به نام خداوند مهربان

به نام خداوند بخشنده و بخشایشگر

نام دانشجو: ابوالفضل امیری (مقطع کارشناسی)

مهندسی نرم‌افزار ۱

نام استاد: دکتر احمد فراهی

دانشگاه پیام‌نور

دانشکده فنی مهندسی

گروه مهندسی کامپیوتر و فناوری اطلاعات

Top of Form

# اصول اساسی مهندسی نرم افزار



## مهندسان فقط سازنده نیستند

مهندسی نرم‌افزار تماماً یافتن و به‌کارگیری بهترین روش‌ها برای حل مشکلات فنی نرم‌افزار است (به همین دلیل بسیار سرگرم‌کننده است).

به این دلیل که مهندسان نرم‌افزار فقط سازنده نیستند و نرم‌افزار کالا نیست. کامپایلر (که کد منبع را به چیزی تبدیل می‌کند که کامپیوتر می‌تواند اجرا کند) همان چیزی است که در واقع تمام ساختن را انجام می‌دهد. از طرف دیگر مهندس باید بفهمد کامپایلر واقعاً برای ساخت چه چیزی ساخته شده است. این نیاز به حل خلاقانه مسئله دارد.

بنابراین مهندسان شباهت بیشتری به معماران یا حتی طراحان دارند - آنها به‌شدت در مرحله **«طراحی»** فرایند حل مسئله زندگی می‌کنند و به طور مکرر برای حل مشکلات تعریف نشده یا غیرمعمول موردنیاز هستند. آنها برای این کار بیشتر از آنکه به دنبال یافتن و طراحی راه‌حل مناسب در میان آزمایش چیزهای جدید باشند، به فکر ساخت سریع هستند.

این بدان معنا نیست که مهندسان نرم‌افزار دائماً با ابزاری (مانند زبان‌ها و چارچوب‌ها و…) برای کارآمدتر کردن طراحی “نقشه” نرم‌افزار خود روبرو نمی‌شوند … اما همیشه به کسی احتیاج است تا تشخیص دهد چه چیزی باید ساخته شود.

به دلیل خلاقیت ذاتی که برای حل مشکلات نرم‌افزار لازم است، کد منبع چندین مهندس که “محصول مشابه” را می‌سازند احتمالاً بسیار متفاوت به نظر می‌رسد. با وجود این، اگر آنها به اصول، الگوها، طرح‌ها و روش‌های ثابت برای انجام این کار پایبند باشند، همه آنها احتمالاً به راه‌حل‌های مشابه کارآمد دست پیدا می‌کنند و وظیفه خود را انجام می‌دهند.

این اصول، الگوها، طرح‌ها و روش‌های تولید نرم‌افزار خوب، هسته اصلی مهندسی نرم‌افزار را تشکیل می‌دهد. بسیاری از آنها از سایر رشته‌های مهندسی به ارث رسیده‌اند، درحالی‌که دیگران از سال‌ها ساخت نرم‌افزار به وجود آمده‌اند. در این مقاله، نگاهی خواهیم انداخت به برخی از این اصول راهنمای سطح بالا و بهترین روش‌ها.

هدف ما از این مقاله این نیست که شما همه این اصول و کلمات اختصاری را به‌خاطر بسپارید بلکه این است که شما مهندس بودن و چگونگی “کار” مهندسی را درک کنید.

## چگونه یک مهندس یک مسئله را حل می‌کند

رمز و راز مهمی در مهندسی وجود ندارد - شما فقط یک فرایند سیستماتیک را برای حل خلاقانه مشکلات اعمال می‌کنید. این فرایند به شرح زیر است:

1. **مشکل را درک کنید**
2. **برای راه‌حل برنامه‌ریزی کنید**
3. **آن برنامه را اجرا کن**
4. **نتایج خود را برای دستیابی به‌دقت بیشتر بررسی کنید**

چیزی که اکثر مردم نمی‌فهمند این است که بیشتر تلاش قابل‌توجه در دو قسمت اول فرایند انجام می‌شود. اگر در هر یک از این موارد شکست بخورید، مهم نیست که آیا شما برنامه خود را به طور کامل انجام داده‌اید یا دقیق؛ بنابراین اگر می‌خواهید مانند یک مهندس فکر کنید، **قبل از غوطه‌ور شدن در اجرای راه‌حل خود**، **مشکلات خود را کاملاً حل کنید**.

