# سخن اول

ما به دو دلیل Clean Code می خونیم دلیل اول اینکه برنامه نویس هستیم دلیل دوم اینکه قصد داریم برنامه نویس بهتری بشیم.

# [بخش اول](https://virgool.io/@NimaShirinzadeh/clean-code-%D8%A8%D8%AE%D8%B4-%D8%A7%D9%88%D9%84-dgzw9ik0zusd)

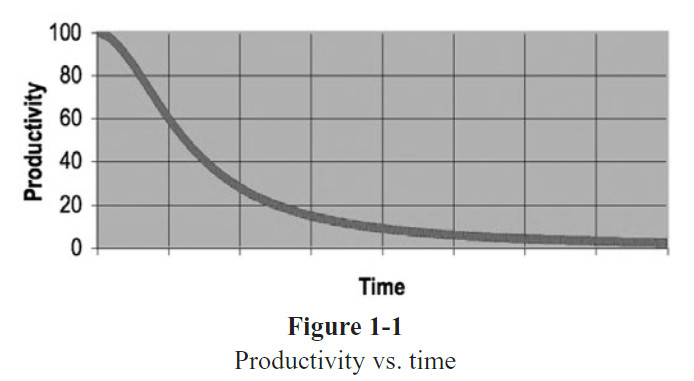
# چرا Clean Code

ما مجبوریم کد تمیز بنویسیم چون با کد کثیف رفع باگ، ارائه فیچر یا ارائه نسخه جدید سخت تر و سختر میشه و با بزرگ شدن پروژه هی کار سخت تر میشه و فاصله بین ریلیزها بیشتر میشه تا در نهایت به نقطه ای میرسیم که مجبوریم مرگ پروژه رو اعلام کنیم.

# چرا کد کثیف می نویسیم

چون ددلاین از آنچه فکر می کنیم به ما نزدیکتر است. مدیران رو باید به اهمیت کد تمیز آگاه کنیم و در برآوردهامون زمان تمیز کردن یا تمیز نوشتن رو هم در نظر بگیریم.

# نمودار نسبت بازده به زمان بر اساس یک کد بد



# [بخش دوم](https://virgool.io/@NimaShirinzadeh/clean-code-%D8%A8%D8%AE%D8%B4-%D8%AF%D9%88%D9%85-i3dth5q3dinc)

هدف این بخش اینه که کدهایی که می نویسیم Quick skim‌ باشه و خوندن کدها مثل خوندن روزنامه راحت و روان باشه.

خوب اول اسم تمام متغیرها، فانکشن‌ها و ... باید Meaningful Names باشه.

# تفاوت بین نام معنادار و بدون معنا

نام یک متغیر، کلاس یا فانکشن باید بتونه دلیل وجودی، کاری که داره انجام میده، جا و نحوه ی استفاده شو توصیف کنه.

# نمونه نام های بودن معنا

مخاطب رو دچار سوءتفاهم می کنه

اسامی که در کلماتشون تفاوت کمی وجود داره اسامی مناسبی نیستند.

پرهیز از ایجاد تمایز با انتخاب کلمات متفاوت بدون معنا مثل استفاده از اعداد یا استفاده از کلماتی مثل a، an و the و ...

استفاده از اسامی غیر قابل تلفظ و ناخوانا

برای یک مفهوم چند کلمه انتخاب نشه و یک کلمه چند مفهوم را پشتیبانی نکنه مثلا واژه get همیشه یک مفهوم رو ساپورت کنه

# دو استراتژی برای انتخاب نام

بر اساس راه حل (بهترین استراتژی)

براساس مشکل (تنها مشکل این استراتژی زمانی هست که شخص دیگه ای بخواد روی کد کار کنه بنابراین باید در مورد مشکل از شخصی که توی اون پروژه تجربه داشته سوال بپرسه.)

نکته: زمانی که تمرکز یه کد بیشتر روی حل یک مشکل هست بهتره که اسم از دامنه مشکلات انتخاب بشه.

# [بخش سوم](https://virgool.io/@NimaShirinzadeh/clean-code-%D8%A8%D8%AE%D8%B4-%D8%B3%D9%88%D9%85-owpf1yesn4k1)

در این بخش میخوایم در مورد functionها صحبت کنیم.

# اندازه فانکشن

اول اینکه فانکشن نباید بزرگ باشه چون خواندن، درک کد، تست و اشکال یابی رو سخت می کنن.

هر فانکشن باید کمتر از 20 خط کد باشد. براساس کدهای Kent Beck هر فانکشن 3 تا 4 خط.

بهتر است کدهای درون if و while به یه فانکشن منتقل بشه و if و while تک خطی بشن.

سطوح تو در تو از یک تا دو سطح نباید بیشتر بشه.

# انجام یک کار

فانکشن ها باید یک عمل را انجام بدهند. ان ها باید ان را به خوبی انجام دهند. و فقط ان را انجام دهند.

نکته: تلاش برای کوچک بودن فانکشن از چند کاره بودن ان جلوگیری می کند و تلاش برای ساخت فانکشن با یک وظیفه به کوچک بودن ان کمک می کند. به نوعی این دو قانون برای یک دیگر نقش کاتالیزور رو دارند.

نکته-Use Descriptive Names: انتخاب نام مناسب از اهمیت بالایی برخوردار است. هر قدر که توابع ما کوچک تر باشند و بر روی یک وظیفه متمرکز باشند انتخاب اسم مناسب (descriptive name) برای ما اسان تر می شود.

نکته: اسم طولانی ولی توصیف کننده عملکرد فانکشن از یک اسم کوتاه و مبهم و حتی از کامنت گذاری هم مهم تر و بهتر است.

# پارامترهای فانکشن

بهترین تعداد ورودی برای هر فانکشن 0 است.

وجود بیش از سه ورودی برای توابع قابل قبول نیست.

نکته: وجود ورودی برای فانکشن ها نه تنها اشنایی با عملکرد فانکشن های را سخت می کند بلکه در هنگام تست هم باید ترکیبات ورودی ها مورد تست قرار گیرد. که معمولا فانکشن های بیش از 2 ورودی فرایند تست سختی دارند.

# استراتژی های کاهش پارامترهای فانکشن

زمانی که یک فانکشن بیش از دو ورودی داشته باشد در صورت امکان دو استراتژی برای کاهش ورودی‌ها وجود دارد.

1. بعضی از ورودی ها را بتوان درون یک کلاس قرار داد.

🗷Circle makeCircle(double x, double y, double radius);

🗹Circle makeCircle(Point center, double radius);

1. برای بعضی از ورود‌ها با یک ترتیب خاص یک اتفاق خاص می افتد که در این صورت می توان از یک List یا Object به جای تمام ان ورود ها استفاده کرد.

🗷String.format("%s worked %.2f hours.", name, hours);

🗹String.format("%s worked %.2f hours.",List or Object);

نکته: یک فانکشن باید در یک زمان یا کاری انجام دهد یا جوابی تولید کند. انجام هر دو کار در یک زمان برای یک فانکشن درست نیست.

🗷public boolean set(String attribute, String value);

🗹public Boolean AttributeExists(String attribute);

🗹public void set(String attribute, String value);

# جلوگیری از تکرار

یکی از اهداف اصلی نوشتن فانکشن ها جلوگیری از تکرار کد هست . در صورت داشتن کد تکراری برای اصلاح یا بهبود عمکرد آن کد باید n جا رو تغییر داد.

اصول و روش های مختلفی برای کم کردن و جلوگیری از تکرار معرفی شده ک برخی از اونها عبارتند از:

object-oriented programming

Structured programming

Aspect Oriented Programming

Component Oriented Programming