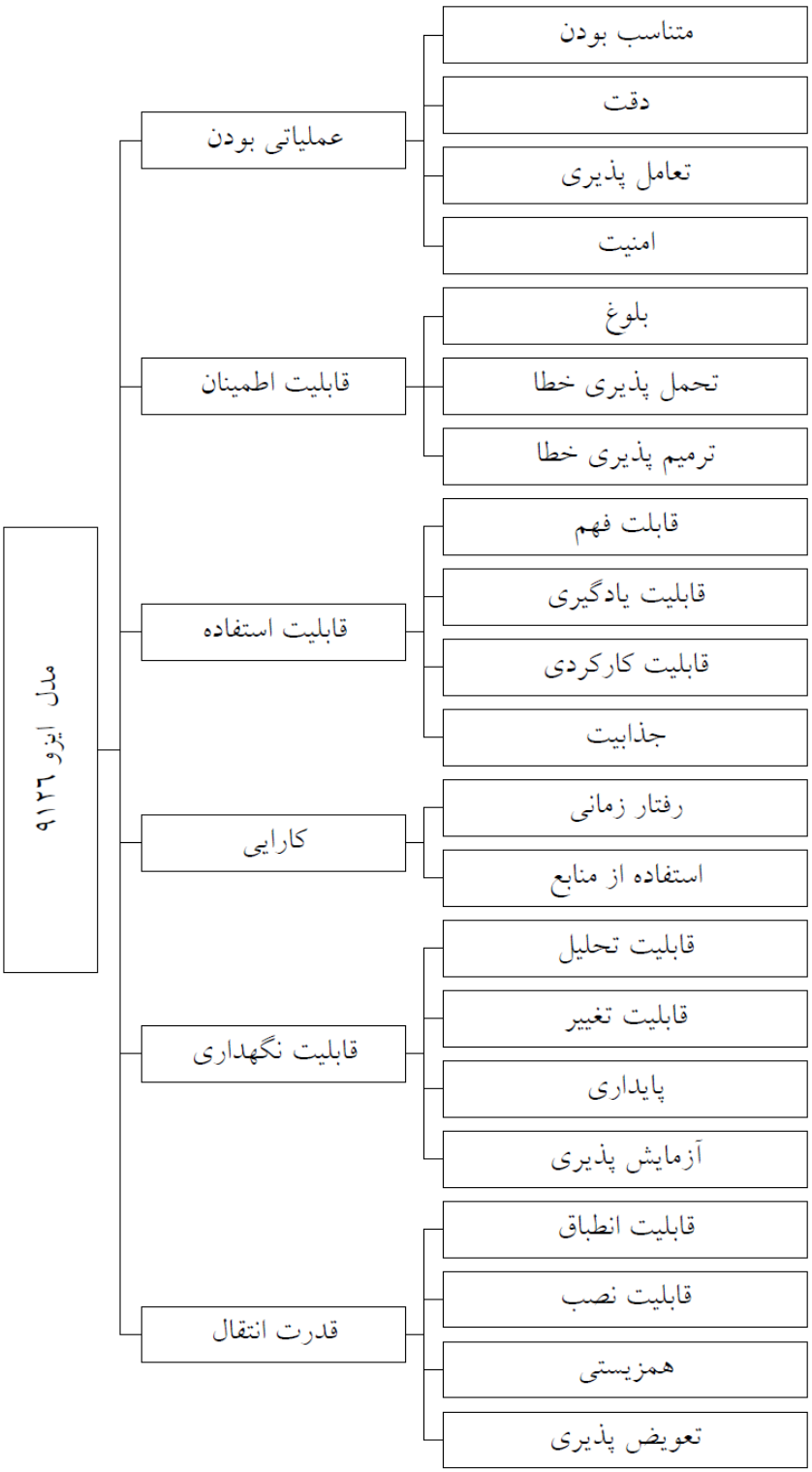
گروه 7 ، گزارش کار 7

محمد رئوفی صدر ، محمد مهدی معصومی



1.**قابلیت نگهداری**

تجزیه و تحلیل: اقدامات تلاش یا منابع نگهدارنده صرف شده در تلاش برای تشخیص نواقص یا علل عدم موفقیت یا شناسایی قسمت هایی که باید اصلاح شوند.

