بسمه تعالی

خلاصه‌ای از کتاب

Extreme Programming Explained: Embrace Change, 2nd Edition
Kent Beck, Cynthia Andres
Addison Wesley Professional
Nov 2004
240 Pages

این خلاصه با توجه به اولویتهای شرکت مشاورین صنعت نرم‌افزاری اعوان تهیه شده است و در نتیجه خلاصه کاملی از کتاب نیست.

۱۳۸٤/۸/۱۸

فصل ۱ XP چیست؟

- دید واقع‌گرایانه این است که تیم تولید نرم‌افزار از انسانها تشکیل شده است با تمام خصوصیاتشان نه از ماشینها. بنابراین فرآیندهای مهندسی نرم‌افزار نمی‌توانند شبیه الگوریتمهایی که برای ماشینها نوشته شده‌اند اجرا شوند.
- نگران deadline و محدودیت سایر منابع نباشید، زیرا این نگرانی اجازه نمی‌دهد تمام تلاشتان را بکنید. وقتی انسان احساس کند منابع کافی در اختیار دارد بهتر عمل می‌کند.
- نگران انتظارات اشتباه دیگران نباشید. این وظیفه آنهاست که انتظارات خود را مدیریت کنند نه شما.

فصل ۲ یادگیری رانندگی

- تولید نرم‌افزار شبیه رانندگی است. باید تطبیق‌پذیر باشد. افراد تیم باید آگاه (حواس جمع) باشند.

فصل ۳ ارزشها، اصول، و عملکردها

- تعریف عملکرد[1] (تکنیکی که قابل مشاهده است و کار بوسیله آن انجام می‌شود)، ارزش[2] (اهدافی که عملکردها قرار است به آنها برسند)، و اصل[3] (پلی که در یک context ارزشها را به عملکردها متصل می‌کند)
- ارزشها کلی هستند در حالیکه اصول و عملکردها وابسته به محیط هستند.

فصل ۴ ارزشها

[1] Practice
[2] Value
[3] Principle

- اشکال کار در چیزهایی نیست که افراد نمی‌دانند، اشکال در چیزهایی است که افراد می‌دانند ولی اشتباه است.
- بیشتر افراد فکر می‌کنند که در مهندسی نرم‌افزار عملکرد شخصی افراد و کارهایی که انجام می‌شود اهمیت دارد، درحالیکه در واقع نحوه عملکرد افراد در تیم و سازمان است که اهمیت دارد.
- ارزشهایی که XP بر روی آنها بنا شده: ارتباطات، سادگی، feedback، شجاعت، و احترام.
- سادگی وابسته به context است.
- feedback: هرچه زودتر یک مشکل را بدانید، حل آن آسانتر است.
- احترام هر یک از اعضای تیم مستقل از تخصص و نقشی که دارند، به عنوان یک انسان، باید حفظ شود.
- هر تیمی می‌تواند برای خود ارزشهایی قرار دهد و تیم را با آنها تطبیق دهد.

فصل ۵ اصول

انسانیت

- چون انسانها نرم‌افزار را ایجاد می‌کنند، نمی‌توان فرآیند را مثل اجرای نرم‌افزار (خشک و بی خلاقیت) تعریف کرد. در غیر این صورت کار گره می‌خورد و به منافع مالی سازمان نیز لطمه وارد می‌شود. بنابراین، باید با اعضای تیم انسانی برخورد شود. انسانها برای درست کار کردن این نیازها را دارند:
  ۱. **امنیت اولیه:** مثل امنیت جانی، آبرویی، و شغلی و همچنین تامین مالی و...
  ۲. **موفقیت:** داشتن موقعیت و توانایی مفید بودن در اجتماع
  ۳. **تعلق:** حساب شدن جزء تیمی که اعتبارسنجی کارشان و قابلیت حساب کردن روی دیگران را به آنها بدهد و در راه هدف مشترک همکاری کنند.
  ٤. **رشد:** داشتن موقعیت رشد و گسترش دید
  ۵. **صمیمیت:** قابلیت درک کردن دیگران و درک شدن توسط دیگران
- انسانها نیازهای دیگری هم دارند مثل: استراحت و ورزش. زمانهایی که مشغول کار نیستند به اینگونه مسائل می‌رسند و با انرژی به تیم بر می‌گردند. بنابراین بهتر است ساعت کاری محدود باشد.
- اگر اهداف افراد و تیم در یک راستا باشد افراد برای موفقیت تیم از خود گذشتگی نشان می‌دهند.

اقتصاد

- برای هر کاری باید پول پرداخت شود. به عبارت دیگر باید کارها ارزش تجاری داشته باشد.
- انجام زودترکارهای با اولویت تجاری بالاتر ارزش (سوددهی) پروژه را حداکثر می‌کند.
- دو اصل اقتصادی در نرم‌افزار:
  ۱. **ارزش پول در زمان:** از دو مقدار برابر پول در امروز و فردا، پول امروز ارزش بیشتری دارد:
     - تولید نرم‌افزار وقتی پول را زودتر بدست آورد و دیرتر خرج کند سودمندتر است.
     - طراحی افزایشی هزینه طراحی را در آخرین زمان ممکن خرج می‌کند. (در این زمان نسبت به زمانهای قبلی طراح از نظر شناخت نرم‌افزار در بهترین وضعیت به سر می‌برد.)
     - پول گرفتن به ازای ویژگیهای تحویل داده‌شده نیز در مواقع لزوم مفید است.

۲. **ارزش انتخاب سیستم یا تیم**: آینده نگری در انجام پروژه. مثلا طوری پروژه انجام شود که بتوان آنرا در چند جای دیگر نیز استفاده کرد. البته این کار باید حساب شده انجام شود نه دیمی.

سود دو (همه) جانبه

هر عملی باید به نفع تمام طرفهای مرتبط باشد و همه این را احساس کنند. و گرنه روابط خراب می‌شود. به همین دلیل باید روشی که برای حل مشکلی در نظر گرفته می‌شود فقط به آینده نگاه نکند به طوریکه در حال حاضر به شکل ضرر به نظر برسد (مثل مستندسازی مفصل به منظور راحتتر کردن نگهداری) بلکه باید هم در حال حاضر برای ما سودمند باشد هم در آینده، و هم برای مشتری (مثل استفاده از تولید مبتنی بر آزمون و فاکتورگیری مجدد).

تشابه با خود

وقتی یک راه حل یا ساختار در یک جا جواب می‌دهد، ممکن است در جاهای دیگر با مقیاس متفاوت نیز جواب دهد. حداقل به عنوان یک نقطه شروع مناسب است. مثال از کتاب:

For example, the basic rhythm of development is that you write a test that fails and then you make it work. The rhythm operates at all different scales. In a quarter, you list the themes you want to address and then you address them with stories. In a week, you list the stories you want to address, write tests expressing the stories, then make them work. In a few hours, you list the tests you know you need to write, then write a test, make it work, write another test, and make them both work until the list is done.

(برای مثال از دنیای واقعی به متن کتاب مراجعه شود.)

بهبود

- در مهندسی نرم‌افزار perfect صفت نیست، فعل است. هیچ چیز کاملی نمی‌توان یافت (فرآیند، طراحی، کد) اما همواره می‌توان کار را کاملتر کرد.
- برای شروع یک کار لازم نیست کاملترین تخصص وجود داشته باشد، بلکه می‌توان شروع کرد و با گذشت زمان کار را بهتر کرد.

تنوع

- تیمهایی که همه یک جور هستند، با وجود راحتی مؤثر نیستند. در تیم باید تخصصها، سلایق، و دیدهای متفاوتی وجود داشته باشد تا مشکلات و خطرات را بهتر ببینند، راه حلهای متنوعی به ذهنشان برسد، ...
- لازمه تنوع برخورد آرا است. البته نباید در تیم تضاد و تنفر باشد.
- دو راه حل طراحی متفاوت یک موقعیت است نه یک مشکل.

انعکاس (خودنگری)

- تیمهای خوب فقط سرشان به کارشان نیست، بلکه به این سوالات نیز فکر می‌کنند: چگونه دارند کار می‌کنند؟ چرا دارند کار می‌کنند؟ این تیمها موفقیت و شکست را تحلیل می‌کنند. آنها سعی نمی‌کنند خطاهایشان را مخفی کنند بلکه آنها را کشف می‌کنند و از آنها درس می‌گیرند.
- اینکار می‌تواند به صورت رسمی یا غیر رسمی (مثلا در زمان نهار) انجام شود.

- در اینکار نباید افراط کرد. نباید زمان کار کردن کلا صرف بهبود روش کار کردن شود.

