در برنامه نویسی، مفهوم "KISS" به عنوان یک اصل طراحی نرم افزاری به کار می‌رود. "KISS" مخفف عبارت "Keep It Simple, Stupid" یا " Keep It Simple and Straightforward" است و به معنای "ساده نگه داشتن، احمقانه نکنید" می‌باشد.

این اصل طراحی تاکید بر روی استفاده از راه‌حل‌های ساده و قابل فهم برای حل مسائل دارد. به عبارت دیگر، زمانی که از اصل KISS پیروی می‌شود، سعی می‌شود که هر فرآیند یا کدی که نوشته می‌شود، به سادگی قابل فهم و مدیریت باشد.

اصل KISS هدفش این است که از پیچیدگی‌های غیرضروری و اضافی در طراحی نرم‌افزار جلوگیری کند. این اصل می‌تواند به کدنویسی، طراحی رابط کاربری، ساختار دیتابیس و سایر جنبه‌های توسعه نرم‌افزار اعمال شود.

به طور خلاصه، اصل KISS توصیه می‌کند که در هنگام توسعه نرم‌افزار، از پیچیدگی‌ها و اضافی‌ها جلوگیری شود و به جای آن، راه‌حل‌های ساده و قابل فهم به کار گرفته شود.

اصل YAGNI یک اصل اساسی در مهندسی نرم‌افزار است که مخفف عبارت "You Aren't Gonna Need It" است. این اصل از اصول مهم متدهای توسعه نرم‌افزاری مانند متدولوژی توسعه نرم‌افزار خیزی (Agile) و برنامه‌نویسی افزادانه (Extreme Programming) است.
به طور خلاصه، اصل YAGNI بیان می‌کند که برنامه‌نویسان نباید قراردادن کدهایی را که در حال حاضر نیازی به آنها نیست بپذیرند. به جای این‌که طراحی یا پیاده‌سازی کدهای اضافی که ممکن است در آینده استفاده شوند، برنامه‌نویسان تمرکز خود را بر روی نیازهای فعلی واضح و مشخص مشتری متمرکز می‌کنند.اهمیت اصل YAGNI این است که باعث کاهش پیچیدگی بی‌مورد کد می‌شود و از ایجاد کدهای اضافی که ممکن است هیچ‌وقت استفاده نشوند جلوگیری می‌کند. این اصل همچنین به توسعه سریع‌تر و تسهیل تغییرات در آینده کمک می‌کند.
بنابراین، با توجه به اصل YAGNI، برنامه‌نویسان باید از اضافه کردن کدهای از پیش تصمیم‌گیری شده و توهمی که ممکن است در آینده مفید باشند پرهیز کنند.

اصل DRY در برنامه‌نویسی به معنای "Don't Repeat Yourself" یا به مخفف DRY است. این اصل مهمی است

که توسط برنامه‌نویسان به عنوان یک راهنمای معمول برای نوشتن کدهای بهینه و قابل نگهداری در نظر گرفته می‌شود. اصل DRY بیان می‌کند که هر بخش از سیستم نباید در چندین قسمت مختلف تکرار شود، بلکه یک جایگاه مشخص برای آن بخش باید وجود داشته باشد و از آنجا استفاده شود.
استفاده از اصل DRY به برنامه‌نویسان کمک می‌کند تا کد بهینه‌تری بنویسند و از مشکلات نگهداری کد کمتری رنج ببرند. با رعایت اصل DRY، اگر یک تغییر در اجزای مشترک کد نیاز باشد، تنها کافی است که تغییری را در یک جا اعمال کنند، و این تغییر به صورت خودکار در بقیه بخش‌ها اعمال خواهد شد، که این موضوع به معنای افزایش قابلیت نگهداری، توسعه و تست کد است.
در کل، اصل DRY توصیه می‌کند که برنامه‌نویسان از تکرار کد و اطلاعات خودداری کنند و برای اشتراک اطلاعات و قطعه‌کدهای مشترک، از مکانیزم‌ها و ویژگی‌های مدولاریته و تجزیه و تحلیل خوبی استفاده کنند.

اصل SOLID یک مجموعه از اصول طراحی است که برای ایجاد کد قابل توسعه، قابل نگهداری و با کیفیت استفاده می‌شود. این اصول اولین بار توسط رابرت سی. مارتین (به نام "آمبول" مشهور است) مطرح شد و هر یک از حروف

یک اصل خاص را نشان می‌دهند SOLID:

S - Single Responsibility Principle
(اصل مسئولیت تکی):
یک کلاس یا ماژول باید تنها یک مسئولیت یا وظیفه را داشته باشد.

O - Open/Closed Principle
(اصل باز/بسته):
کلاس‌ها باید برای توسعه‌پذیری باز و برای تغییرات بسته باشند، به این معنی که تغییر یک کلاس باید بدون تغییر کد موجودیت‌های دیگر امکان‌پذیر باشد.

L - Liskov Substitution Principle
(اصل جایگزینی لیسکف):
اگر S نوع زیر کلاس T است، باید هر جایی که T مورد استفاده قرار می‌گیرد، می‌توان S را جایگزین کرد.

I - Interface Segregation Principle
(اصل تجزیه و تفکیک رابط):
کاربردهای مختلف یک رابط باید از آن رابط به شکل مجزا استفاده کنند و نباید وابسته به چیزهایی باشند که نیاز

ندارند.

D - Dependency Inversion Principle
(اصل وابستگی معکوس):
کلاس‌ها باید به نوعی باشند که وابستگی به اجزای دیگر برای آنها مناسب باشد و انتظار از اجزا به جزییات معکوس باشد.

این اصول، اگر به درستی پیاده‌سازی شوند، باعث کاهش وابستگی‌ها، افزایش خوانایی کد، کاهش احتمال خطاها و افزایش سهولت توسعه و نگهداری کد می‌شود.