تغییرپذیری: اقدامات تلاش نگهدارنده در رابطه با اجرای یک اصلاح خاص.

پایداری: اقدامات مربوط به رفتار غیر منتظره نرم افزار ، از جمله مواردی است که در هنگام آزمایش مشاهده می شود.

قابلیت تست: اقدامات تلاش نگهدارنده و کاربران در تلاش برای آزمایش نرم افزار اصلاح شده.

اقدامات دیگری که نگهدارنده ها استفاده می کنند شامل

اندازه نرم افزار ،

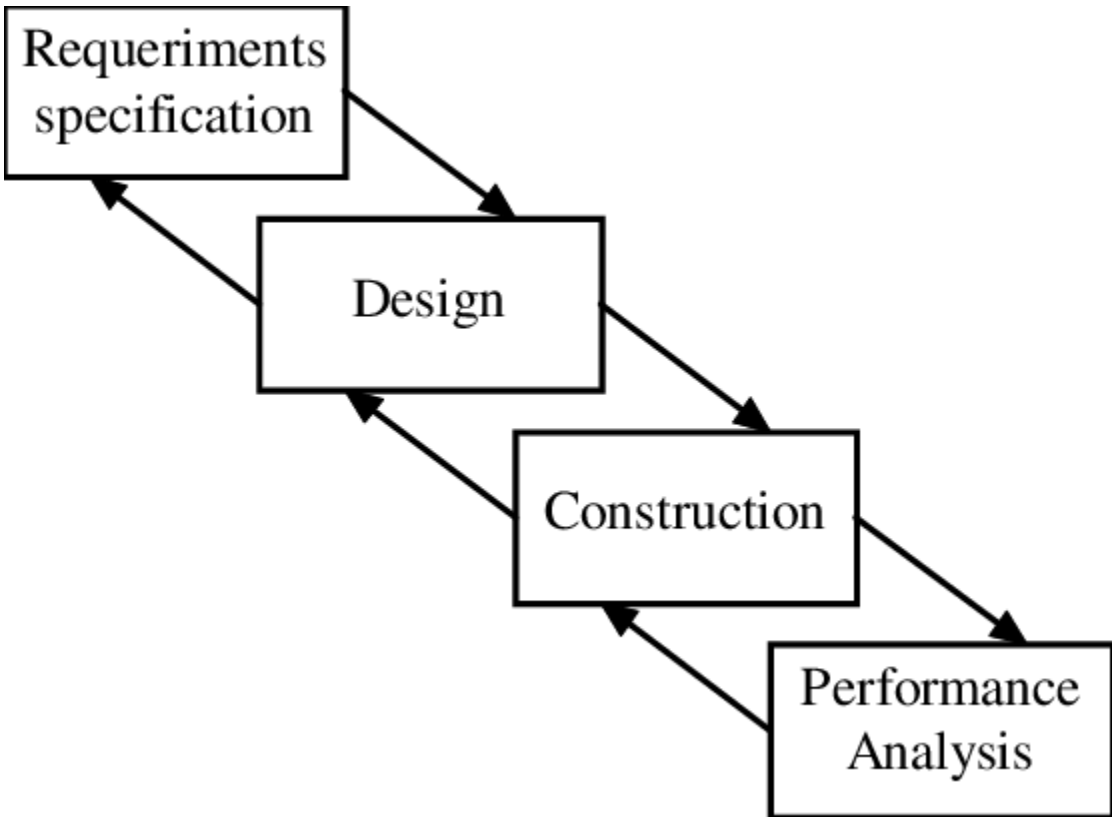
پیچیدگی نرم افزار ،

قابل فهم ، و قابلیت نگهداری

2.**کارایی**

تجزیه و تحلیل ، لحن و امتحان مجدد - تحکیم ، تجزیه و تحلیل و به اشتراک گذاری نتایج آزمون. سپس مرتب سازی کنید و دوباره تست کنید تا ببینید آیا پیشرفت یا کاهش عملکرد وجود دارد یا خیر. از آنجایی که پیشرفت ها معمولاً با هر بار آزمایش کوچکتر می شوند ، در هنگام ایجاد تنگناها توسط CPU متوقف شوید. سپس ممکن است گزینه افزایش قدرت پردازنده را در نظر بگیرید