موقعیت (شانس)

مشکلات را باید به صورت موقعیتهایی برای بهبود نگاه کرد یا آنها را به موقعیت تبدیل کرد.

تکرار

در جاهایی که ارزشش را دارد برخی فعالیتها باید تکرار یکدیگر باشند. مثلا چون خطا در نرم‌افزار جدا برای پروژه دردسر درست می‌کند (اعتماد مشتری را از بین می‌برد و کار گره می‌خورد ...)، از راههای مختلفی به مبارزه با آن می‌پردازیم: برنامه‌ریزی دونفره، تجمیع مداوم و ... اگر یکی نتوانست یک خطا را کشف کند ممکن است دیگری بتواند. پس این هزینه اضافه را می‌دهیم که از حاصل آن استفاده کنیم.

شکست

بعضی وقتها قبول شکست به منزله بدست آوردن دانش است. اگر واقعا نمی‌دانیم از بین چند راه حل ممکن کدام بهتر است یک راه حل این است که همه را پیاده‌سازی کنیم. در برخی موارد اینکار سریعتر از فکر کردن و بحث کردن تیم طراحی جواب می‌دهد.

کیفیت

- [بر خلاف تصور رایج] بالابردن استاندارد کیفیت مورد نیاز موجب کندی پروژه نیست بلکه برعکس است. هر بهبودی در کیفیت سایر معیارهای پروژه را بهتر می‌کند (هزینه، زمان، قابلیت پیش‌بینی و ...). [معمولا در پایان پروژه، در اثر نادیده گرفتن کیفیت، باید هزینه و زمانی بیش از هزینه و زمان صرفه جویی شده پرداخت (به علت سخت شدن تغییرات و مشکلات معماری و ...)]
- کیفیت فقط یک مطلب تجاری نیست، افراد نیازمند این هستند که از کاری که می‌کنند خوششان بیاید (احساس خوب کار کردن).
- کیفیت وسیله‌ای برای کنترل پروژه نیست در XP به جای آن از محدوده[4] استفاده می‌شود.

قدمهای کوچک (بچه‌گانه)

انجام دادن تغییرات بزرگ در یک گام همواره وسوسه کننده‌اند، اما اینکار مضر است زیرا با وجود اینکه تغییرات کوچک ممکن است احساس کندی ایجاد کنند، در مجموع بهتر از انجام تغییرات بزرگ ناموفق و عقبگرد است.

مسئولیت پذیرفته شده (نه تفویض شده)

- مسئولیت قابل تخصیص دادن نیست بلکه پذیرفتنی است.
- همراه مسئولیت اختیار نیز باید باشد.

[4] Scope

- تخصیص اشتباه دردسرساز است مثلا اگر کسی مسئول کاری است ولی خودش دانش لازم را ندارد و قرار است با راهنمایی دیگری کار را انجام دهد و دیگری در مسئولیت شریک او نیست، کار با مشکل مواجه می-شود. هیچ یک از این دو نفر در جایگاه درست قرار نگرفته‌اند (کار با یکی است، مسئولیت با دیگری).

فصل ۶ عملکردها

- عملکردهای XP برای دستیابی به ارزشها مفیدند و به خودی خود چیزهای مفیدی نیستند. باید بدانید که با استفاده از یک عملکرد به چه ارزشی می‌خواهیم برسیم وگرنه استفاده کورکورانه از عملکردها مفید و موفقیت آمیز نیست. مفید بودن عملکردها به دامنه نیز وابسته است.
- عملکردها را یکی یکی پیاده کنید.
- عملکردها وقتی با هم مورد استفاده باشند موثرتر هستند.

فصل ۷ عملکردهای اولیه (اصلی)

باهم بنشینید

همه تیم در یک اتاق که به اندازه کافی جا دارد کار کنند تا به اندازه کافی باهم ارتباط داشته باشند. البته باید در صورت لزوم به نیاز برخی افراد به یک جای خصوصی توجه شود. مثلا با پارتیشن جای کسی جدا شود...

تیم کامل

- در تیم باید تمام قابلیتها و زاویه دیدهای مورد نیاز وجود داشته باشد. افراد تیم باید احساس تیم بودن، باهم بودن، و پشتیبانی از کار، رشد، و یادگیری یکدیگر داشته باشند.
- اگر به میزان قابل توجهی به مهارتی نیاز است که در تیم موجود نیست باید شخصی با این مهارت به تیم بیاید. و اگر نیاز به مهارت شخصی تمام می‌شود باید از تیم برود.
- دو عدد جالب:
    - ۱۲ تعداد افرادی است که انسان در یک روز می‌تواند با راحتی با آنها ارتباط برقرار کند.
    - وقتی۱۵۰ در تیم باشند انسان دیگر چهره‌های آنها را نمی‌شناسد.

    اگر این دو مرز رد شوند، اعتماد کردن سخت می‌شود، درحالیکه اعتماد لازمه همکاری است.

اطلاع رسان بودن محل کار

محیط فیزیکی که تیم در آن مشغول است باید نشان دهد که چه خبر است. مثلا روی دیوار بردی وجود داشته باشد که "داستانهای کاربر" روی آن به صورت طبقه‌بندی شده نصب شده باشند. در این صورت اگر چند وقت هیچ داستانی به قسمت "انجام شده" اضافه نشود معلوم است که مشکل وجود دارد.

کارکردن با انرژی

- فقط به اندازه‌ای کار کنید که می‌توانید مفید کار کنید. اگر شخصی در یک روز خود را بکشد و دو روز بعد از آن خسته باشد به ضرر پروژه است.

- وقتی مریض می‌شوید، استراحت کنید تا زودتر خوب شوید. این بهترین خدمت و احترام به خود و تیم است.

برنامه نویسی دونفره

- همه برنامه‌های نهایی را به صورت دو نفر روی یک دستگاه بنویسید.
- دستگاه را طوری قرار دهید که هر دو به اندازه کافی جا داشته باشند. همچنین موش و صفحه کلید را با حرکت دادن به یکدیگر دهید.
- یکدیگر را مشغول به کار نگهدارید
- برای پالایش در سیستم یورش ذهنی[5] کنید.
- ایده‌ها را واضح کنید.
- وقتی همکارتان گیر می‌کند، ابتکار عمل را در دست بگیرید تا از حس بدی که "شما نمی‌توانید وضعیت را تغییر دهید" خلاص شوید یا از گیر نجات پیدا کنید. (شک بین این دو جمله از من است نه کتاب.)
- یکدیگر را به عملکردهای تیم مسئول نگهدارید.
- برنامه‌نویسی دو نفره به این معنی نیست که هیچ کس حق ندارد هیچ کاری را تنها انجام دهد. افراد می‌توانند تنها روی ایده‌ها فکر کنند، نمونه‌سازی کنند، و ... اما وقتی قرار است ایده‌ای پیاده‌سازی نهایی شود، باید نمونه ساخته شده کنار گذاشته شود و برنامه‌نویسی دو نفره انجام شود.
- در برنامه‌نویسی دو نفره نکاتی باید رعایت شوند:
  - افراد به فضاهای متفاوتی نیازمندند تا راحت کار کنند.
  - بهداشت و نظافت شخصی افراد مهم است. مثلا وقتی کسی بیمار باشد ممکن است سرایت کند. هنگام عطسه جلوی دهان را باید گرفت و...
  - از عطرهای شدید باید پرهیز کرد.

  اگر این نکات و نکات مشابهی که راحتی افراد را در هنگام برنامه‌نویسی دونفره تضمین می‌کنند، رعابت نشوند، افراد نمی‌توانند درست فکر کنند. اگر شخصی در برنامه‌نویسی دونفره با شخصی راحت نیست، در حقیقت به کار تیم لطمه می‌خورد، پس باید آنرا با مدیر در میان بگذارد.

داستانها

- کلمه requirement با توجه به اینکه طبق فرهنگ لغات معنی اجباری دارد، گمراه کننده است و باعث مطلق‌گرایی می‌شود. در واقع ۱۰ یا ۲۰ درصد یا حتی ۵ درصد از requirementهای گفته شده برای دستیابی به مزایای حرفه‌ای مورد انتظار از سیستم کافی هستند. بنابراین بقیه اختیاری هستند نه اجباری.
- در XP به هر عملکرد قابل مشاهده توسط کاربر توسط یک داستان که روی یک کارت نوشته می‌شود، بیان می‌شود. این کارتها روی دیوار نصب می‌شوند. روی کارت تخمین زمانی پیاده‌سازی نوشته می‌شود. داستانها یک نام دارند و یک شکل گرافیکی یا توضیح متنی معمولی عملکرد را نشان می‌دهد. به نظر نویسنده یک کارت دست نویس از کامپیوتری ارزشمندتر است.
- این نکته که تخمین زمانی روی کارت نوشته می‌شود، کمک می‌کند با کاربر درست به توافق برسیم:

[5] Brainstorm

This gets everyone thinking about how to get the greatest return from the smallest investment. If someone asks me whether I want the Ferrari or the minivan, I choose the Ferrari. It will inevitably be more fun. However, as soon as someone says, "Do you want the Ferrari for $150,000 or the minivan for $25,000?" I can begin to make an informed decision. Adding new constraints like "I need to haul five children" or "It has to go 150 miles per hour" clear the picture further. There are cases where either decision makes sense. You can't make a good decision based on image alone. To choose a car wisely you need to know your constraints, both cost and intended use. All other things being equal, appeal comes into play.

