

یکی از جاهایی که مهندسی نرم‌افزار وارد عمل می‌شود و استفاده از آن لازم می‌نماید، پروژه‌های گروهی است. به این نکته دقت کنید که ساخت یک نرم‌افزار ساده و کوچک ممکن است از عهده‌ی یک فرد به‌تنهایی برآید، ولی وقتی نرم‌افزار بزرگ باشد عملاً این‌کار غیرممکن است. تجربه نشان داده که یک برنامه‌نویس به‌طور متوسط می‌تواند در هر روز حداکثر ۱۰۰ (یک‌صد) خط کد بنویسد. لذا طبیعی است که اگر این فرد بخواهد به‌تنهایی یک نرم‌افزار متوسط که شامل ۱۰۰۰۰۰۰ (یک میلیون) خط کد باشد را بسازد بایستی ۱۰۰۰۰ (ده هزار) روز کار کند. تصور کنید که فرد در سال ۲۰۰ روز کار کند، پس برای

ساخت یک نرم‌افزار متوسط باید ۵۰ سال کار کند! طبیعی است که این امر پذیرفتنی نیست. یک راه‌حل بدیهی این است که کار به صورت گروهی انجام شود. برای مثال، به جای این‌که یک نفر ۵۰ سال کار کند، یک تیم ۵۰ نفره به‌مدت یک‌سال کار کنند. همین‌جاست که استفاده از روش‌ها و اصول مهندسی نرم‌افزار لازم می‌شود.

حال که به اهمیت و لزوم مهندسی نرم‌افزار پی بردیم، بد نیست تعریفی از مهندسی نرم‌افزار ارائه دهیم. مؤسسه مهندسان برق و الکترونیک[1] مهندسی نرم‌افزار را به‌صورت زیر تعریف می‌کند [۲]:

مهندسی نرم‌افزار دو چیز است: (۱) اِعمال یک رویکرد سامان‌مند[2]، منظم[3] و کمّی برای تولید، عملیاتی‌کردن[4] و نگهداری[5] نرم‌افزار؛ یعنی، اِعمال مهندسی در نرم‌افزار، (۲) مطالعه‌ی رویکردهای[6] انجام موارد مطرح شده در (۱).

تعریف فوق می‌گوید که مهندسی نرم‌افزار هم یک حرفه است که یک مهندس برای خود انتخاب می‌کند (بخش اول تعریف)، هم یک علم است که در دانشگاه تدریس می‌شود (بخش دوم تعریف). در بخش اول صحبت از یک روش یا متدولوژی می‌شود که باید منظم و قاعده‌مند باشد، یعنی بتوان از این متدولوژی استفاده کرده و آن‌را برای ساخت محصولات نرم‌افزاری متفاوت به‌کار برد. در عین حال این روش باید منجر به نرم‌افزاری بشود که عملیاتی است و کار می‌کند. همچنین این روش برای نگهداشت نرم‌افزار نیز تدارک اندیشیده است. نهایتاً، این روش کمّی است، یعنی بتوان معیارهای مختلفی را در حین انجام کار توسعه نرم‌افزار اندازه‌گیری نمود. در بخش دوم تعریف، صحبت از یک رشته‌ی تحصیلی است که در آن نظریه‌ها و مباحث نظری مرتبط با مهندسی نرم‌افزار تدوین گشته، آموزش داده شده، یا بر روی آن‌ها پژوهش می‌شود.

همان‌گونه که در شکل ۱-۱ نشان داده شده است، اگر از منظر کلان به فعالیت‌های مهندسی نرم‌افزار نگاه کنیم، این فعالیت‌ها در سه شاخه قرار می‌گیرند:

۱) فعالیت‌های توسعه نرم‌افزار،

۲) فعالیت‌های مدیریت کیفیت نرم‌افزار، و

۳) فعالیت‌های مدیریت پروژه.