گفتنش از انجامش راحت‌تر است - مهندسان اغلب دوست دارند چیزهایی را ایجاد کنند، تا جایی که ترجیح می‌دهند در ساختن یک چیز جالب غوطه‌ور شوند تا اینکه مطمئن شوند که مسئله درست را حل می‌کنند. صبر کنید، به نظر می‌رسد کمی آشنا است …

در [مینی دوره Design](https://www.vikingcodeschool.com/web-design-basics) گفتیم که اگر شما محصولی را طراحی می‌کنید زیرا فکر می‌کنید طراحی آن جالب است و نه به این دلیل که کاربران بخواهند، شکست خواهید خورد. یک اصل مشابه در مورد کدنویسی توسط مهندسان اعمال می‌شود - اگر بیشتر خود را کدنویس می‌دانید زیرا ساختن سرگرم‌کننده است و مشکل را به طور کامل بررسی نکرده‌اید و یا رویکرد خود را زودتر از موعد برنامه‌ریزی نکرده‌اید، وقت زیادی را تلف خواهید کرد.

اگر به باقی‌مانده این دوره کوتاه توجه کنید، انواع و اقسام موازات بین ایده‌های طراحی کاربرمحور خوب و مهندسی نرم‌افزار خوب را مشاهده خواهید کرد.

## اصول و بهترین روش‌های مهندسی (نرم‌افزار)

مهندسان اصطلاحات اختصاری را بسیار دوست دارند زیرا بیانگر وضوح و سادگی هستند. در حقیقت، تقریباً هرآنچه در مورد مهندسی نرم‌افزار باید بدانید، با اصول، قواعد و اختصارات زیر خلاصه می‌شود. اکثر آنها به اشکال دیگر مهندسی نیز مربوط می‌شوند. لیست زیر، از موارد سطح بالا شروع می‌شود و در انتها به موارد مخصوص به کد ختم می‌شود.

مهم‌ترین آن‌هایی که درک می‌شوند موارد سطح بالا هستند - شما می‌توانید بعداً برگردید و در مورد برخی از نرم‌افزارهای خاص بیشتر فکر کنید، اگر مدتی پشت فرمان کد واقعی باشید!

### شروع سطح بالا

1. **قبل از اینکه بخواهید راه حلی را پیاده کنید، مسئله را کاملاً حل کنید - به‌طورجدی** مهم‌ترین کاری است که احتمالاً می‌توانید انجام دهید.
2. **تقسیم و تسخیر کنید - م**شکلات بزرگ‌تر را به چندین مشکل کوچک‌تر تقسیم کنید که می‌تواند به طور مستقل روی آنها کار شود. آنها شاتل فضایی یا اهرام را در یک‌قدم بزرگ نساختند. به‌این‌ترتیب مشکلات ترسناک قابل‌کنترل می‌شوند.
3. **KISS (ساده و احمقانه نگهش دار)** – KISS (Keep It Simple Stupid) ممکن است شعار مهندس باشد. ما تمایل خود را برای ساختن سیستم‌های بیش از حد پیچیده در بعضی مواقع اذعان کرده‌ایم (جلسات ناشناس برای این نوع کارها وجود دارد) و ما را مجبور می‌کند اعتراف کنیم که سادگی راه‌حل شما را بسیار بهتر می‌کند. این نیست که بگوییم سادگی آسان است! این اغلب نتیجه کشیدن مو و دندان‌قروچه است زیرا محلول‌های پیچیده برای تجزیه اجزای اصلی آنها بازسازی می‌شوند. KISS به روشهای بیشماری دیگر نیز اعمال می شود – به عنوان مثال ، در میان مبتدیان ، تمایل واقعی به بیش از حد فکر کردن در مورد مشکلات منطقی و ارائه راه حل های عجیب و غریب وجود دارد ، در حالی که روش ساده “گنگ” به سادگی بهتر است.
4. **به‌خصوص از اشتباهات خود بیاموزید - م**هندسی همیشه در حال تغییر است و به‌ویژه مهندسی نرم‌افزار ممکن است یکی از سریع‌ترین زمینه‌های تغییر در کره زمین باشد. شما باید همیشه یاد بگیرید، هم از افراد دیگر این صنعت و هم از طریق پذیرفتن اشتباهات و کدهای ضعیف خود. این دستورالعمل یک حرفه طولانی و تحریک‌کننده در این زمینه است.
5. **دلیل وجود این نرم‌افزار را به‌خاطر بسپارید - ت**صمیمات کوچک بی‌شماری وجود دارد که در طول کار خود به‌عنوان یک توسعه‌دهنده با آنها روبرو خواهید شد و اگر تصویر بزرگ‌تر را نمی‌دانید، ممکن است وقت خود را برای رفتن به یک مسیر اشتباه تلف کنید. این تقریباً اصل اساسی است!
6. **به یاد داشته باشید که شما از این نرم‌افزار استفاده نخواهید کرد - م**هندسان بسیار وسوسه دارند که راه‌حل‌هایی را برای خود طراحی کنند. به کاربر فکر کنید نه به خودتان. تصور نکنید کاربر شما همان سطح صلاحیت فنی یا آشنایی با سیستم را دارد که شما می‌دانید.