دورهای هفتگی

۱. در اول هر هفته یک جلسه داشته باشید که در آن این مسائل را مرور کنید:
   - پیشرفت تا به امروز و پیشرفت هفته گذشته نسبت به کار مورد انتظار در آن.
   - مشتری را مجبور کنید که به اندازه پیاده‌سازی در یک هفته داستان انتخاب کند.
   - داستانها را به کارها بشکنید. اعضای تیم برای کارها ثبت‌نام می‌کنند و آنها را تخمین می‌زنند.
۲. سپس آزمونهایی نوشته می‌شوند که در صورت کامل شدن داستانها جوابشان مثبت خواهد بود.
۳. پیاده‌سازی داستانها انجام می‌شود. البته یک تیم باانگیزه پیاده‌سازی را کامل انجام دهد و فقط به رد کردن آزمونها بسنده نمی‌کند تا در پایان هفته برنامه قابل استقرار داشته باشد.

- در این روش مالکیت یک کار به یک شخص داده می‌شود و این تاثیر خوبی دارد.
- روش تقسیم کارها نیز می‌تواند با خواست خود افراد یا با استفاده از یک stack باشد.
- اگر کاری بدلیل تخصص باید به کس دیگری داده شود ولی به دلیل مکانیزم تقسیم کار به دیگری داده شده، با استفاده از برنامه‌نویسی دونفره مشکل قابل حل است.

دورهای فصلی

- هر بار یک فصل کار کنید، سپس توجه را روی تیم، پروژه، پیشرفت، و قرارداشتن در مسیر اهداف بزرگتر متمرکز کنید.
- در برنامه‌ریزی فصلی روی این موارد تمرکز کنید:
  - مشخص کردن گلوگاهها، مخصوصا آنهایی که از بیرون تیم نشأت می‌گیرند.
  - برنامه‌ریزی برای تعمیرات (بهبود مشکلات)
  - انتخاب موضوع[6] یا موضوعات فصل.
  - انتخاب داستانهایی برای موضوعات منتخب به اندازه یک فصل.
  - تصویرکلی: نحوه قرارگیری پروژه نسبت به سازمان

راحتتر کردن (کمی شل کردن)

- از آنجا که رسیدن به قرارها و زمان تحویلها از نظر روحیه تیمی و اعتماد مشتری اهمیت زیادی دارد، باید سعی شود با دید واقع گرایانه از تعریف کار زیادی برای یک محدوده زمانی خودداری شود.

[6] Theme

- می‌توان کارهایی را به صورت انجام در صورت اضافه آوردن وقت در برنامه قرار داد.
- باید با سهامداران شفاف صحبت کرد و تا جای ممکن کار اضافی و فشرده نپذیرفت.
- در صورتیکه سازمان آمادگی برای پذیرش حرف صادقانه و شفاف را در این زمینه ندارد، خود شخص اینکار را برای خودش بکند و به خود فشار عصبی وارد نکند.

ساختن‌های ۱۰ دقیقه‌ای

- زمان ساختن[7] (که شامل اجرای خودکار آزمونها نیز می‌شود) نباید بیشتر از ۱۰ دقیقه طول بکشد زیرا افراد از آن پرهیز خواهند کرد. همچنین نباید کمتر طول بکشد تا وقت برای قهوه خوردن باشد. (این موارد ظاهرا با پروژه‌های ما مناسبتی ندارد. به نظر می‌رسد ساختن اینا با ما کمی متفاوت است! شاید هم به دلیل آزمون باشد.)
- البته در هر ساختنی لزوما آزمونها اجرا نمی‌شود یا همه چیز دوباره ساخته نمی‌شود. برای اینکه دقیقا چه روالی اجرا شود، در هر پروژه باید تصمیم گیری شود.
- نساختن یا اجرا نکردن برخی آزمونها بعد از یک تغییر این مخاطره را دارد که برخی خطاها بعد از تغییر کشف نشوند.
- ساختنهای خودکار به کاهش استرس نیز کمک می‌کنند زیرا با هر تغییری برنامه‌نویس نتیجه را می‌بیند.

تجمیع دائمی

- مشکلات تجمیع قابل پیش‌بینی نیست. تغییرات را در سیستم تجمیع کنید و بیازمایید پیش از آنکه چند ساعت از تغییر بگذرد.
- همواره یک محصول کامل (قابل استقرار) بسازید. منظور این نیست که همه نیازمندیها پیاده‌سازی شوند، بلکه منظور این است که قسمتهای پیاده‌سازی شده به شکل نهایی و قابل استقرار باشند. در این صورت آخر کار مشکل کمتری خواهید داشت.
- با توجه به مطالب این قسمت من این انواع را برای تجمیع درک کردم (این مطالب برداشت من است و ممکن است انواع دیگری نیز وجود داشته باشد):
  ۱. دیرتر از روزانه: هر چند روز (مثلا ۷ روز) سیستم تجمیع، ساخته، و آزمون شود.
  ۲. روزانه: هر روز پس از پایان کار سیستم تجمیع، ساخته، و آزمون شود.
  ۳. آسنکرون: پس از انجام تغییرات check in انجام می‌شود و سیستم ساخت، متوجه تغییرات شده و خطاها را به طریقی (مثلا email) اطلاع می‌دهد. بعد از check in برنامه‌نویس مشغول کار دیگری می‌شود. و بعد که email چک کرد خطاها را اصلاح می‌کند.
  ٤. سنکرون: پس از اعمال تغییر ساختن انجام می‌شود و feedback گرفته می‌شود. در حین ساخت دو نفر برنامه‌نویس می‌توانند کمی به کاری که کرده‌اند فکر کنند و ...

[7] Build

روش ما شبیه سنکرون است و در مجموع مناسب به نظر می‌رسد. اما با توجه به اینکه روال مشخصی نداریم، شاید در برخی موارد که ساختن لازم نیست ساختن انجام شود و برعکس. اگر بتوانیم روال مشخصی درآوریم با جلوگیری از ساختنهای اضافه در وقت صرفه‌جویی می‌شود و با انجام ساختهای مناسب کیفیت بالا می‌رود.

برنامه نویسی آزمون–اول

- قبل از پیاده‌سازی آزمون بنویسید. اینکار چند خاصیت دارد:

۱. انحراف از محدوده[8]: معمولا در برنامه‌نویسی پیش می‌آید که برنامه‌نویس از خودش چیزهایی اضافه کند: مثلا برای فلان حالت این شرط را می‌گذارم. با نوشتن آزمونها برنامه‌نویس ابتدا به دقت مشخص می‌کند که چه می‌خواهد بکند. در صورتیکه آن کارهای اضافه را هم بخواهد انجام دهد می‌تواند برای آنها نیز آزمون بنویسد.

۲. جفت شدگی و یکدستی[9]: اگر نوشتن آزمون سخت است احتمالا اشکال از طراحی است. آزمون برنامه loosely coupled و highly cohesive آسان است.

۳. اعتماد: از طریق اینکار برنامه‌نویس برنامه تمیز و درستی تحویل می‌دهد و در تیم اعتماد بدست می‌آورد.

٤. ریتم: برنامه‌نویسی را تحت ریتم منظمی در می‌آورد. در بسیاری از مواقع برنامه‌نویس گم می‌شود و معلوم نیست چه می‌کند. اما با این روش واضح است که چه کاری انجام می‌شود یا مشغول درست کردن یک خطای مشخص است یا اضافه کردن یک آزمون و خاصیت جدید. برنامه‌نویسی می‌تواند چنین روندی داشته باشد: آزمون – کد – refactor.