آن‌چه که قبلاً تحت عنوان فعالیت‌های تبیین نیازمندی‌ها، طراحی نرم‌افزار، ساخت نرم‌افزار و آزمون نرم‌افزار مطرح گردید جزو فعالیت‌های شاخه اول، یعنی توسعه نرم‌افزار، می‌باشند. در شاخه دوم کارهایی قرار می‌گیرند که بر روی کیفیت انجام فعالیت‌ها و کیفیت محصول تمرکز دارند. این کارها تحت عنوان

---

[1] Institute of Electrical and Electonics Engineers (IEEE)

[2] Structured

[3] Disciplined

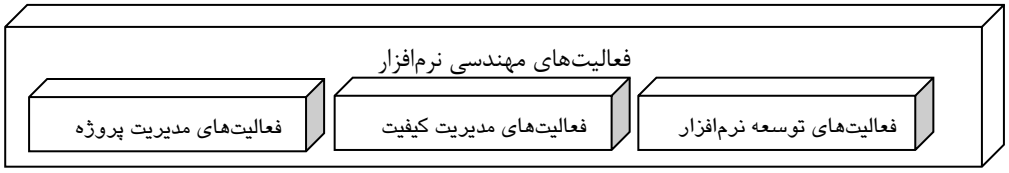
[4] Operation

[5] Maintenance

[6] Approach

تضمین کیفیت نرم‌افزار[1] نیز شناخته می‌شوند و به‌طور کلی به دو دسته‌ی وارسی[2] و اعتبارسنجی[3] تقسیم می‌شوند. هدف از وارسی این است که مطمئن شویم محصول را به‌درستی ساخته‌ایم و هدف از اعتبارسنجی این است که مطمئن شویم محصولِ درستی ساخته‌ایم. به‌عبارت دیگر، وارسی با فرآیند تولیدِ محصول، و اعتبارسنجی با خودِ محصول سروکار دارد.

باتوجه به اهمیت بحث متدولوژی در توسعه‌ی نرم‌افزار، در بخش بعدی مرور مختصری بر متدولوژی‌های توسعه‌ی نرم‌افزار خواهیم داشت. مجدداً یادآور می‌شود که ساخت نرم‌افزارهای متوسط و بزرگ، و حتی نرم‌افزارهای کوچک اما پیچیده، حتماً نیاز به متدولوژی دارد و اگر به روش سَرسَری[4] ساخته شود، احتمال عدم موفقیت پروژه یا پایین بودن کیفیتِ محصول، بسیار زیاد است.




شکل ۱-۱- سه شاخه از فعالیت‌های مهندسی نرم‌افزار (برگرفته از [۱])

## ۱-۱-۲ متدولوژی‌ها در مهندسی نرم‌افزار

برای واژه‌ی متدولوژی، معادل فارسی «روش‌شناسی» پیشنهاد شده است، ولی ما از همان کلمه اصلی استفاده می‌کنیم. همان‌گونه که ترجمه فارسی اشاره دارد، منظور از متدولوژی، روشِ انجام یک کار است. متدولوژیِ نرم‌افزار در واقع چگونگی انجام هرکدام از فعالیت‌هایی که در بخش قبل معرفی گردید را مشخص می‌کند. یادآور می‌شود که فعالیت‌های مذکور را به سه دسته تقسیم نمودیم، ولی تمرکز بیشتر در بحث متدولوژی‌های نرم‌افزاری بر روی «فعالیت‌های شاخه‌ی توسعه نرم‌افزار» است، گرچه نباید از اهمیت فعالیت‌های دوشاخه دیگر نیز غافل شد. برخی صاحب‌نظران، مانند آقای پرسمن[5] (نویسنده کتاب معروف مهندسی نرم‌افزار)، فعالیت‌های شاخه اول را فعالیت‌های اصلی و فعالیت‌های دو شاخه دیگر را فعالیت‌های فرعی یا چتری[6] نام نهاده‌اند [۲]. اغلبِ کتاب‌های درسی مهندسی نرم‌افزار (مانند کتاب معروف آقای سامرویل [۳]) نیز بیشتر بر روی فعالیت‌های اصلی تمرکز داشته و بحث راجع به مواردی چون مدیریت کیفیت و مدیریت پروژه را به کتاب‌ها و دروس دیگر واگذار می‌نمایند.

