SS2023🥷発表内容（文字起こし）

1.
みなさん、こんにちは。キヤノンメディカルシステムズの深谷美和です。
これから「テストからはじめよ」〜忍者式テスト 20 年の実践から〜というお話しをします。
よろしくおねがいします。

2.
わたしたちは、X線CT装置に搭載するソフトウェアをエクストリームプログラミングで開発しています。
今日はこのチームの活動の中から、反復開発とそれに有効なプラクティス「忍者式テスト」を説明します。

3.
アジェンダです。
はじめに、前提となる反復開発の話、次に忍者式テスト、そして、忍者式テストの効果をお話しします。

4.
自己紹介です。
深谷です。ソフトウェア開発に従事しています。プログラマを経験したのち、現在はテスターをしています。
関です。プログラマです。Rubyのコアコミッタでもあります。今日は主にスライドをめくる係です。

5.
本題に入る前に「話したくなったわけ」をお伝えしたいです。

このチームにJoinする前、わたしは別のチームで開発していました。当時、限られた期間の中でどのようにテストしていけばよいか悩んでいました。そしたら、関くんが「うちのチームは、毎日テストを全部手動でやり直している」と言うんですね。

わたしたちが扱っている製品は、規模が大きく複雑です。当然テストケースも膨大なので、それを全部？どうやって？？　と、なんだか「変わったチームだなあ」と思っていました。

数年後このチームにJoin するのですが、本当に全部やり直していました。こんなチームは見たことがないです。これは世界に紹介しなくては！と思いました。

それから、もうひとつ。

タイトルにある「テストからはじめる」という習慣が、わたしはとても気に入っています。何かを作るとき、どう作るのか？ではなく「どうやってためすのか？」から考える、ということです。

わたしたちは「うまくいったらどうなるの？」というフレーズをよく使います。

作ろうとしたものができたときの具体的な状態をイメージするのです。
これは、製品だけでなく、あらゆる活動についても同じです。

もちろんこの発表自体も「この発表がうまくいったらどうなるの？」から、はじめました。
わたしたちのチームに、興味を持っていただけたら嬉しいです。

6.
さて、本題にはいります。はじめは反復開発のお話です。

7.
これは、おなじみの、ロイズによる「大規模ソフトウェア開発のための実行ステップ」です。

8.
このようにソフトウェア開発はシステム要求からコーディングまでの「作り出すステップ」と、テスティング、運用の「確認するステップ」で構成されます。

このテスティングを展開し、

9.
各ステップに対応するテストを配置すると、V字モデルになります。

これらのモデルは、各ステップの依存関係を示したものです。
前のステップが終わらなければ、次のステップに移れない、といった関係です。
時系列のモデルではありません。

10
実際の開発は各ステップを1回やって終わりではありませんよね。
ステップは1つずつ進みますが、問題が見つかるとnステップ前に戻ります。
たとえば、テストで問題が見つかり、設計に戻った結果、さらに要求まで戻るケースもあります。

問題に気づいて、nステップ前に戻るのは健全な状態です。

11.
ロイズは、むしろ積極的に「基本設計を二度行いましょう」と言っています。
一度目は問題発見に力を注ぎ、二度目に解決策を組み込むといった作戦です。

12
さて、ここでIPAの資料「工程別の問題発見能力の違い」を見てみます。
このグラフは不具合の原因工程と発見工程の比率を示したものです。

不具合の原因工程は紫色のバーです。上の「作り出すステップ」に偏っていますね。
一方、黄色のバーが示す発見工程は、下に偏っています。

13.
円グラフにしたものがこちらです。
不具合の原因は「作り出すステップ」で全体の6割弱、不具合の発見は「確認するステップ」で全体の7割を超えています。

これは、みなさんの実感とも、そう変わらないのではないでしょうか。

14.
以上のことから
・企画・仕様、設計の問題を、作り出すステップで見つけるのは難しい
・テストは問題を見つける能力が高い
ということがわかります。

試さずに会議やレビューだけですべての問題を発見するのは難しいです。
これは上流工程がサボっているのではなく「そのときに見つける」のが難しいんですね。このことを真摯に受け入れる必要があります。

だから、中間成果物、たとえば、設計書は承認済み！とか、部品はできてます！などがそうですが、それを根拠にうまくいっていると信じるのは、危なっかしいと感じます。

このように、テストは問題を見つける能力が高いので、試してわかることは試した方が効率が良いです。
実験が容易なのはソフトウェアの特徴のひとつです。

15.
これらの事実と特徴を利用しましょう。

工程と時期をわけて考え、問題を発見する能力が高い工程を、プロジェクトの初期から行います。
そうすれば、早い時期に問題を発見できる可能性が高まりますし、早い時期に発見できれば、修正が容易になります。