Plan و تست های عملکرد طراحی - تعیین کنید که چطور ممکن است میزان استفاده در بین کاربران نهایی متفاوت باشد و سناریوهای کلیدی را برای آزمایش تمام موارد استفاده احتمالی مشخص کنید. لازم است که انواع مختلفی از کاربران نهایی شبیه سازی شود ، داده های تست عملکرد را برنامه ریزی کرده و مشخص کنید که چه معیارهایی جمع آوری خواهد شد.



3.**قابلیت اطمینان**

طبق ANSI ، قابلیت اطمینان نرم افزار به این شرح است: احتمال عملکرد نرم افزار بدون عیب برای مدت زمان مشخص در یک محیط مشخص.

قابلیت اطمینان نرم افزار برای ویژگی های کیفیت نرم افزار ، همراه با قابلیت ها ، قابلیت استفاده ، عملکرد ، قابلیت سرویس دهی ، قابلیت ، نصب ، قابلیت نگهداری و مستندات مهم است. رسیدن به قابلیت اطمینان نرم افزار بسیار سخت است ، زیرا پیچیدگی نرم افزار تمایل به بالا بودن دارد.

**معیارهای قابلیت اطمینان نرم افزار

1.معیارهای محصول

تصور می شود اندازه نرم افزار منعکس کننده پیچیدگی و قابلیت اطمینان است

2.معیارهای مدیریت پروژه

محققان دریافته اند که مدیریت خوب می تواند منجر به تولید محصولات بهتر شود

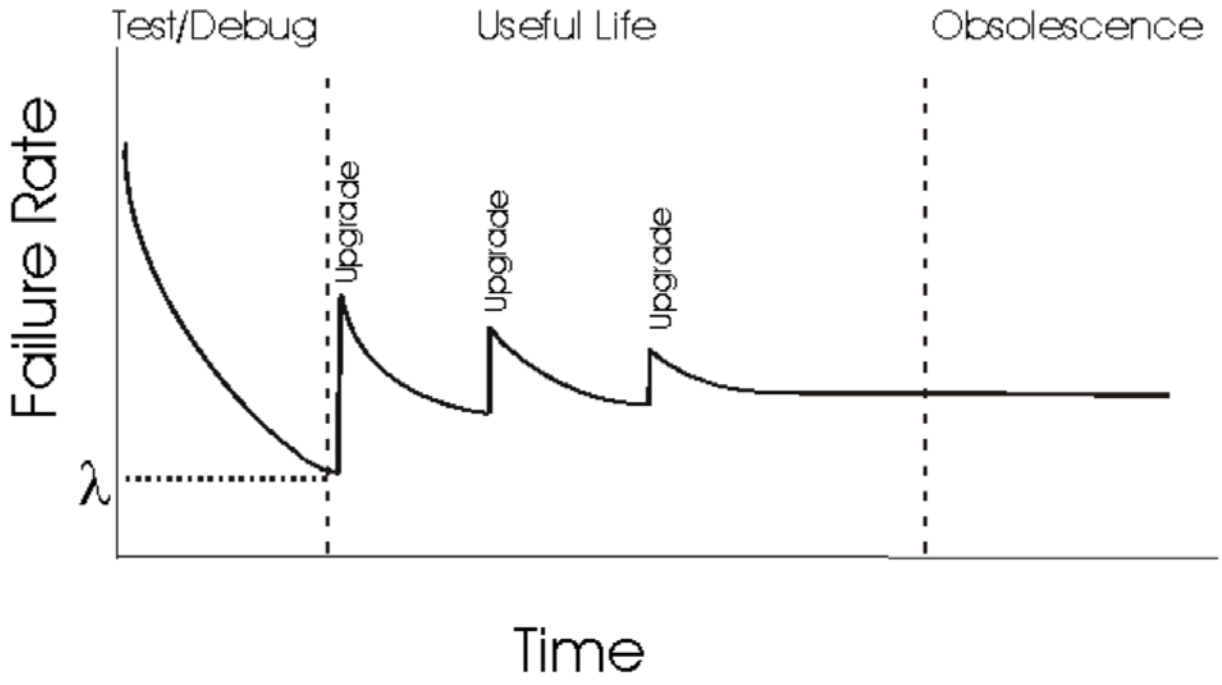
قابلیت اطمینان بالاتری با استفاده از فرآیند توسعه بهتر ، فرایند مدیریت ریسک ، فرایند مدیریت پیکربندی و غیره حاصل می شود.