---

[1] Software Quality Assurance (SQA)
[2] Verification
[3] Validation
[4] Ad hoc
[5] Roger S. Pressman
[6] Secondary (Umbrella) Activities

تاکنون متدولوژی‌های متعددی برای توسعه‌ی نرم‌افزار معرفی شده است. متدولوژی‌های نرم‌افزار را می‌توان از جنبه‌های مختلفی تقسیم‌بندی نمود. یک تقسیم‌بندی معروف، تقسیم متدولوژی‌ها به دو دسته‌ی سنگین‌وزن[1] و سبک‌وزن[2] است. در این تقسیم‌بندی، منظور از وزنِ متدولوژی، میزان قوانین و مقررات و مستندات متدولوژی است. به‌عبارتی، هر چه یک متدولوژی قوانین و مقرارت بیشتر و دست و پا گیرتری داشته باشد و مهندس نرم‌افزار را مجبور به تولید مستندات بیشتری نماید، سنگین‌وزن‌تر است. برعکس، متدولوژیی که فقط در صورت لزوم و مفید بودن یک مستند، آن را تولید کند و قوانین و مقررات زیادی نداشته باشد، سبک‌وزن است. به متدولوژی‌های سبک‌وزن، به‌اصطلاح «چابک[3]» نیز گفته می‌شود. ذکر این نکته مهم است که معرفی متدولوژی‌های چابک، که اوج معروفیت آن‌ها به اواخر قرن بیستم برمی‌گردد، عملاً واکنشی بود به پروژه‌های شکست‌خورده‌ی نرم‌افزاری، که بسیاری از متخصصان دلیل شکست این پروژه‌ها را سنگین بودن متدولوژی استفاده شده در تولید نرم‌افزار می‌دانستند.

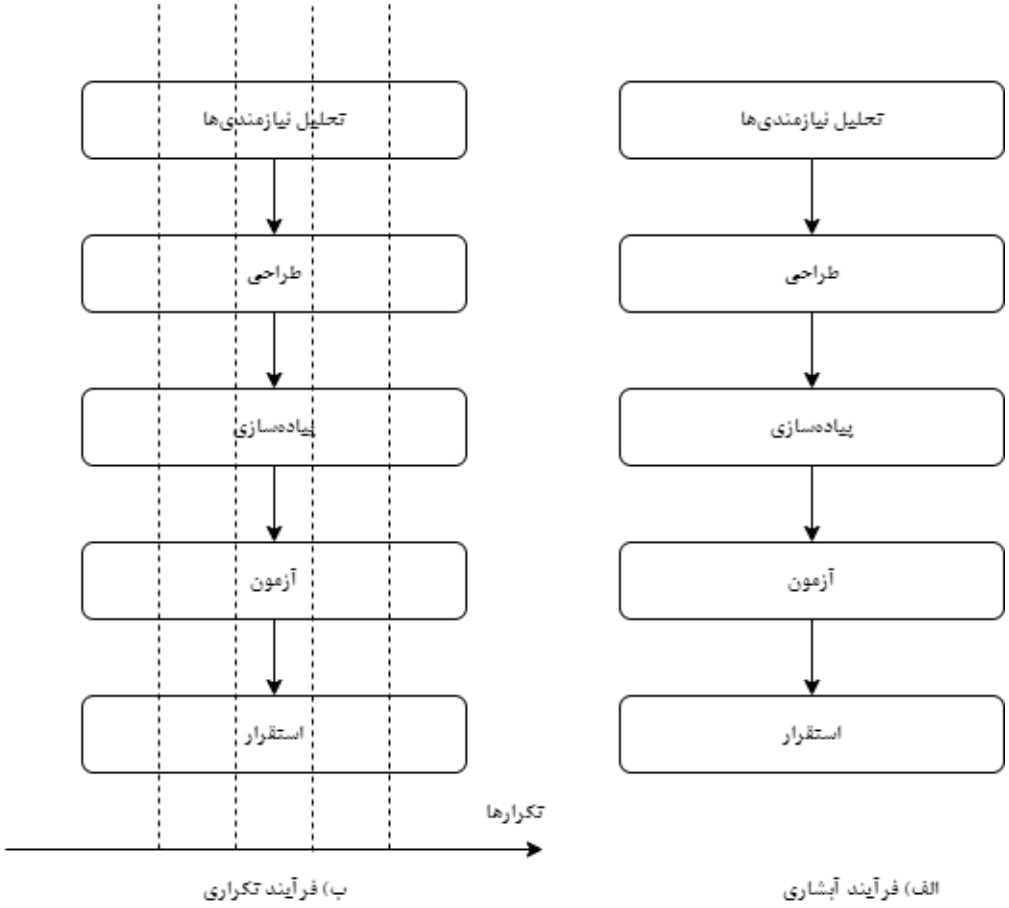
یکی دیگر از جنبه‌های تقسیم‌بندی متدولوژی‌ها، «ترتیب و جریان[4]» انجام فعالیت‌ها است. از این منظر، متدولوژی‌ها به دو دسته‌ی ترتیبی[5] و تکراری[6] تقسیم می‌شوند. در روش‌های ترتیبی فعالیت‌ها به‌ترتیب یکی پس از دیگری انجام می‌شوند، در حالی‌که در روش‌های تکراری هر فعالیت در چند نوبت (تکرار) انجام می‌شود. از جمله متدولوژی‌هایی که مبتنی بر روش ترتیبی هستند می‌توان به مدل آبشاری[7]، روش ساختمند تحلیل و طراحی سیستم‌ها[8] و مدل وی[9] اشاره نمود. در عوض، متدولوژی‌هایی مانند فرآیند یکنواخت[10] و یا متدولوژی‌های چابک مانند اسکرام (Scrum) و اکس‌پی (XP) مبتنی بر روش تکراری هستند.