طراحی افزایشی

- در مهندسی نرم‌افزار داده‌هایی ارائه شده (توسط Boehm) که هرچه زمان می‌گذرد تغییرات دشوارتر می‌شوند. این مسئله امروزه صحت ندارد، یا حداقل تیمهای XP می‌توانند طوری کار کنند که اینطور نباشد. این به معنی به حداقل رساندن طراحی نیست بلکه به معنی تغییر زمان طراحی نسبت به روشهای قدیمی است.
- هر روز روی طراحی سرمایه‌گذاری کنید اینکار کمک می‌کند که نیاز تیم به حس کردن موفقیت (پس از برنامه‌نویسی) برآورده شود و پرداخت هزینه طراحی تا آخرین لحظه ممکن به تعویق انداخته شود. طراحی شما باید برای محصولی که تا به امروز ساخته‌اید مناسب باشد نه برای فردا. هرچه زمان طراحی و پیاده‌سازی نزدیکتر باشند مفیدتر است.
- در حین پروژه دائما باید طراحی بهبود داده شود. مثلا قسمتهای تکراری باید در یک جا فاکتورگیری شوند.
- اگر تغییرات قدم به قدم را یاد بگیریم این کارها عملی است.

در مورد این ایده باید محتاط بود چرا که در جامعه مهندسی نرم‌افزار مخصوصا معماری نرم‌افزار حرفهای دیگری زده می‌شود. نظر من این است که این ایده برای سیستم‌هایی که ما می‌نویسیم تقریبا مناسب است.

---

[8] Scope creep
[9] Coupling and Cohesion

فصل ۸ شروع

- تغییرات را کم کم اعمال کنید تا از جائیکه هستید به جای مناسب بروید.
- برای هر شخص، پروژه، و سازمان یک سری عملکردها و با شکلهای مختلف مفید است.
- تغییرات با چشم و گوش باز انجام دهید: بدانید چرا از یک عملکرد استفاده می‌کنید. در صورت امکان معیاری برای مقایسه با قبل داشته باشید...
- از خودتان شروع کنید و تغییرات را به دیگران تحمیل نکنید. اینکار اعتماد را از بین می‌برد.

فصل ۹ عملکردهای ثانویه (فرعی[۱۰])

عملکردهای توضیح داده شده در این فصل احتمالا درصورتی مناسب هستند که عملکردهای اصلی (فصل ۷) قبلا به کار گرفته شده باشند وگرنه ممکن است خطرناک باشند. البته تشخیص این مطلب به عهده شماست.

واقعا درگیر بودن مشتری

کسانی که زندگی و کارشان به سیستم مرتبط است را جزء تیم قرار دهید (در جلسات هفتگی و فصلی شرکت دهید.) این کار تاثیرات احتمالی زیر را دارد:

- (از خودم:) تشخیص خطاهای نیازمندیها و تضمین کیفیت
- دریافت بودجه با مشاهده پیشرفت کار
- ارتباط مستقیم در خوشحال نگه داشتن مشتری موثرتر است تا بی ارتباط بودن یا ارتباط با نماینده مشتری. ارتباط با نماینده مشتری ممکن است باعث ایجاد ویژگیهای بی‌استفاده در سیستم، نوشتن آزمونهای غیر مورد نیاز در آزمون پذیرش، و ... شود زیرا ممکن است نظر او با مشتری تفاوتهایی داشته باشد.

نویسنده می‌گوید به این موضوع این اعتراضات می‌شود:

- سیستم مختص این مشتری خواهد بود و عمومی کردن آن ممکن نخواهد بود. جواب: تضمین عمومیت محصول با اعضای بازاریابی تیم است. به طور کلی ارزش محصول با درگیر بودن مشتری بالا می‌رود (من این مورد را خوب نمی‌فهمم و با اعتراض انجام شده موافقم.)
- تشبیه به کارخانه سس: اگر مشتری بداند در تیم نرم‌افزار چه خبر است هرگز به ما اعتماد نمی‌کند. جواب: آیا مطمئنید الآن اعتماد دارد؟ مشتری تا بحال با نرم‌افزار کار کرده است و می‌داند چه خبر است اگر نداند هم خواهد فهمید! به هر حال شفاف عمل کردن از پنهان کاری سودمندتر است و وقتی شما در حال پنهان کردن چیزی نیستید زمان هم صرف آن نمی‌کنید.

(نظر من:) می‌توان دو جور جلسه داشت بامشتری و بی‌مشتری. و بهتر است از اول همه چیز شفاف باشد. اگر کاری را در یک مدت معین نمی‌شود کرد، مشتری الآن بداند بهتر است تا در پایان زمان. تشخیص این نقاط کور به عهده مدیر پروژه است.

استقرار افزایشی

[10] Corollary

در انتقال از سیستم موجود به سیستم جدید باید افزایشی عمل کرد و انتقال یکباره امکان‌پذیر نیست و اگر باشد به هزینه‌اش نمی‌ارزد.

ادامه دادن تیم

یک تیم موفق باز هم با هم کار کنند (برای شرکتهای بزرگ که چند تیم دارند) و با افراد به صورت یک شیء (واحد برنامه‌نویسی قابل اتصال) رفتار نشود زیرا این در مجموع به ضرر سازمان است. اعضای تیمهای موفق نسبت به یکدیگر اعتماد کسب کرده‌اند.

کوچک کردن[11] تیم

این مورد هم بیشتر برای شرکتهای بزرگ است: وقتی کار جا می‌افتد، با ثابت نگهداشتن بارکاری می‌توان تیم را کوچک کرد و افراد بیرون رفته از تیم تیمهای جدیدی تشکیل می‌دهند. اگر تیمها خیلی کوچک شدند، تیمهای خیلی کوچک را می‌توان با هم پیوند داد (تویوتا اینجوری کار می‌کند.)

تحلیل ریشه‌یابی

هر وقت خطایی بعد از پیاده‌سازی پیدا می‌شود، خطا و علت آنرا باهم از بین ببرید. منظور این نیست که این خطای خاص دوباره برنگردد، منظور این است که تیم دیگر چنین اشتباهی نکند.

در XP عکس‌العمل نسبت به یک خطا چنین است:

۱. آزمون خودکار در سطح سیستمی برای نمایش خطا نوشته شود.
۲. یک آزمون واحد با حداقل محدوده، که خطا را نشان می‌دهد نیز نوشته شود.
۳. سیستم را اصلاح کنید تا آزمون واحد رد شود. در این صورت باید آزمون سیستم نیز رد شود، در غیر این مورد به ۲ برگردید.
٤. بعد از اصلاح خطا تحلیل کنید که چرا خطا ایجاد شده بود و کشف نشده بود! تغییرات لازم برای پیشگیری از تکرار این مشکل در آینده به عمل بیایند.

مطالعه این قسمت از کتاب مناسب است:

Taiichi Ohno has a simple exercise for this last step, the Five Whys. Ask five times why a problem occurred. So, for example,

1. Why did we miss this defect? Because we didn't know the balance could be negative overnight.
2. Why didn't we know? Because only Mrs. Crosby knows and she isn't part of the team.
3. Why isn't she part of the team? Because she is still supporting the old system and no one else knows how.
4. Why doesn't anyone else know how? Because it isn't a management priority to teach anyone.

[11] Shrinking

5. Why isn't it a management priority? Because they didn't know that a $20,000 investment could have saved us $500,000.

After Five Whys, you find the people problem lying at the heart of the defect (and it's almost always a people problem). Addressing that problem and the other problems encountered along the way will give you some reassurance that you won't ever have to deal with this particular mistake again.

کد اشتراکی

هر عضو تیم می‌تواند در هر زمان هر قسمت از کد را که می‌خواهد بهبود بخشد. این کار وقتی قابل تحقق است که تیم به حد قابل قبولی از مسئولیت‌پذیری رسیده باشد و افراد نباید کد را بدون توجه به تاثیر روی کار سایرین و سایر قسمتهای سیستم تغییر دهند.

کدنویسی و آزمون

- فقط کد و آزمون را به عنوان فرآورده‌های دائمی نگهداری کنید. سایر مستندات را با استفاده از کد تولید کنید. مشتری بابت کاری که سیستم می‌کند یا خواهد کرد پول می‌دهد و مستنداتی که به این موارد کمک می‌کنند باارزشند و سایر مستندات اتلاف منابع هستند.
- این عملکرد را بصورت افزایشی با پیشرفت تیم می‌توان جا انداخت. هر چه در طراحی افزایشی موفقتر باشیم مستندات (تصمیمات) طراحی در ابتدای کار سبکتر می‌شوند. هر چه جلسات فصلی در استخراج اولویتهای حرفه واضحتر عمل کنند نیاز به مستندات سنگین نیازمندیها کمتر می‌شود.
- جواب دادن به سوال اساس کار را تشکیل می‌دهد:
  ۱. چه کار می‌خواهیم انجام دهیم.
  ۲. چه کار نمی‌خواهیم انجام دهیم.
  ۳. چگونه آن کاری را که می‌خواهیم، انجام دهیم.