作戦はこうです

・システムテスト、運用・実機テストをより早い時期に行う
・結合された状態でのテストを早く、頻繁に！

16.
そこで反復開発ですよ！（どわあああぁ〜〜〜〜ん：脳内で銅鑼を鳴らす）

17.
テストを早い時期から実施するために、プロジェクト全体を「n回のV字」で構成します。

これは時系列にV字を並べた図です。
たとえば、1か月のプロジェクトを4回のV字で構成したとすると、

18.
こんな風に、最初の週からテストするんです。
問題を見つける能力の高い「テスト」をプロジェクトの初期から実施できます。

19.
また、反復しているので、発見した問題や違和感や知見を、すぐに次のV字に活かせます。

ロイズの主張で「2回開発しましょう」というのがありましたけど、n回やってもいいわけです。
反復開発、ほんとよくて、おすすめです。

20.
反復開発を実際にやろうとすると、難しいところがあります。
短い周期でV字をまわすと「頻繁に、システム全体のテスト」を実施しなくてはなりません。

このテストの必要性は、一般的なバージョンアップ開発と同じです。

新しい反復により追加、変更された仕様だけでなく「システム全体が問題なく動くことの確認」が必要です。

これをね、頻繁にやろうとすると、かなりの負担になります。

21.
そこで忍者式テストですよ！（どわあああぁ〜〜〜〜ん：脳内で銅鑼を鳴らす）

22.
まず「忍者式テストの概要」と「それを構成する要素」について説明します。
そして「規模とどのように向き合ってきたのか」
最後に20年間の「テストの記録」を使って、チームの歩みをふりかえります。

23.
忍者式テストは、テスト駆動開発を「受け入れテストのレベル」で行います。
どういうことかというと

・xUnitを使ったTDDのように、受け入れテストからはじめる開発です。
・テストコードに導かれてプログラムを開発するように、受け入れテストに導かれてストーリーを実現します。ストーリーについては、このあと説明します。
・ストーリーが増えるたびに、それまでに作った全てのストーリーの受け入れテストをやりなおします。

とてもシンプルな作戦です。

24.
まずは、忍者式テストの全体像をつかみましょう。
これは、ある開発の初日から三日間のようすです。

1日目にストーリー1をコミットしました。これをバージョン1のシステムとして受け入れテストします。
2日目はバージョン2です。このとき、ストーリー2だけでなく、ストーリー1の内容も確認します。
3日目も同様です。バージョン3では、ストーリー3だけでなく、ストーリー1と2の内容も確認します。

25.
ストーリーというのは、XPなどの反復開発で製品を変更する単位です。

反復開発では、開発を「ひとつの大きなかたまり」ではなく「既存システムへの小さな変更の連続」と考えます。この小さな変更をXPではストーリーと呼びます。

ストーリーは受け入れテストで確認するので、ユーザーにとって「システムがどのように変化するか」がわかるものでなければなりません。

製品は「ストーリーが積み重なって」できています。

26.
わたしたちは、ストーリーをチケットで管理しています。

チケットには、ストーリーの概要や、要求の背景、テストケースを書きます。
テストケースには、システムの具体的な変化とその確認方法を書きます。

開発日記には、毎日の試行錯誤の様子、実験した内容や結果などを書きます。
ここを読むとソースコードには残りづらい「設計や実装の根拠」がわかります。

ストーリーの大きさや粒度は、開発しながら「調整し続けていくもの」です。
わたしたちが扱うストーリーの多くは数日で完成する大きさです。

また、開発中に見つけたバグもストーリーと同様に扱っています。バグ管理システムなどで別管理ということはしてません。

27.
受け入れテストはストーリーごとに行います。

チケットに書かれたテストケースをもとに、システム全体で結合した状態で試します。ユーザーが使うのと同じように一連の流れで試します。

28.
この受け入れテストは手動で行っています。

わたしたちが見つけたいものは、理想との差分、なんですね。

機能が動くかどうかはもちろんですが、使いづらさや、誤解をまねく仕様を見つけたいです。
それから、手触りの良し悪し。
たとえば、操作の反応の自然さ、なめらかさなど、本物を触らないとわからないことを大切にしています。

テストケース自体もテストの対象です。

このストーリーでやりたかったことに対して、それが試せる内容になっているのか？と問い直し、過不足はないか、テストのやりづらさはないかを、テストしながら見ていきます。
受け入れテストを繰り返す過程で、テストケースを洗練していくんですね。

また、陳腐化した仕様の修正や、開発プロセスの微調整も対象です。

わたしたちが見つけたいものは、こういった、理想の製品、テストケース、開発プロセスなど、さまざまな理想との差分です。

29.
そして、この理想も日々アップデートされます。

なぜなら、製品に求めるレベル、よい製品のイメージは、市場や環境の変化によって変わっていきますし、自分自身の製品に対する理解も深まっていくからです。

そういうわけで、手動でテストしています。

30.
さてみなさん、この図は忍者式テスト3日間の様子ですが、これが1か月、1年となったらどうなるでしょうか？
このテストケースがどんな感じになっていくのか、想像してみてください。