به‌عنوان مثال، در یک متدولوژی که مبتنی بر مدل آبشاری باشد، تاکید بر این است که ابتدا نیازمندی‌ها به‌طور کامل و با جزئیات کافی شناسایی و تبیین شوند، سپس فعالیت‌های بعدی مانند طراحی نرم‌افزار شروع شود. در مقابل، در یک متدولوژی که مبتنی بر مدل تکراری است، دورها[11] یا تکرارهای مختلفی انجام می‌شود، و در هر دور، کمی از فعالیت تبیین نیازمندی‌ها انجام شده و بلافاصله به سراغ طراحی و پیاده‌سازی نرم‌افزار می‌رود و سپس در دورهای بعدی مقدار دیگری از نیازمندی‌ها تبیین و

---

[1] Heavyweight
[2] Lightweight
[3] Agile
[4] Flow
[5] Sequential
[6] Iterative
[7] Waterfall Model
[8] Structured System Analysis and Design Method (SSADM)
[9] V Model
[10] Unified Process
[11] Iteration

طراحی و پیاده‌سازی شده و به بخش‌های قبلی اضافه می‌گردد. بدین سبب، اصطلاح مدل افزایشی[1] نیز به این روش اطلاق می‌شود. شکل ۱-۲ یک دید کلی از فعالیت‌های اصلی توسعه نرم‌افزار را در دو مدل آبشاری و تکراری نشان می‌دهد.




شکل ۱-۲- نمایش کلی از فعالیت‌های توسعه نرم‌افزار در دو مدل آبشاری (راست) و تکراری (چپ) [۱]

جدای از تقسیم‌بندی‌های فوق (سنگین‌وزن/سبک‌وزن یا ترتیبی/تکراری) برای متدولوژی، روش‌های نوینی در توسعه نرم‌افزار نیز مطرح شده‌اند که از آن جمله می‌توان به مهندسی نرم‌افزار مدل‌رانده اشاره نمود. مهندسی نرم‌افزار مدل‌رانده موضوع اصلی این کتاب است و در بخش بعدی به‌اختصار معرفی خواهد شد.

[1] Incremental

## ۱-۲ مهندسی نرم‌افزار مدل‌رانده

«مهندسی نرم‌افزار مدل‌رانده» یا به‌اختصار «مهندسی مدل‌رانده»، روشی است در تولید نرم‌افزار که تمرکز اصلی آن بر مدل است نه کد؛ بدین معنی که ابتدا نرم‌افزار، مدل‌سازی می‌شود و سپس کد به‌طور خودکار از روی مدل تولید خواهد شد. به‌همین دلیل است که می‌گوییم «مدل‌ْرانده»، یعنی این مدل است که فرآیند توسعه را به‌پیش می‌راند. لازم به‌ذکر است که هدف نهایی روش‌های مدل‌رانده تولید صددرصد (۱۰۰٪) کد از روی مدل است. با این وجود، ما هر چه به این هدف نزدیک‌تر شویم می‌توانیم ادعا کنیم که مدل‌رانده کار کرده‌ایم گرچه ممکن است مقداری از کد به‌طور خودکار تولید شود و نیاز باشد تا مقداری کد نیز به‌صورت دستی اضافه یا ویرایش شود. به‌همین دلیل، در بسیاری از مواقع در روش‌های مدل‌رانده، از اصطلاح «نیمه‌خودکار[1]» نیز استفاده می‌شود.