فقط یک پایگاه کد

- شما می‌توانید در یک شاخه [(اصطلاح مربوط به مدیریت نسخه)] جدا کد بزنید اما این شاخه نباید بیش از چند ساعت دوام بیاورد. (این کار برای ما جا افتاده است.) چند شاخه کردن در بلند مدت به صرفه نیست.
- اگر چند تا شاخه دارید برای کم کردن آن برنامه‌ریزی کنید.
- می‌توان از طریق فایلهای تنظیمات کاری کرد که سیستم ساخت چند جور محصول بسازد. حتی اگر برای چند مشتری همزمان محصول می‌سازید.

استقرار روزانه

- هرشب تولیدات روز را در محصول نهایی قرار دهید تا با برنامه‌نویسان با نرم‌افزار مستقر شده سنکرون باشند و تصمیماتشان فیدبک درست بگیرد.
- این عملکرد در عملکردهای ثانویه قرار گرفته است زیرا مقدمات زیادی برای دستیابی به آن لازم است. مثلا تعداد خطاها باید از سالی چندتا بیشتر نباشد، محیط ساخت باید خودکار باشد، استقرار باید خودکار و با

امکانات roll out افزایشی و roll back باشد، و مهمتر از همه اعتماد درون تیمی و اعتماد مشتری به تیم باید بالا باشد.

- از استقرار سالانه کم کم می‌توان به استقرار روزانه رسید. (برای ما استقراری بین هفتگی تا ماهانه مناسب به نظر می‌رسد. حتما با مشتری در این رابطه توافق کنید، مشتری باید از اینکار خوشحال باشد و حتی کمک مالی و انسانی کند.)

قراردادهای با امکان چانه‌زنی در مورد محدوده

- در قرارداد هزینه، زمان، و کیفیت را قطعی کنید، اما برای محدوده سیستم امکان چانه‌زنی قرار دهید.
- از طریق امضای مجموعه‌ای از قراردادهای کوتاه مدت به جای یک بلند مدت مخاطرات را کاهش دهید. می‌توان قسمتهای اختیاری نیز تعریف کرد که در صورت توافق طرفین انجام شود.
- به جای قراردادهای با هزینه تغییرات بالا می‌توان قراردادهایی نوشت که محدوده کمتری از ابتدا قطعی شده و هزینه تغییرات پایینتر است.

پرداخت به ازای هر استفاده

این مدل تجاری در برخی از نرم‌افزارها دیده می‌شود: مثل نرم‌افزار ارسال SMS که مشتری را با هر استفاده شارژ می‌کند. این مدل از مدل فروش License بهتر است. (فعلا به کار ما نمی‌خوره)

فصل ۱۰ تیم کامل

- نباید خواسته‌های و سلایق فردی یا زیرگروهی اعضای تیم به تمامیت تیم و هدف مشترک لطمه بزند. [حتی اگر تشخیص شما به نظر شما صد در صد درست است اما در مجموع تیم به نتیجه دیگری می‌رسد باید به آن نتیجه گردن نهید. این لازمه کار تیمی است و بدون آن فواید کار تیمی از بین می‌رود.]
- براساس اصل جریان، کارها باید افزایشی باشد حتی معماری. این برخلاف خواست بسیاری از اعضا یا زیرگروهها خواهد بود که اعتقاد دارند کار آنها باید زودتر انجام شود.

آزمون‌گرها

آزمون گرها به مشتری و برنامه‌نویسان در نوشتن آزمونهای سیستم کمک می‌کنند. کار آنها افزایشی است.

طراحان تعامل

طراحان تعامل [ظاهرا این نقش عبارت است از ترکیبی از استخراج کننده نیازمندیها و طراح واسط کاربر] نیز کارشان افزایشی است.

معمار

- در XP معماری یک فاز نیست یک فعالیت است. معماران نیز کارشان افزایشی است و شامل refactoring است. آنها می‌توانند از نوشتن آزمونهایی که معماری را زیر فشار کاری قرار می‌دهند استفاده کنند. کافی است معماری به قدری خوب باشد که آزمون را رد کند.

- معمار مثل برنامه‌نویس کد می‌زند با این تفاوت که یک تصویر کلی از سیستم در ذهنش هست و در مواقع لزوم معماری را تغییر می‌دهد.
- معماری در XP برخلاف تفرقه بنداز و فتح کن، فتح کن و تفرقه بنداز است. تقسیم سیستم به مرور زمان انجام می‌گیرد.

مدیر پروژه

- مدیر پروژه باید ارتباطات تیم با بیرون و درون تیم را تسهیل کند. در XP برنامه‌ریزی یک فاز نیست یک فعالیت است.
- مدیر پروژه نباید گلوگاه شود.

مدیر محصول

- مدیر محصول مسئول تنظیم محصول از طریق انتخاب داستانهایی است که باید پیاده‌سازی شوند. این انتخاب بر اساس حرفه انجام می‌شود و نه مسائل فنی حتی اگر کارهای مهمتر پیش‌نیاز انجام نشده داشته باشند (مثل ایجاد امکان ویرایش چیزی قبل از امکان ساخت آن.) درموارد پیش‌نیاز دار می‌توان از طریق داده‌های الکی مشکل را حل کرد.
- مدیر محصول باید تضمین کند که از هفته اول یک محصول تمام (قابل اجرا) وجود دارد.
- اگر تیم در حال بکارگیری عملکرد "واقعا درگیر بودن مشتری" است، برقراری تعادل بین نیازهای مشتری و نیازمندیهای بازار (فروشهای بعدی محصول) به عهده مدیر محصول؟ است.

مسئولان اجرایی[12] (مسئولان سازمان)

- باید پشتیبانی لازم را از بکارگیری XP داشته باشند و به تیم شجاعت و اطمینان بدهند.
- تیم باید در مقابل آنها صداقت (شفافیت) داشته باشد.
- حق دارند از همه چیز یک تیم XP سوال کنند.
- دو معیار برای ارزیابی موفقیت تیمهای XP در به کارگیری عملکردها:
  ۱. کم بودن تعداد خطاهای پس از تولید (تعداد انگشت شمار).
  ۲. زمان به سودرسی تغییرات در فرآیند.

[یکی از نشانه‌های موفقیت روحیه تیمی این است که تیم باید شنگول و با انگیزه باشد و این از طرز کارکردن و مکالمات مشخص می‌شود.]

نویسندگان فنی[13]

- دو نقش دارند:
  ۱. به تیم در مورد ویژگیها فیدبک سریع بدهند: مثلا ارائه فلان ویژگی دشوار است.
  ۲. ارتباط با مشتری: از طریق نوشن راهنمای کاربر، خودآموز و ...

[12] Executives
[13] Technical Writers

- مشکل XP در این زمینه این است که مستندات را مثل مخلفات می‌کند. مثلا به دلیل ثابت نگه نداشتن نیازمندیها تا روز آخر، نوشتن راهنمای کاربر با مخاطره مواجه است.
- ویژگیهای مستندات خوب:
  ۱. دقیقا بعد از پیاده‌سازی ویژگی مستند شونده، نوشته شود.
  ۲. در صورت نیاز به تغییر ویژگی، مستند نیز به سادگی تغییر کند.
  ۳. ارزان باشد.
  ۴. زمانی به چرخه تولید اضافه نکند.
  ۵. نگارش آن برای تیم ارزشمند باشد.

کاربران

- کاربران در نوشتن داستانها کمک می‌کنند.
- کاربران نمونه باید تصمیماتی را که شک دارند، تا صحبت با سایر کاربران به تعویق بیندازند.

برنامه‌نویسان

- داستانها را به کارها می‌شکنند.
- داستانها و کارها را تخمین زمانی می‌زنند.
- آزمون می‌نویسند.
- ویژگیها را پیاده‌سازی می‌کنند.
- کارهای خسته‌کننده را خودکارسازی می‌کنند.
- کم‌کم طراحی سیستم را بهبود می‌بخشند.
- به دلیل تعامل نزدیک برنامه‌نویسان باهم، آنها باید مهارتهای اجتماعی و ارتباطی داشته باشند.

منابع انسانی

دو چالش در XP ارزیابی اعضای تیم و استخدام است. بدلیل برنامه‌نویسی دو نفره، ارزیابی شخصی افراد دشوار است. اعضای باارزش:

- محترمانه برخورد می‌کنند.
- با دیگران همکاری می‌کنند (در کار تیمی شرکت می‌کنند.)
- می‌توانند بدون انتظار به دستور از بالا تصمیم‌گیری و عمل کنند.
- به تعهدات خود عمل می‌کنند (سر موعد کارها را تحویل می‌دهند.)