31.
みなさんのご想像のとおりです。こんなふうに、たいへんなことになります。

実際の開発では、毎日複数のストーリーをコミットしており、たいへんなペースでテストの束が積み上がっていきます。

32.
というわけで、チームがどのように「規模と向き合ってきたのか」をお話しします。

33.
忍者式テストをはじめた20年前はチケットの数も少なく、1日ですべてのテストケースを確かめることができていました。

でも1日で「回せなくなる日」がやってきます。

わたしたちは、1日ではなく、一定期間でテストケースをすべて回す、という作戦に切り替えることにしました。
そこで、テストをいい感じに回すためのアルゴリズムを開発します。

まんべんなく、かつ、効果の高いテストケースを抽出するアルゴリズムです。

具体的には

・まだテストしていない新しいストーリーと修正ホヤホヤのチケットを最優先とし
・前回パスしたテストは、出現頻度を落とします。

つまり隠す期間を計算しています。

また、開発の状況に応じて機能ごとに、出現頻度の「重み」を調整します。
たとえば、機能Aの改修をしている場合、機能Aと関連する機能Bの出現頻度も上げる、といった具合です。

これにより、新しい機能、問題があった機能、注目している機能を優先しつつ、ある期間で見るとシステム全体をテストできます。

また、抽出したテストスイートを「本日のおすすめテスト」と呼んでいます。

わたしたちのチームでは、毎日ひとり1時間この受け入れテストを担当しています。
このペースで無理なくできる量しか表示しないのも、量に圧倒されないための工夫です。

34.
さて、ここからはそのアルゴリズムによってテストした結果を見ていきたいと思います。

まずこのグラフの見方ですが、よこ軸が営業日です。20年分なので4000営業日を超えてます。

たて軸はチケット番号です。チケットの数です。
1つのチケットには複数のテストケースが書かれているのでテストケースの数ではないです。

そして、ひとつひとつの点がテスト結果です。テストをパスしたものは水色の点、NGが赤色の点です。NGが少なくて見えなかったので、赤色を強調しています。

この図からわかることを説明します。

35.
開発初期は、毎日すべてのチケットのテストを実施しているため隙間がありません。ミチミチですね。

開発が進みチケットが増えていくと色付けがだんだん薄くなります。俯瞰して見ると、まんべんなく全体的にテストしているのがわかります。

36.
チケットの数がリニアに増えています。
開発が進むに連れ、システムが複雑になり、また、要求される内容も高度になっていきますが、チケットの増え方は一定です。
これは、20年間、同じペースで開発していることを示しています。

小さな反復とこまめなテストによって「規模が大きくなっても変更コストを一定に保てる」というXPの主張どおりになっています。

37.
グラフの稜線のあたりの赤色が目立ちます。これは新しいチケットがすぐNGになっていることを示しています。
新しいチケットを優先的にテストしている、ということと、そこで多くの問題が見つかっていることがわかります。

作る前のレビューや開発中に議論を尽くしてもなお、実際に製品を触ってわかることがある、ということです。

また、開発中に設定したゴールはクリアしても、製品を前にすると、より高いゴールを求めてしまうことも多いです。

テストで「より高いゴールを求める」のはテスターだけでなく、プログラマもそうです。
うちのチームはプログラマもテストがうまいんですよ。「プログラマの帽子」と「テスターの帽子」を一日に何度も切り替えてるな、と感じます。素晴らしい。

38.
3800日目くらいから、下半分に白が目立つ部分があります。
これは次世代の製品にシフトした時期と一致しています。事業計画に従って、テストの注力点を調整しているのがわかります。

その後、徐々に全体をまんべんなくテストするように回復しています。

39.
最後に、忍者式テストの効果についてお話しします。

40.
忍者式テストを始めて最初に起きたことは、難しそうな問題も躊躇せずに修正できるようになったことです。不具合や性能などの「実装レベルの問題」に積極的に対応できるようになりました。

これは、もしどこかを壊しても、わたしたちは見つけられる、直せる、という安心感がそうさせたのだと思います。

次に起きたのは、直せる幅が広がったことです。
理想の製品をイメージして、それとの差分を積極的に探しだす者が現れました。それがチームに広がっていき、数年後には全員がずっと「良い製品とは何か」を問いながら、開発できるようになりました。

これが忍者式テストの「一番大きな効果」だと思います。

41.
また副次的な効果として「学習機会の創出」があります。

毎日のテスト時間や、朝会のチケットの読み合わせなど、さまざまな場面で、製品の仕様、設計、実装など、製品に関する質疑応答が絶えず行われます。

これは「圧倒的な量のドリル学習」に相当します。これにより、ひとりひとりの脳内にシステムのあらゆることが配線され、製品を動かしたときに「瞬時におかしさに気づくようなこと」が起きるのです。誰かのちょっとした疑問が、別の誰かの思考を刺激します。そうやってチームは「ひとつの巨大な脳を持った生き物」のようになるのです。

42.
今日は、反復開発と忍者式テストとその効果についてお話ししました。

忍者式テストは「XPを拡張するプラクティス」であり、その名前には「テスト」という言葉が入っておりますが、テストだけでなく、テストを起点とした「開発全体の有り様」を語っていることが、今日の話から伝わりましたら、嬉しいです。

以上になります。ありがとうございました。

43.