### ۱-۲-۱ اصطلاحات مطرح در مهندسی نرم‌افزار مدل‌رانده

با توجه به این‌که در مبحث مدل‌رانده، اصطلاحات متعددی مطرح شده‌اند، جادارد که در همین ابتدا، برخی اصطلاحات رایج را معرفی نماییم. اولین اصطلاح «مهندسی مدل‌مبنا[2]» است که منظور از آن، روش‌هایی در توسعه نرم‌افزار است که از مدل بهره می‌برند ولی مدل نقش محوری نداشته و کد بر اساس مدل (اما به شکل مستقل) توسط برنامه‌ساز نوشته خواهد شد.

لازم به‌ذکر است که مدل و مدل‌سازی از دیرباز نقش مهمی در مهندسی نرم‌افزار داشته‌اند. به‌عنوان نمونه، بیشتر مواقع از مدل به‌عنوان نقشه‌ای[3] برای ساخت، یا به‌منظور مستندسازی و یا ارتباط[4] بهتر افراد با یکدیگر استفاده می‌شده است. برای مثال، نمودار مورد کاربرد[5] در زبان مدل‌سازی یکنواخت(یوام‌ال)[6] یک مدل تصویری است که به‌خوبی نشان می‌دهد «چه کسی با سیستم چه کاری می‌کند». شکل ۱-۳ یک مثال ساده از یک نمودار موردِ کاربرد را برای یک سیستم بانکی نشان می‌دهد که در آن دو کنشگر[7] و چهار مورد کاربرد نشان داده شده است. کنشگرِ مشتری بانک می‌تواند از حساب خود پول برداشت کند یا به حساب خود پول واریز نماید. در مقابل، کارمند بانک می‌تواند برای یک مشتری حساب ایجاد نماید یا حساب یک مشتری را مسدود نماید.

---

[1] Semi-automatic

[2] Model-Based Engineering (MBE)

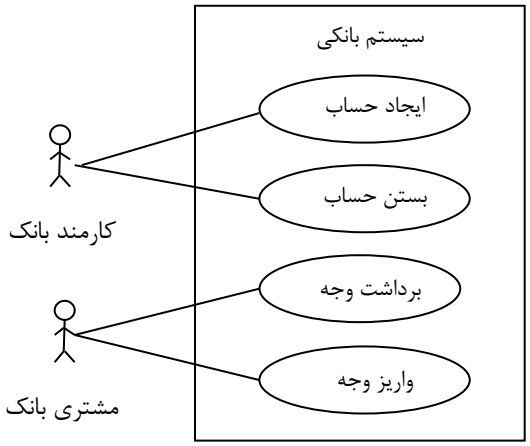
[3] Blueprint

[4] Communication

[5] Use Case Diagram

[6] Unified Modeling Language (UML)

[7] Actor




شکل ۱-۳- یک مدل یوام‌ال (نمودار مورد کاربرد) ساده برای یک سیستم بانکی

پس از «مهندسی مدل‌مبنا» به دومین اصطلاح، یعنی «مهندسی مدل‌رانده» می‌رسیم که در واقع نوع خاصی از «مهندسی مدل‌مبنا» است که در آن مدل، نقش اساسی و محوری دارد. در این روش، مدل نه‌تنها می‌تواند کاربردهایی چون مستندسازی یا نقشه‌ای برای ساخت داشته‌باشد، بلکه فراتر از آن، کد به‌طور خودکار از روی مدل تولید خواهد شد. همان‌گونه که در پیش‌گفتار کتاب نیز یادآور شدیم، در طول این کتاب، ما اصطلاح «مهندسی نرم‌افزار مدل‌رانده» را مترادف با همان مفهوم «مهندسی مدل‌رانده» در نظر گرفته‌ایم، چون به‌هر حال هدف در این روش‌ها، تولید نرم‌افزار است و منظور از مهندسی، همان مهندسی نرم‌افزار است.