از بین دو برنامه نویس فوق‌العاده ماهر و ماهر اما اجتماعی، دومی برای XP مناسبتر است.

نقشها

- نقشها در تیمهای رشدیافته XP ثابت نیستند. در شروع استفاده از XP نقشهای ثابت بد نیست اما کم کم به وضعیتی می‌رسیم که ثابت بودن نقشها برای هدف تیم مضر می‌شود. هر شخص باید هر کاری می‌تواند برای تیم بکند.

- تناسب اختیار و مسئولیت افراد باید حفظ شود. اگر کسی پیشنهادی می‌دهد باید خودش بتواند از انجام آن پشتیبانی کند.

فصل ۱۱ نظریه محدودیتها

- XP برای تولید نرم‌افزار است نه قسمتهای دیگر کار مثل بازاریابی.
- برای پیشرفت فرآیند تولید نرم‌افزار مثل هر فرآیند دیگر باید سیستم را به صورت کلی نگاه کرد و از بهینه-سازیهای خرد پرهیز کرد. به عنوان مثال فرضا ممکن است نوشتن آزمون یا انجام مرور در حین پیاده‌سازی زمان پیاده‌سازی را طولانی کند، اما در مجموع از زمان کل کار بکاهد.
- نظریه محدودیتها می‌گوید در هر زمان یک (و گاهی دو) محدودیت (گلوگاه) در سیستم وجود دارد. برای بهینه‌سازی کار باید برای آن گلوگاه فکری کرد. وقتی آن گلوگاه رفع شد حتما گلوگاه دیگری وجود دارد که باید روی آن کار کرد. گلوگاه مرحله‌ای از کار است که کارها دم در آن مرحله تلمبار می‌شوند.
- ممکن است گلوگاه فعلی به تولید نرم‌افزار ربط نداشته باشد. در این صورت باید به طریقی مستقل از روشهای منهدسی نرم‌افزار به آن مشکل پرداخت.

فصل ۱۲ برنامه ریزی: مدیریت محدوده

برنامه‌ریزی در XP مثل خرید از یک فروشگاه است در حالیکه شما پول محدودی دارید. در این حالت مجبورید از بین مجموعه کالاهای مورد نیاز زیرمجموعه‌ای را انتخاب کنید که هزینه آن بیش از پول شما نشود. در برنامه-ریزی XP داستانها نقش کالاها را دارند و تخمین زمانی ضمیمه شده به داستانها نقش اتیکت کالاها را دارد.

با توجه به اینکه وقت تحویل پروژه از ابتدا مشخص است (دو)نفر–ساعاتی که دارید نیز مشخص است. اگر تخمین زمانی کل داستانها از (دو)نفر–ساعتهای موجود بیشتر است باید به اندازه زمان موجود داستانهای با ارزشتر را انتخاب کنید.

با توجه به غیرمطمئن بودن تخمینها باید:

۱. با فیدبک تخمینها را اصلاح کرد.
۲. برنامه‌ریزی را تا جای ممکن به تاخیر انداخت تا بر اساس آخرین اطلاعات انجام شود. علت وجود برنامه-ریزیهای هفتگی و فصلی در XP همین است.

برنامه‌ها تخمینی از آینده هستند نه پیش‌بینی آینده. حداکثر کاری که یک برنامه می‌تواند انجام دهد پیش‌بینی یکی از حالات ممکن برای آینده است. اما این به معنی بی‌ارزش بودن آنها نیست بلکه برنامه به منزله یک نقطه شروع است. برنامه امکان هماهنگی با سایر تیمها و ... را فراهم می‌آورد [یکی دیگر از فواید برنامه این است که تیم را وادار می‌کند برای رسیدن به موعد تحویلها بیشتر تلاش کند. مسلما وقتی یک موعد پیش روی تیم است بیش از وقتی که فعلا خبری از تحویل و این حرفها نیست تلاش می‌کند.] برنامه باید براساس وقایعی که اتفاق می‌افتد تغییر یابد.

از بین سه پارامتر زمان، هزینه و کیفیت معمولا ۲ تا را مشتری مشخص می‌کند (زمان و هزینه) و تنها پارامتری که در اختیار تیم است کیفیت است. اما در حقیقت این پارامتر نیز در اختیار تیم نیست، زیرا اگر تیم با پایین آوردن کیفیت نشان دهد که پیشرفت بیشتری صورت گرفته، درنهایت زمان و هزینه بیشتری از دست می‌دهد تا کیفیت را به میزان قابل قبول برساند و در ضمن به رضایت مشتری و ارتباط با مشتری لطمه وارد می‌شود. تنها متغیری که خارج از این مدل باقی می‌ماند محدوده است. اگر محدوده به عنوان یک پارامتر مطرح شود:

- یک راه امن برای تطبیق داریم.
- راهی برای مذاکره داریم.
- محدوده‌ای برای نیازمندیهای مسخره و غیرضروری مشخص می‌شود.

در برنامه‌ریزی برای یک دوره (طولانی یا کوتاه) کارهای زیر را انجام دهید:

۱. اقلام کاری که ممکن است انجام آنها لازم باشد را فهرست کنید.
۲. آنها را تخمین زمانی بزنید.
۳. برای این دوره یک بودجه مشخص کنید.
٤. براساس بودجه در مورد کاری که در این دوره می‌تواند انجام شود، توافق کنید. در حین مذاکره تخمین و بودجه را تغییر ندهید.

این روش حتی در برنامه‌ریزی برای یک کار چند ساعته نیز استفاده می‌شود.

نفر–ساعت موجود را بر دو تقسیم کنید تا برنامه‌نویسی دو نفره را به حساب آورید. به برنامه‌نویسی دو نفره این انتقاد می‌شود که زمان را نصف می‌کند، در حالیکه نویسنده از روی تجربه معتقد است که اگر دو نفر جدا جدا کار کنند نمی‌توانند در مجموع به اندازه اینکه با هم کار کنند کار مفید کنند.

در برنامه‌ریزی به نظر همه اعضای تیم توجه کنید. این کار یک کار جمعی است که هم صحبت کردن دارد و هم گوش دادن. همه اعضا باید در آن شرکت کنند. اینکار انگیزه افراد را برای کاهش هزینه بالا می‌برد.

باید برنامه‌ریزی را با استفاده از تجربه بهتر کرد. به عنوان مثال در برنامه این هفته همانقدر کار قرار دهید که در هفته گذشته موفق شدید تمام کنید.

اگر در وسط یک دوره زمانی هستید و از برنامه عقب افتاده‌اید فکر کنید مشکل چیست: اهمیت دادن به جزئیات بی‌اهمیت؟ سهل انگاری در اجرای یک عملکرد مفید؟ اگر راه حلی نیافتید زیر مجموعه با اولویت بالاتری از کار آن دوره را انتخاب کنید (با نظر مشتری) و روی آن تمرکز کنید. در ادامه به این پاراگراف از کتاب توجه کنید:

The time this replanning takes will be more than repaid in increased harmony and efficiency as the team works towards the deployment date. Without the adjustment, you are working under a lie. Everyone knows it and has to hide to protect themselves. This is no way to get good software done and deployed; and it undermines team and individual confidence.

این نشان دهنده اهمیت شفاف بودن است.

فصل ۱۳ آزمون: زود، مکرر، و خودکار

خطاها اعتماد را، که لازمه تولید نرم‌افزار است، از بین می‌برد. مسئله دشوار این است که هم خطا پرهزینه است و هم رفع خطاها. پس بهتر است از خطا تا جای ممکن پیشگیری شود (با برنامه‌نویسی دو نفره، آزمون و...) و اگر رخ داد از آن درس گرفته شود.

میزان خطای قابل قبول در سیستمهای مختلف متفاوت است.

The world's largest web site may have a hundred software errors a second and remain economically viable because 99.99% of the pages appear correctly. Any given user experiences the web site as being reliable. The space shuttle, on the other hand, might be limited to one software-related failure per century to remain viable.

یک هدف در تیم نرم‌افزار رسیدن به میزان خطایی است که از نظر حرفه مناسب باشد.

یک هدف دیگر کاهش سطحی خطا تا جایی است که اعتماد به تیم افزایش پیدا کند. این سرمایه‌گذاری اثرات مفیدی در کار تیمی دارد:

Mistakes introduced by one programmer make it harder for everyone else to do their work. Every mistake by one team member that affects another costs the team time, energy, and trust. Good work and good teamwork build morale and confidence. If you can respect and trust your colleagues, you can be more productive and enjoy your work more. Hiding errors to protect yourself, while sometimes seemingly necessary, is a tremendous waste of time and energy. Trust energizes participants. We feel good when things work smoothly. We need to be safe to experiment and make mistakes. We need testing to bring accountability to our experimentation so that we can be sure we are doing no harm.

