بسمه تعالی

درس مهندسی نرم افزار(جبرانی)

استاد :

جناب آقای دکتر فراهی

محقق:

علی جلالی

شماره دانشجویی:

۹۸۰۲۱۸۶۹۶

عنوان تحقیق:

اصول محوری مهندسی نرم افزار

شروع سطح بالا

- **قبل از اینکه بخواهید راه حلی را پیاده کنید ، مسئله را کاملاً حل کنید** – به طور جدی مهمترین کاری است که احتمالاً می توانید انجام دهید.

- **تقسیم و تسخیر کنید** – مشکلات بزرگتر را به چندین مشکل کوچکتر تقسیم کنید که می تواند به طور مستقل روی آنها کار شود. آنها شاتل فضایی یا اهرام را در یک قدم بزرگ نساختند. به این ترتیب مشکلات ترسناک قابل کنترل می شوند.

- **KISS (ساده و احمقانه نگهش دار)** – KISS (Keep It Simple Stupid) ممکن است شعار مهندس باشد. ما تمایل خود را برای ساختن سیستم های بیش از حد پیچیده در بعضی مواقع اذعان کرده ایم (جلسات ناشناس برای این نوع کارها وجود دارد) و ما را مجبور می کند اعتراف کنیم که سادگی راه حل شما را *بسیار بهتر می کند*. این نیست که بگوییم سادگی آسان است! این اغلب نتیجه کشیدن مو و دندان قروچه است زیرا محلولهای پیچیده برای تجزیه اجزای اصلی آنها بازسازی می شوند. 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.

- **به خصوص از اشتباهات خود بیاموزید** – مهندسی همیشه در حال تغییر است و به ویژه مهندسی نرم افزار ممکن است یکی از سریع ترین زمینه های تغییر در کره زمین باشد. شما باید همیشه یاد بگیرید ، هم از افراد دیگر این صنعت و هم از طریق پذیرفتن اشتباهات و کد های ضعیف خود. این دستورالعمل یک حرفه طولانی و تحریک کننده در این زمینه است.

- **دلیل وجود این نرم افزار را به خاطر بسپارید** – تصمیمات کوچک بی شماری وجود دارد که در طول کار خود به عنوان یک توسعه دهنده با آنها روبرو خواهید شد و اگر تصویر بزرگتری را نمی دانید ، ممکن است وقت خود را برای رفتن به یک مسیر اشتباه تلف کنید. این تقریباً اصل اساسی است!

- **به یاد داشته باشید که شما از این نرم افزار استفاده نخواهید کرد** – مهندسان بسیار وسوسه دارند که راه حل هایی 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 می دانید.

فکر کردن در مورد روند کار

- **چشم انداز روشنی برای پروژه داشته باشید** – اگر دقیقاً نمی دانید چه چیزی را می خواهید بسازید ، چیز اشتباهی خواهید ساخت. اغلب ، به خصوص اگر به صورت آزاد کار می کنید یا در یک تیم کار می کنید ، این بدان معناست که سوالات مهمی از مشتری می پرسید. ضرب المثل قدیمی این است: "شما دقیقاً همان چیزی را ساختید که ما خواسته بودیم ، اما آنچه را که نیاز داریم ، درست نکردید".

- **یک فرآیند دقیق داشته باشید** – مهندسی نرم افزار یک فعالیت خلاقانه در طراحی است اما باید بصورت سیستماتیک انجام شود. نیازی نیست که در روند کار غرق شوید ، اما نمی توانید با عجله به راه حل 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 پروژه های خود باشید ، در غیر این صورت آنها به سرعت از شما دور می شوند.
- **برنامه ها را به سرعت توسعه دهید** – این یکی از مواردی است که در بخش Agile کمی بیشتر به آن خواهیم پرداخت اما مطمئناً در اینجا قابل ذکر است. نیازهای پروژه نرم افزار به طور مداوم تغییر می کند. هرچه روند کار شما سریعتر باشد ، بهتر می توانید به آن تغییرات پاسخ دهید.