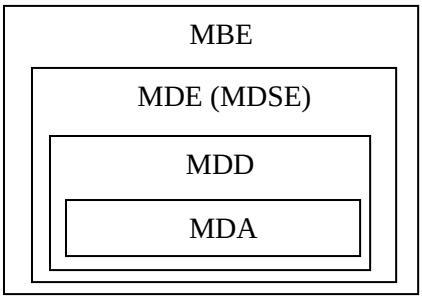
اصطلاح سوم، «توسعه مدل‌رانده[1]» است که در واقع نوع خاصی از «مهندسی مدل‌رانده» می‌باشد که تمرکز آن صرفاً بر «تولید و ساخت نرم‌افزار» استوار است، در حالی‌که مفهوم «مهندسی مدل‌رانده» باید همه فعالیت‌های چرخه حیات اعم از نیازمندی‌ها، طراحی، ساخت، و آزمون را دربرگیرد. پس به‌اختصار خواهیم گفت که «توسعه مدل‌رانده» زیرمجموعه‌ی «مهندسی مدل‌رانده» است. ضمناً ما اصطلاح «توسعه نرم‌افزار مدل‌رانده[2]» را نیز معادل «توسعه مدل‌رانده» درنظر می‌گیریم، با این استدلال که در «توسعه مدل‌رانده» هم هدف «توسعه‌ی نرم‌افزار» بوده است.

چهارمین اصطلاح، «معماری مدل‌رانده[3]» است که در دنیای مدل‌رانده مطرح شده است. با توجه به این‌که این موضوع (یعنی معماری مدل‌رانده) در فصل ٤ معرفی خواهد شد، در این‌جا به ذکر همین نکته

[1] Model Driven Development (MDD)
[2] Model Driven Software Development (MDSD)
[3] Model Driven Architecture (MDA)

بسنده می‌کنیم که «معماری مدل‌رانده» در واقع نگرش «گروه مدیریت شئ» یا به‌اختصار «اوام‌جی» [1] است به «توسعه مدل‌رانده»؛ بدین معنی که، اگر روش مدل‌رانده با استفاده از استانداردها و واژگانی [2] که این مؤسسه معرفی کرده‌است انجام شود، می‌توان گفت تولید نرم‌افزار طبق روش «معماری مدل‌رانده» انجام شده است. به‌طور خاص، در «معماری مدل‌رانده» تاکید می‌شود که برای این‌که از مدل به کد برسیم، لازم است ابتدا «مدل مستقل از سکو[3]» برای سیستم طراحی نموده، سپس آن‌را به «مدل خاص سکو[4]» تبدیل کرده و بعد آن‌را به کد تبدیل کنیم. از این رو است که دو اصطلاح PIM و PSM در روش «معماری مدل‌رانده» بسیار رایج هستند.

بر اساس مطالب گفته شده در خصوص اصطلاحات مطرح در روش‌های مدل‌رانده، شکل ۱-٤ ارتباط میان چهار مفهوم یا اصطلاح معرفی شده، یعنی «مهندسی مدل‌مبنا»، «مهندسی مدل‌رانده»، «توسعه مدل‌رانده» و «معماری مدل‌رانده» را نشان می‌دهد.



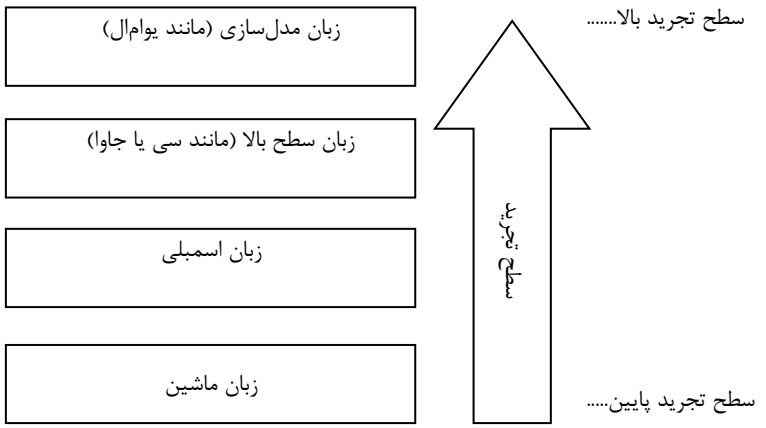

شکل ۱-٤- ارتباط میان اصطلاحات مطرح درحوزه مدل‌رانده (برگرفته از [٤])

### ۱-۲-۲ اصول مهندسی نرم‌افزار مدل‌رانده

مهندسی نرم‌افزار همواره سعی داشته‌است سطح تجرید (یا انتزاع)[5] را بالاتر ببرد تا کار برنامه‌سازان ساده‌تر شود. ابداع زبان‌های برنامه‌سازیِ سطح بالا نیز به‌همین دلیل بود. یعنی به‌جای برنامه‌نویسی با زبان‌های سطح پایینی چون زبان ماشین یا زبان اسمبلی، برنامه‌ساز می‌تواند با زبانی سطح بالاتر (مثلا سی یا جاوا) برنامه‌نویسی کند و سپس این برنامه توسط یک مترجم[6] به زبان ماشین ترجمه شود. در ادامه‌ی این مسیر، در مهندسی نرم‌افزار مدل‌رانده، قصد بر آن است که حتی برنامه‌ساز از برنامه‌نویسی