XP از دو مطلب در این زمینه استفاده می‌کند:

- آزمون نرم‌افزار دوباره چک کردن است. وقتی شما یک سری عدد را یکبار از پایین به بالا و یکبار از بالا به پایین جمع می‌زنید و نتیجه یکسان می‌گیرید احتمال خطا به شدت کم می‌شود. همینطور در مورد آزمون می‌توان گفت آزمونهای نوشته شده و کد هر دو بیان این مطلب هستند که برنامه چه کاری می‌کند. اگر برنامه آزمونها را رد کند. احتمال خطا دار بودن برابر است با احتمال اینکه هم برنامه و هم آزمون غلط باشند [حاصلضرب احتمالات] که کمتر است.
- هرچه دیرتر خطا کشف شود هزینه رفع آن بالاتر است. بنابراین بهتر است بلافاصله پس از ایجاد خطا کشف شود.

اینکه یک برنامه‌نویس خودش آزمون هم بنویسد این مزیت را دارد که هزینه مبادله اطلاعات با آزمون نویس از بین می‌رود اما بدلیل وجود فقط یک زاویه دید برخی مزایای دوباره چک کردن از بین می‌رود. به همین دلیل در XP علاوه بر آزمونهایی که برنامه‌نویس می‌نویسد، آزمونهایی نیز وجود دارند که از زاویه دید کاربر نوشته می‌شوند (آزمون سیستم). این دو روش یکدیگر را دوباره چک می‌کنند.

XP در مورد نحوه انجام آزمون دو حرف دارد:

- به جای انجام آزمون بعد از پیاده‌سازی باید آنرا در حین پیاده‌سازی انجام داد.
- باید آزمون خود کار باشد (حتی آزمونهای فشار و بارکاری.)

آزمون می‌تواند قبل از کد و بعد از آن نوشته شود. در XP هرجا ممکن باشد آزمون قبل از کد نوشته می‌شود. اینکار به استقلال واسط از پیاده‌سازی نیز کمک می‌کند.

آزمونها به ما در مورد سیستم اطمینان می‌دهند. با وجود اینکه ممکن است آزمونی باشد که ما ننوشته‌ایم و سیستم رد نکند، اما حداقل آزمونهایی را که سیستم رد می‌کند، می‌دانیم. آزمونهای نوشته شده را می‌توان معیاری برای پیشرفت سیستم نیز قلمداد شود.

بهتر است یک برنامه‌نویس در هر لحظه یک آزمون رد نشده داشته باشد و آنرا رفع کند، سپس آزمون بعدی را بنویسد تا با رد شدن اولی و داشتن اطلاعات بیشتر مجبور به بازنویسی دومی نشود.

فصل ۱۴ طراحی کردن: ارزش زمان

- XP طراحی تدریجی را به نهایت می‌رساند. [XP تفکر رایج در "اول معماری یا طراحی" را کاملا زیر سؤال می‌برد.] به عقیده نویسنده طراحی نرم‌افزار برخلاف طراحی‌های فیزیکی (مثل ساختمان) در حین پیاده‌سازی قابل تغییر و اصلاح است.
- بقدری طراحی کنید که برای گرفتن فیدبک کافی باشد، بعد با استفاده از فیدبک طراحی را اصلاح کنید تا فیدبک بعدی را بگیرید. با توجه به اینکه تجربه بهترین وسیله برای بهبود طراحی است در XP طراحی یک کار همیشگی است و با پیاده‌سازی اصلاح می‌شود. البته روش مناسب طراحی در موقعیتهای مختلف متفاوت است. ممکن است در برخی پروژه‌ها طراحی کامل پیش از پیاده‌سازی به صرفه باشد یا طراحی خوبی را نتیجه دهد.
- مهمترین قاعده طراحی "یک و فقط یک [جا]" می‌باشد. هر داده، ساختار، یا منطق باید در یک جا در سیستم وجود داشته باشند نه بیشتر.
- اگر در یک سیستم کثیف کار می‌کنید هم می‌توانید کم کم طراحی را بهبود دهید. هرجایی از سیستم را که تغییر می‌دهد می‌توانید از نظر طراحی تغییر دهید، اما از آن محدوده خارج نشوید. (می‌توانید فهرستی از سایر بهبودهای قابل انجام در طراحی تهیه کنید.)
- هرگز سیستم را متوقف نکنید تا تغییرات (بهبود) را در طراحی اعمال کنید. در XP طراحی در خدمت رابطه معتمدانه بین افراد فنی و افراد حرفه [سهامداران و مدیران] است. نباید از هدف تحویل هفتگی دور شد زیرا افراد حرفه کارکردهای سیستم را می‌بینند نه خوبی طراحی را.
- به طور خلاصه طراحی در XP بقدری به تاخیر بیفتد که تحت راهنمایی تجربه انجام شود. این کار این مزایا را دارد:
  - تحویل سریعتر نرم‌افزار
  - تصمیم‌گیری با اطمینان بیشتر
  - اجتناب از زندگی در کنار تصمیمات بد
  - حفظ روال تولید در حالیکه مفروضات اولیه طراحی منسوخ می‌شوند

- تیمهای XP در حد امکان راه حلهای ساده را ترجیح می‌دهند. معیارهایی برای سنجش سادگی طراحی:

۱. برای مخاطبین مناسب باشد: اگر سیستم بیش از شعور کاربران قابلیت داشته باشد، برای آنها ساده نخواهد بود [و در ضمن منابع به هدر رفته‌اند.]
۲. گویا: هر ایده‌ای که لازم است مبادله شود در سیستم قرار می‌گیرد. شبیه لغات موجود در مجموعه لغات، اجزای سیستم با خوانندگان آینده ارتباط برقرار می‌کنند.
۳. فاکتورگیری شده: تکرار ساختار یا منطق فهم و تغییر کد را دشوار می‌کنند.
٤. حداقل: طبق ۳ مورد بالا سیستم باید کمترین اجزای ممکن را داشته باشد. در این صورت حداقل مستندات، آزمون و ارتباطات را خواهیم داشت.

فصل ۱۵ تطبیق مقیاس XP

در مسائل زیر مقیاس پروژه‌ها باهم متفاوت است:

- تعداد افراد
- سرمایه‌گزاری
- اندازه کل سازمان
- زمان
- پیچیدگی مسئله
- پیچیدگی راه حل
- نتیجه شکست (سیستمهای امنیت-بحرانی[14] و سلامتی-بحرانی[15])

نویسنده معتقد است با آگاهی و تطبیق مناسب XP مقیاس‌پذیر است. اکنون برخی از این موارد را بررسی می‌کنیم. (مقیاس‌پذیری برای سرمایه‌گزاری، اندازه کل سازمان، پیچیدگی مسئله، و نتیجه شکست در کتاب هست ولی چون به کار ما نمی‌خورد نیاوردم.)

تعداد افراد

لزوما برای حل مسائل ظاهرا بزرگ نیاز به تیمهای بزرگ نیست.

اگر مسئله واقعا بزرگ است آنرا به چند قسمت کوچک تقسیم کنید (این یک مسئله کوچک است که با یک تیم کوچک حل می‌شود). سپس هر قسمت را به یک تیم کوچک بدهید. برای اینکه از مخاطرات مربوط به تجمیع اجتناب کنید مکررا تجمیع را انجام دهید. این یک راهبرد فتح کن و تقسیم کن است نه تقسیم کن و فتح کن.

کار تیمها باید طوری باشد که هر تیم انگار تنها تیم است. به این طریق هزینه ارتباطات به حداقل می‌رسد و ارتباطات به حالتهای استثنایی محدود می‌شود. اگر مشاهده شد که ارتباطات زیاد است، باید ساختاربندی مجدد پروژه به عنوان یک انتخاب مورد توجه قرار گیرد. اگر این هم مشکل را حل نکرد آنگاه تیم به سراغ روشهای مدیریت پروژه‌های بزرگ می‌رود.

[14] Security-Critical
[15] Safety-Critical

بنابراین نباید یک دفعه به سراغ آخرین راه حل رفت بلکه باید همواره سعی کرد با حداقل تعداد نفرات مشکل را حل کرد و اگر نشد افراد و تیمها را زیادتر کرد.

زمان

در مورد مقیاس پذیری برای زمان طولانی XP بدلیل وجود آزمونها مناسب است.