3.معیارهای پردازش

بر اساس این فرض که کیفیت محصول یک عملکرد مستقیم از فرآیند است ، می توان از معیارهای پردازش برای تخمین ، نظارت و بهبود قابلیت اطمینان و کیفیت نرم افزار استفاده کرد. صدور گواهینامه ISO-9000 یا "استانداردهای مدیریت کیفیت" مرجع عمومی برای خانواده ای از استانداردهای تولید شده توسط سازمان استاندارد بین المللی (ISO) است.

4. معیارهای خطا و خرابی

هدف از جمع آوری معیارهای خطا و خرابی این است که بتوانید چه زمانی این نرم افزار را به اجرایی بدون عدم موفقیت نزدیک کنید



4.**قابلیت استفاده کردن**

در این بخش تعریف قابلیت استفاده در زمینه توسعه نرم افزار و ارتباط آن با سایر جنبه های فرآیند توسعه مشخص شده است.

1.راحتی در استفاده

قابلیت استفاده ، معیاری برای استفاده از یک محصول برای انجام کارهای تعیین شده آسان است. این با مفاهیم مرتبط با سودمندی و دوست داشتن متمایز است.

2.کشف در مقابل یادگیری در مقابل کارآیی

جنبه های زیادی برای قابلیت استفاده وجود دارد ، اما به طور سنتی این اصطلاح به طور خاص به ویژگی های کشف ، یادگیری و کارآیی اشاره دارد.

1. کشف شامل جستجوی و پیدا کردن ویژگی محصول در پاسخ به نیاز خاص است. آزمایش قابلیت استفاده می تواند تعیین کند که کاربر برای یافتن یک ویژگی و چه تعداد خطا (گزینه های اشتباه در مورد مکان) کاربر را در طول راه ایجاد می کند.
2. یادگیری به روندی است که توسط آن کاربر نحوه استفاده از یک ویژگی کشف شده برای انجام یک کار را می فهمد. آزمایش قابلیت استفاده می تواند تعیین کند که این فرایند چه مدت طول می کشد و همچنین چند خطا را کاربر هنگام یادگیری ویژگی ایجاد می کند.
3. کارآیی به نکاتی اطلاق می شود که کاربر ویژگی "تسلط" آن را داشته و بدون نیاز به یادگیری بیشتر از آن استفاده می کند. آزمایش قابلیت استفاده می تواند تعیین کند چه مدت طول می کشد تا کاربر با تجربه مراحل لازم برای استفاده از ویژگی را انجام دهد.

این سه جنبه اساسی از قابلیت استفاده ، به شدت تحت تأثیر ماهیت کار مورد نظر و فرکانس عملکرد کاربر قرار می گیرد. بعضی از ویژگی ها به ندرت مورد استفاده قرار می گیرند و یا آنقدر پیچیده هستند که اساساً کاربر هر بار آنها را از یاد می برد.

5.**تغییر پذیری**

قابلیت تغییر کم هزینه

لیست غیر جامع شامل زیر برخی از شیوه ها و اصول کم تلاش است که در بسیاری از پروژه ها با آن روبرو شده ام. در پروژه های خود باید از همه جوانب استفاده کنید مگر اینکه دلیل خیلی خوبی نداشته باشید. توجه به آنها قطعاً در آینده پرداخت خواهد شد!