### فکرکردن در مورد روند کار

1. **چشم‌انداز روشنی برای پروژه داشته باشید - ا**گر دقیقاً نمی‌دانید چه چیزی را می‌خواهید بسازید، چیز اشتباهی خواهید ساخت. اغلب، به‌خصوص اگر به‌صورت آزادکار می‌کنید یا در یک تیم کار می‌کنید، این بدان معناست که سؤالات مهمی از مشتری می‌پرسید. ضرب‌المثل قدیمی این است: “شما دقیقاً همان چیزی را ساختید که ما خواسته بودیم، اما آنچه را که نیاز داریم، درست نکردید”.
2. **یک فرایند دقیق داشته باشید - م**هندسی نرم‌افزار یک فعالیت خلاقانه در طراحی است اما باید به‌صورت سیستماتیک انجام شود. نیازی نیست که در روند کار غرق شوید، اما نمی‌توانید باعجله به راه‌حل برسید. ما در حال حل برخی از مشکلات بسیار پیچیده هستیم، بنابراین شما باید مراقب یک روش منطقی و متفکرانه برای حل آنها و یک رویکرد دقیق برای مدیریت پروژه‌های خود باشید، در غیر این صورت آنها به‌سرعت از شما دور می‌شوند.
3. **برنامه‌ها را به‌سرعت توسعه دهید - ا**ین یکی از مواردی است که در بخش Agile کمی بیشتر به آن خواهیم پرداخت اما مطمئناً در اینجا قابل‌ذکر است. نیازهای پروژه نرم‌افزار به طور مداوم تغییر می‌کند. هرچه روند کار شما سریع‌تر باشد، بهتر می‌توانید به آن تغییرات پاسخ دهید.
4. **ابتدا کار را انجام دهید، سپس درستکار کنید، سپس زیبا به نظر برسید - ا**ین سازگاری است با مورد “توسعه سریع برنامه‌ها” و دوباره بر فلسفه “اول امتحانش کنید” تأکید دارد. به طور خاص، این مقاله می‌گوید که شما ابتدا باید چیزی بسازید که کار کند (حتی اگر با سیم و نوارچسب کنار هم نگه‌داشته شود) و سپس ببینید آیا ارزش دارد که برای درستکار کردن وقت بگذارید. تا آخرین مرحله فرایند، نباید واقعاً راه‌حل شما “خوب” به نظر برسد (به‌عنوان‌مثال، دوباره‌سازی کد خود).