[1] Object Management Group (OMG)
[2] Terminology
[3] Platform Independent Model (PIM)
[4] Platform Specific Model (PSM)
[5] Abstraction
[6] Compiler

نیز رهایی یابد و به جای تفکر روی کدنویسی، به طراحی نرم‌افزار (که معمولا با مدل‌سازی همراه است) بیشتر بپردازد. لذا مدل‌سازی یک فعالیت اصلی در روش‌های مدل‌رانده محسوب می‌شود. بدیهی است که برای مدل‌سازی نیاز به یک زبان مدل‌سازی می‌باشد. نهایتاً وقتی طراحی (مدل) آماده شد، مدل به‌طور خودکار (و توسط تعدادی مبدل یا تبدیل[1]) به کد، تبدیل می‌شود. شکل ۱-۵ ارتباط زبان‌های برنامه‌سازی و زبان‌های مدل‌سازی را با سطوح تجرید نشان می‌دهد. همان‌گونه که در شکل مشخص است، سطح تجریدِ بالا وقتی حاصل می‌شود که مفاهیم برای انسان قابل فهم‌تر باشند و در سطح تجریدِ پایین‌تر، مفاهیم برای انسان سخت‌تر و برای ماشین قابل فهم‌تر می‌باشند.

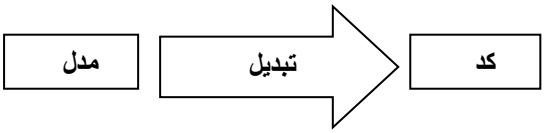



شکل ۱-۵- ارتباط میان زبان‌ها و سطوح تجرید

چنان‌چه بخواهیم مقایسه‌ای ساده و کلیشه‌ای میان مهندسی نرم‌افزار سنتی (منظور، آن روش تولید نرم‌افزار است که تمرکزش بر کدنویسی است) و مهندسی نرم‌افزار مدل‌رانده (روشی که تمرکزش بر مدل‌سازی است) داشته باشیم، می‌توانیم معادله‌ی معروف نیکلاس ویرث[2] که می‌گوید نرم‌افزار متشکل از الگوریتم و ساختمان‌داده است (الگوریتم + ساختمان‌داده = نرم‌افزار) را بدین شکل بیان کنیم که در روش‌های مدل‌رانده، نرم‌افزار متشکل از مدل و تبدیل است (مدل + تبدیل = نرم‌افزار).

شکل ۱-٦ در یک دید کلان، نگرش مدل رانده را در تولید نرم‌افزار به‌تصویر می‌کشد. تبدیل، خود برنامه‌ای است که به یک زبان تبدیل[3] نوشته شده است و مدل را به کد تبدیل می‌کند.

[1] Transformation
[2] Niklaus Wirth
[3] Transformation Language




شکل ۱-٦- نگرش کلان روش مدل‌رانده

وقتی صحبت از مدل به میان می‌آید، این سوال مطرح می‌شود که مدل چگونه ساخته می‌شود. همان‌گونه که برای نوشتن یک برنامه، به زبان برنامه‌نویسی نیاز است، برای ساختن یک مدل هم به یک زبان مدل‌سازی نیاز است. هر زبان مدل‌سازی به‌طور کلی شامل سه جزء است که عبارتند از: نحو انتزاعی[1]، نحو عینی[2] و معنا[3].