1. پیکربندی: بیشتر سیستم ها نوعی گزینه پیکربندی دارند. حتی یک اسکریپت کوچک دارای گزینه های پیکربندی است. به خصوص برای اسکریپت های کوچک و نمونه های اولیه پرتاب که به طور ناگهانی رشد می کنند ، پیکربندی اغلب به سختی در یک محیط خاص کدگذاری می شود و تغییر بعد PITA است. بیشتر چارچوبها روشهای ساده ای برای ارائه گزینه های پیکربندی فراهم می کنند ، از آنها استفاده می کنند اعداد و رشته های جادویی: اکنون به طور جدی ، ثابت ها به راحتی می توانند تعریف شوند. فقط این کار را انجام دهید دیر یا زود شما مجبور به پالایشگاه خواهید شد.
2. نما و رابط ها: پیچیدگی را پنهان کرده و اطلاعات ارائه شده را محدود کنید. اگر می خواهید یک رابط را از یک کلاس که در کل برنامه استفاده می شود استخراج کنید ، به احتمال زیاد موارد استفاده از روش هایی را که نمی خواهید در رابط استخراج شده پیدا کنید ، پیدا خواهید کرد. تعریف و استفاده از رابط فقط چند دقیقه طول می کشد. البته یک رابط خوب مدتی طول می کشد ، اما شروع با رابط این امکان را به شما می دهد که بعداً به طور صحیح رفیکتور شوید.
3. نامگذاری و مستندات: فقط وقت خود را صرف کنید تا به درستی مواردی را که می نویسید اسم گذاری و مستند سازی کنید. تلاش زیادی نیست در حالی که شما تمام کد را در ذهن خود دارید ، اما جرات نکنید که ماه ها بعد به کد غیرمجاز خود برگردید. و به یاد داشته باشید ، نامگذاری یکی از سخت ترین کارهاست!
4. مدیریت وابستگی: هر زبان برنامه نویسی مدرن دارای نوعی سیستم برای مدیریت وابستگی ها مانند کتابخانه های مورد نیاز است. بیاموزید که از آن برنامه های مدیریت وابستگی استفاده کنید. شروع به کج کردن دستی نکنید.
5. چارچوب: چرخ را دوباره اختراع نکنید. اگر با استفاده از چارچوب کد کمتری می نویسید ، در صورت نیاز باید کد کمتری را تغییر دهید.

قابلیت تغییر قیمت بالا

هنگام تعریف معماری سیستم ، باید تصمیمات مهم زیادی بگیرید. همچنین برخی از شیوه ها و اصولی وجود دارد که می توانید برای بدست آوردن درجه تغییرپذیری بالایی در سیستم خود استفاده کنید. با این حال ، اطمینان حاصل کنید که بازده سرمایه گذاری را برای این تصمیمات کاملاً تخمین بزنید!

1. طراحی دامنه محور: هنر ساختن سیستم های عظیم و پیچیده ای است که به راحتی قابل تغییر است. DDD بر الگوبرداری مناسب از اشخاص دامنه واقعی و تعامل تأکید دارد. سپس تغییرات در دامنه می توانند به زیر ساخت های پشتیبانی کمک کنند.
2. نقشه برداران رابطه ای شیء: کتابخانه های ORM سطح بالایی از انتزاع نسبت به پایگاه داده شما را ارائه می دهند. طرحواره و سؤالات شما به راحتی تغییر می یابد یا حتی به صورت خودکار به سیستم

عامل های دیگر منتقل نمی شود. با این حال ، ORM ها همچنین دارای اشکالاتی هستند که برای به حداکثر رساندن پتانسیل آنها باید به دقت مورد بررسی قرار گیرد.

3. پیام رسانی: سیستم های توزیع شده می توانند الگوهای مختلف پیام رسانی مختلفی داشته باشند. در حالی که استفاده از الگوی پیام رسانی می تواند تغییر پذیری و استحکام یک سیستم را تا حد زیادی بهبود بخشد ، همچنین بسیاری از موضوعاتی را که باید به آنها پرداخته شود از قبیل سازگاری ، در دسترس بودن و تحمل پارتیشن بندی (نظریه CAP) باز می کند.