پیچیدگی راه حل

XP می‌تواند به مدیریت این مسائل کمک کند و تیمهایی که گیر کرده‌اند را نجات دهد. به پاراگراف زیر که تجربه نویسنده در مورد یکی از مشتریهایش را شرح می‌دهد، توجه کنید:

One client began by getting the build process under control. The team improved the build so instead of taking 24 hours on a dedicated machine with lots of manual intervention, the build took an hour and could run completely automatically on any machine. Then, the team instituted stories and a story board so everyone knew who was working on what and how long they were taking. After two years of steady improvement the team reduced costs 60%, going from seventy engineers to twenty; reduced the time to fix defects 66%; and reduced the time to release for major and minor point releases by 75%, from ten weeks to two weeks. Once the team had stopped digging itself in deeper, it began to climb out by eliminating excess complexity while also fixing defects.

به طور خلاصه راه‌حل XP در برخورد با پیچیدگی عمل کردن محلی و بهبود تدریجی است.

فصل ۱۶ مصاحبه

در این فصل با شخصی به نام Brad Jensen که معاون رئیس شرکت Sabre Airline Solutions که XP را برای ۱۳ تیم به کارگرفته است، مصاحبه شده است.

فصل ۱۷ خلقت [XP]

در این فصل داستان پیدایش XP گفته شده است.

فصل ۱۸ تیلوریزم و نرم‌افزار

تیلور اولین مهندس صنایع بود کار او به این ترتیب بود که نحوه کار کارگران را در کارخانه مشاهده کند و آنرا بهبود بخشد. نام این روش را مدیریت علمی گذاشت (بدلیل شباهت با روش علمی: مشاهده، فرضیه، و آزمایش).
این روش در عین مزایایی که دارد نقایصی هم دارد. این نقایص از سه فرض ساده‌کننده نشات می‌گیرند:

۱. چیزها همانطور که برنامه‌ریزی شده‌اند پیش می‌روند.
۲. بهینه‌سازی خرد به بهینه‌سازی کلان می‌انجامد.
۳. افراد قابل جایگزینی هستند و فقط کافیست به آنها گفته شود چه کنند. [برخورد ماشینی با افراد]

برخی اثرات منفی تفکر تیلوریزم بر نرم‌افزار:

۱. جداسازی برنامه‌ریزی و اجرا: تخمین زمانی به جای برنامه‌نویس توسط دیگری زده شود. یا اینکه توسط یک گروه نخبه (معماران یا تیم چارچوب) از پیش شرح داده شود که یک کار چگونه توسط کس دیگری انجام شود.
۲. جداسازی کیفیت (در قالب واحد کنترل یا تضمین کیفیت) از مهندسی (پیاده‌سازی): این ایده باعث می‌شود تیم مهندسی مسئولیت کیفیت را به عهده خود نداند.

فصل ۱۹ سیستم تولید تویوتا

- تویوتا یک شرکت موفق، پول درآور، و پیشرونده است. این شرکت با تلاش بیش از حد پیش نمی‌رود بلکه با ازبین بردن اتلافهایی که در فرآیند تولیدش می‌شود بهبود می‌یابد.
- در تویوتا همه کسانی که در خط تولید کار می‌کنند مسئول کیفیت هستند نه کسی که ته خط ایستاده. هرکس باید به محض مشاهده یک اشکال خط را متوقف کند.
- کارکنان (یا کارگران ؟) مسئول رفع اتلافها نیز هستند. آنها با مشاهده منبع اتلاف، آنرا گزارش می‌دهند و خود مسئولیت تحلیل و آزمایش برای رفع آنرا به عهده می‌گیرند.
- این روش طبقه‌بندی اجتماعی موجود در تیلوریزم را از بین می‌برد.

این مطالب نیز در مورد سیستم تولید تویوتا و مهندسی نرم‌افزار خواندنی است:

Workers are accountable for the quality of their work because the parts they create are immediately put to use by the next step in the line. This direct coupling of functions was completely counterintuitive to me when I first read about it. I thought that for a mass-production factory to run smoothly there must be a large inventory of parts between any two steps of the process. That way, if the upstream machine stops working, the downstream machine can keep chugging away on the buffer of parts.

TPS turns this thinking on its head. While individual machines may work more smoothly with lots of "work-in-progress" inventory, the factory looked at as a whole doesn't work as well. If you use a part immediately you get the value of the part itself as well as information about whether the upstream machine is working correctly. This view, that parts aren't just parts but also information about their making, leads to pressure to keep all the machines in a line working smoothly and also provides the information necessary to keep the machines working smoothly.

Taiichi Ohno, the spiritual leader of TPS, says the greatest waste is the waste of overproduction. If you make something and can't sell it, the effort that went into making it is lost. If you make something internally in the line and don't use it immediately its information value evaporates. There are also storage costs: you have to haul it to a warehouse; track it while it is there; polish the rust off it when you take it back out again; and risk that you'll never use it at all, in which case you have to pay to haul it away.

Software development is full of the waste of overproduction: fat requirements documents that rapidly grow obsolete; elaborate architectures that are never used; code that goes months without being integrated, tested, and executed in a production environment; and documentation no one reads until it is irrelevant or misleading. While all of these activities are important to software development, we need to use their output immediately in order to get the feedback we need to eliminate waste.

Requirements gathering, for instance, will not improve by having ever more elaborate requirements-gathering processes but by shortening the path between the production of requirements detail and the deployment of the software specified. Using requirements detail immediately implies that requirements gathering isn't a phase that produces a static document; but an activity producing detail, just before it is needed, throughout development.

فصل ۲۰ به کار بستن XP

- مشکلات افراد و تیمها در خودشان است و نه در فرآیند. بنابراین این XP نیست که مشکل آنها را حل می‌کند، بلکه خودشان باید مشکل را حل کنند. XP به بهبود سریع تولید نرم‌افزار می‌انجامد، اما پیشرفت واقعی در بهبود مشکلات (اتلافهای) اصلی خود تیم است که ممکن است با وجود استفاده از XP بازهم وجود داشته باشند. (باید باورها و روشهای غلط را خود تیم اصلاح کند.)
- نحوه استفاده از XP در هر تیمی مخصوص آن تیم است.
- برای اعمال تغییر از خودتان شروع کند. خود به عنوان الگو یا مثال موفقیت عملکردها را نشان دهید.
- آدم باید دائما روش خود را بهبود دهد زیرا هیچ وقت به بهترین حالت نرسیده است.
- یک مربی می‌تواند به استفاده تیم از XP کمک کند، هر چند بدون مربی هم می‌شود تجربه کرد.

فصل ۲۱ خلوص

- روشی برای سنجش اینکه یک تیم XP است یا نه نداریم. مهم موفق بودن است. موفقیت با یک تیم آبشاری بهتر از شکست در یک تیم XP است.
- XP برای کنار گذاشتن مستندات بهانه نمی‌شود. نباید هرکس به اسم XP هرکاری دوست دارد بکند.

فصل ۲۲ راه بی‌پایان برنامه‌نویسی

- عنوان این فصل از کتاب کریستوفر الکساندر[۱۶] در زمینه معماری ساختمان نوشته شده است. او سعی کرده بگوید معماران باید کمتر خودخواه باشند و خواست کسی را که می‌خواهد در خانه زندگی کند در نظر بگیرند. چنین مشکلی در محصولات مهندسی نیز وجود دارد:

We'll give you what you need even if you don't know you need it.

[16] Endless way of building

- از سوی دیگر در مواقع بسیاری مسائل حرفه‌ای و سیاسی بانیان پروژه پا بر روی مسائل فنی تیم تولید می-گذارد. مثلا زمانی که مهلت تحویل به جای وابسته بودن به میزان کار مورد نیاز براساس اولویتهای حرفه تنظیم شود.
- برای انجام کار به صورت مناسب باید تعادل بین خواسته‌های سه گروه مشتری، سازندگان، و کاربران برقرار شود. بنابراین مهندسی نرم‌افزار عبارت نیست از "گروهی برنامه‌نویس و یک مشت آدم دیگه".
- اگر انسان قلبا این موضوع را نپذیرد، تمام بهترین عملکردهای دنیا هم نمی‌توانند بهبود زیادی ایجاد کنند.

فصل ۲۳ تولید در آنسوی آب

مطالبی در مورد صنعت تولید نرم‌افزار در سایر مناطق دنیا گفته شده است.

فصل ۲۳ جامعه [XP]

گروه XP در Yahoo (خود نویسنده کتاب هم ظاهرا در این گروه هست):

http://groups.yahoo.com/group/extremeprogramming

فصل ۲۵ نتیجه‌گیری

XP می‌تواند به اشخاص انگیزه و امید بدهد.