- **ابتدا کار را انجام دهید ، سپس درست کار کنید ، سپس زیبا به نظر برسید** – این سازگاری است با مورد "توسعه سریع برنامه ها"، و دوباره بر فلسفه "اول امتحانش کنید" تأکید دارد. به طور خاص ، این مقاله می گوید که شما ابتدا باید چیزی بسازید که کار کند (حتی اگر با سیم و نوار چسب کنار هم نگه داشته شود) و سپس ببینید آیا ارزش دارد که برای درست کار کردن وقت بگذارید. تا آخرین مرحله فرآیند ، نباید واقعاً راه حل شما "خوب" به نظر برسد (به عنوان مثال ، دوباره سازی کد خود).

فکر کردن به (کد کردن) راه حل

- YAGNI (You Ain’t Gonna Need It) (شما دیگر نیازی به آن نخواهید داشت!) – یک وسوسه بسیار قوی برای ساختن کدی وجود دارد که می تواند با انعطاف پذیری فوق العاده و "عالی" به هرگونه نیاز آینده پاسخ دهد. این کار را انجام ندهید! شما در حال هدر دادن تلاش خود هستید زیرا واقعاً نیازی به آن همه ویژگی یا گزینه اضافی یا انعطاف پذیری اضافی نخواهید داشت. فقط آنچه نیاز دارید بسازید. به ما و هر مهندس دیگری که به دنبال کد قدیمی است و تمام تلاش های بیهوده خود را خسته می کند اعتماد کنید.
- DRY (Don’t Repeat Yourself) (خود را تکرار نکنید) – یکی از بهترین موارد در مورد کد استفاده مجدد از آن است. اگر یک کد خوب می نویسید که یک مشکل مفید را در یک مکان حل می کند ، وقتی مشکل در جاهای دیگر هم پیش آمد ، دوباره به آن مراجعه کنید. از دیدگاه شما ، هر زمان که خود را پیدا می کنید و در چند بار به صورت دستی تایپ می کنید ، راهی وجود دارد که همه آن را در یک کار واحد ترکیب کنید که چندین بار اجرا می شود.
- انتزاع (Abstraction) را بپذیرید – در مهندسی نرم افزار همه چیز در مورد انتزاع یا نادیده گرفتن جزئیات و حل مشکلات مرتبه بالاتر است. به هیچ دلیلی نیازی به نوشتن کد ماشین یا کد اسمبلی ندارید – زبانهای برنامه نویسی امروزی به شما امکان می دهند فقط به کامپیوتر بگویید که چه می خواهید و آن را ارائه می دهد. این امر همچنین در مورد چگونگی برخورد شما با سیستم های پیچیده نیز صدق می کند – تمرکز خود را بر این بگذارید که از عملکرد صحیح سیستم بدون نیاز به دانستن جزئیات پیاده سازی هر یک از اجزای سازنده ، اطمینان حاصل کنید.
- DRITW (Don’t Re Invent The Wheel) (چرخ را دوباره اختراع نکنید) – خوب ، ما مخفف آن را ساختیم اما مهم است. هر زمان که برای انجام کار کلی کد ساخت و ساز 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.
- کدی بنویسید که یک کار را به خوبی انجام دهد – یک قطعه کد فقط باید یک کار را انجام دهد و آن را به خوبی انجام دهد. اگر می خواهید یک راه حل معجزه آسا 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 می کند. !KISS
- اشکال زدایی دشوارتر از نوشتن کد است – کد قابل خواندن بهتر از کد جمع و جور است. بنابراین تلاش برای ترکیب این ۱۰ خط در یک زنجیره عملیاتی از روش هایی که از الگوهای ورودی مبهم استفاده می کنند ، متوقف شود! بعداً شخص دیگری باید آن را بخواند و اشکال زدایی کند و شما به آنها لطفی نمی کنید.

- Kaizen1 (آن را بهتر از زمانی که پیدا کردید بگذارید) – نه تنها اشکالی که می خواهید 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 نمی کند.

حساب سر انگشتی

- یافتن و رفع یک مشکل نرم افزاری در تولید، ۱۰۰ 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 می شوند.
- تقریباً تمام مدت صرف شده برای رفع مشکلات را می توان به تعداد انگشت شماری از ماژول های مشکل دار نسبت داد.
- حدود نیمی از برنامه های کاربر دارای نقایص غیر اساسی است
- فقط ۶۰٪ از ویژگیهای یک سیستم در واقع در تولید استفاده می شود