در نحو انتزاعی مفاهیم زبان و ارتباط میان آن مفاهیم نشان داده می‌شود. در زبان‌های طبیعی و نیز زبان‌های برنامه‌سازی نحو انتزاعی با گرامر زبان نشان داده می‌شود. در نحو عینی، نمادگان[4] استفاده شده برای نمایش مفاهیم و روابط معرفی می‌شود. در زبان‌های مدل‌سازی گرافیکی (مانند یوام‌ال) این نمادگان به‌صورت تصویری است، ولی در زبان‌های متنی (مانند جاوا)، بخش مهمی از این نمادگان در واقع همان واژگان کلیدی[5] هستند که برای زبان انتخاب شده‌اند. در معنا نیز معنی هر کدام از مفاهیم مشخص می‌شود و قیود احتمالی زبان تعریف می‌شوند. به‌عنوان یک مثال ساده، در زبان یوام‌ال، مفهوم کلاس یکی از مفاهیم اصلی است و کلاس‌ها می‌توانند با یکدیگر روابطی (مانند وراثت) داشته باشند. این جزئی از نحو انتزاعی زبان یوام‌ال است. نحو عینی انتخاب شده برای مفهوم کلاس در زبان یوام‌ال، یک مستطیل است که نام کلاس درون آن نوشته می‌شود. از لحاظ معنایی، یک کلاس بیان‌گر خانواده‌ای از اشیاء است.

## ۱-۳ خلاصه

در این فصل با مفاهیم و تعاریف اولیه‌ی مهندسی نرم‌افزار و مهندسی نرم‌افزار مدل‌رانده آشنا شدیم. مهندسی نرم‌افزار، هم یک حرفه و هم یک علم است و هدف آن تولید نرم‌افزار باکیفیت است. مهندسی نرم‌افزار مدل‌رانده به‌عنوان نگرشی جدید در توسعه نرم‌افزار به‌منظور بالابردن سطح تجرید، ابتدا مدلی از سیستم را طراحی می‌نماید و سپس کد را به‌طور خودکار از روی مدل تولید می‌نماید.

---

[1] Abstract Syntax
[2] Concrete Syntax
[3] Semantics
[4] Notation
[5] Keyword

## تمرین

۱- این‌که گفته می‌شود مهندسی نرم‌افزار هم یک علم است هم یک حرفه، منظور چیست؟

۲- فعالیت‌های اصلی که در هر متدولوژی توسعه نرم‌افزار بایستی انجام شود کدامند؟

۳- دو دسته‌بندی رایج برای متدولوژی‌های نرم‌افزاری را معرفی کنید.

۴- چند تعریف عمومی و یک تعریف رسمی از مهندسی نرم‌افزار ارائه نمایید.

۵- هدف نهایی مهندسی نرم‌افزار مدل‌رانده چیست؟

۶- با رسم یک شکل، ارتباط بین اصطلاحات رایج در روش‌های مدل‌رانده را نشان دهید.

۷- منظور از تبدیل در روش‌های مدل‌رانده چیست؟

۸- منظور از سطح تجرید چیست؟با مروری بر تاریخچه علم کامپیوتر مثال‌هایی از بالابردن سطح تجرید بزنید.

۹- سه جزء اصلی هر زبان مدل‌سازی کدامند؟

۱۰- چرا گفته می‌شود که تبدیل روح و قلب مهندسی نرم‌افزار مدل‌رانده است؟

۱۱- برج هانوی یک مسئله‌ی معروف است. ابتدا این مسئله را توصیف کرده و سپس راه‌حلی به‌صورت بازگشتی برای آن بیان نمایید. به‌نظر شما چرا مسئله برج هانوی یک مسئله خوش‌رفتار است؟

۱۲- به‌نظر شما مسئله مقابل بدرفتار است یا خوش‌رفتار؟ ویژگی‌های آن را بیان کنید. نشان دهید که برای هر عدد صحیح n رابطه زیر برقرار است. $1^3 + 2^3 + \cdots + n^3 = (1 + 2 + \cdots + n)^2$

۱۳- یکی از سیستم‌های فروشگاه برخط همانند Amazon یا Ebay را توصیف نمایید. چرا توسعه‌ی این سیستم‌ها یک مسئله بدرفتار است؟

۱۴- شرح مختصری از ماشین فروشنده (Vendor Machine) را ارائه نمایید.

## مراجع


1. David C Kung, *Object-Oriented Software Engineering: An Agile Unified Methodology*, McGraw-Hill Education, 2013.
2. Roger S. Pressman and Bruce R Maxim. *Software Engineering: A Practitioner's Approach*, 8th edition, McGraw-Hill Education, 2015.
3. Ian Sommerville. *Software Engineering*, 10th edition, Pearson, 2015.
4. Marco Brambilla, Jordi Cabot, and Manuel Wimmer, *Model-Driven Software Engineering in Practice*, 2nd edition, Synthesis Lectures on Software Engineering, 2017.