### فکرکردن به (کد کردن) راه‌حل

1. (**You Ain’t Gonna Need It**) **YAGNI (شما دیگر نیازی به آن نخواهید داشت!) - ی**ک وسوسه بسیار قوی برای ساختن کدی وجود دارد که می‌تواند با انعطاف‌پذیری فوق‌العاده و “عالی” به هرگونه نیاز آینده پاسخ دهد. این کار را انجام ندهید! شما در حال هدردادن تلاش خود هستید زیرا واقعاً نیازی به آن‌همه ویژگی یا گزینه اضافی یا انعطاف‌پذیری اضافی نخواهید داشت. فقط آنچه نیاز دارید بسازید. به ما و هر مهندس دیگری که به دنبال کد قدیمی است و تمام تلاش‌های بیهوده خود را خسته می‌کند اعتماد کنید.
2. **DRY (Don’t Repeat Yourself) (خود را تکرار نکنید) - ی**کی از بهترین موارد در مورد کد استفاده مجدد از آن است. اگر یک کد خوب می‌نویسید که یک مشکل مفید را در یک مکان حل می‌کند، وقتی مشکل در جاهای دیگر هم پیش آمد، دوباره به آن مراجعه کنید. از دیدگاه شما، هر زمان که خود را پیدا می‌کنید و در چند بار به‌صورت دستی تایپ می‌کنید، راهی وجود دارد که همه آن را در یک کار واحد ترکیب کنید که چندین بار اجرا می‌شود.
3. **انتزاع (Abstraction) را بپذیرید - د**ر مهندسی نرم‌افزار همه چیز در مورد انتزاع یا نادیده‌گرفتن جزئیات و حل مشکلات مرتبه بالاتر است. به هیچ دلیلی نیازی به نوشتن کد ماشین یا کد اسمبلی ندارید - زبان‌های برنامه‌نویسی امروزی به شما امکان می‌دهند فقط به کامپیوتر بگویید که چه می‌خواهید و آن را ارائه می‌دهد. این امر همچنین در مورد چگونگی برخورد شما با سیستم‌های پیچیده نیز صدق می‌کند - تمرکز خود را بر این بگذارید که از عملکرد صحیح سیستم بدون نیاز به دانستن جزئیات پیاده‌سازی هر یک از اجزای سازنده، اطمینان حاصل کنید.
4. **(Don’t Re Invent The Wheel)** **DRITW (چرخ را دوباره اختراع نکنید) - خ**وب، ما مخفف آن را ساختیم اما مهم است. هر زمان که برای انجام کار کلی کد ساخت‌وساز دارید که مستقیماً با اصول برنامه شما ارتباط ندارد، شخص دیگری احتمالاً قبلاً آن کد را نوشته و بهتر از آن نوشته است. یا در جایی در یک وبلاگ، در Stack Overflow ارسال شده است ، یا به عنوان یک جواهر دارای منبع آزاد است. به جای اتلاف وقت خود در اختراع چرخ ، از کد آنها بیاموزید و از آنها استفاده کنید.
5. **کدی بنویسید که یک کار را به‌خوبی انجام دهد - ی**ک قطعه کد فقط باید یک کار را انجام دهد و آن را به‌خوبی انجام دهد. اگر می‌خواهید یک راه‌حل معجزه‌آسا برای همه کارها درست کنید و همه آن را به یک قطعه کد دهید، کابوسی برای قابلیت نگهداری دارید و احتمالاً چندین عمل را نقض می‌کند. KISS!
6. **اشکال‌زدایی دشوارتر از نوشتن کد است - ک**د قابل خواندن بهتر از کد جمع‌وجور است؛ بنابراین تلاش برای ترکیب این 10 خط در یک زنجیره عملیاتی از روش‌هایی که از الگوهای ورودی مبهم استفاده می‌کنند، متوقف شود! بعداً شخص دیگری باید آن را بخواند و اشکال‌زدایی کند و شما به آنها لطفی نمی‌کنید.
7. **Kaizen1 (آن را بهتر از زمانی که پیدا کردید بگذارید) - نه‌تنها** اشکالی که می‌خواهید حل کنید بلکه کد موجود در آن را برطرف کنید. اگر مشکل اصلی ایراد در طراحی باشد (که معمولاً وجود دارد) رفع اشکال کد کمکی نمی‌کند.

## حساب سرانگشتی

* یافتن و رفع یک مشکل نرم‌افزاری در تولید، **100 برابر** گران‌تر از یافتن و رفع آن در مرحله نیاز و مرحله طراحی است. آن اشکالات و نقایص طراحی را زود بیابید!
* در همین مورد، بیش از نیمی از خطاهای موجود در طراحی در مرحله طراحی مرتکب می‌شوند.
* تقریباً تمام مدت صرف شده برای رفع مشکلات را می‌توان به تعداد انگشت شماری از ماژول‌های مشکل‌دار نسبت داد.
* حدود نیمی از برنامه‌های کاربر دارای نقایص غیراساسی است
* فقط 60٪ از ویژگی‌های یک سیستم در واقع در تولید استفاده می‌شود

## جمع‌بندی

پس هنوز موضوعات اصلی را دریافت می‌کنید؟ **اول فکر کنید، مشکلات را تکه‌تکه کنید و کارها را ساده انجام دهید.**اگر این را فهمیده‌اید، شما رسماً در یک مسیر با سایر اعضای جامعه مهندسی هستید. هر چیز دیگری فقط جزئیات پیاده‌سازی است.