THE AUTOMATION TRAP AND HOW BIOWARE ENGINEERS QUALITY

Alexander Lucas

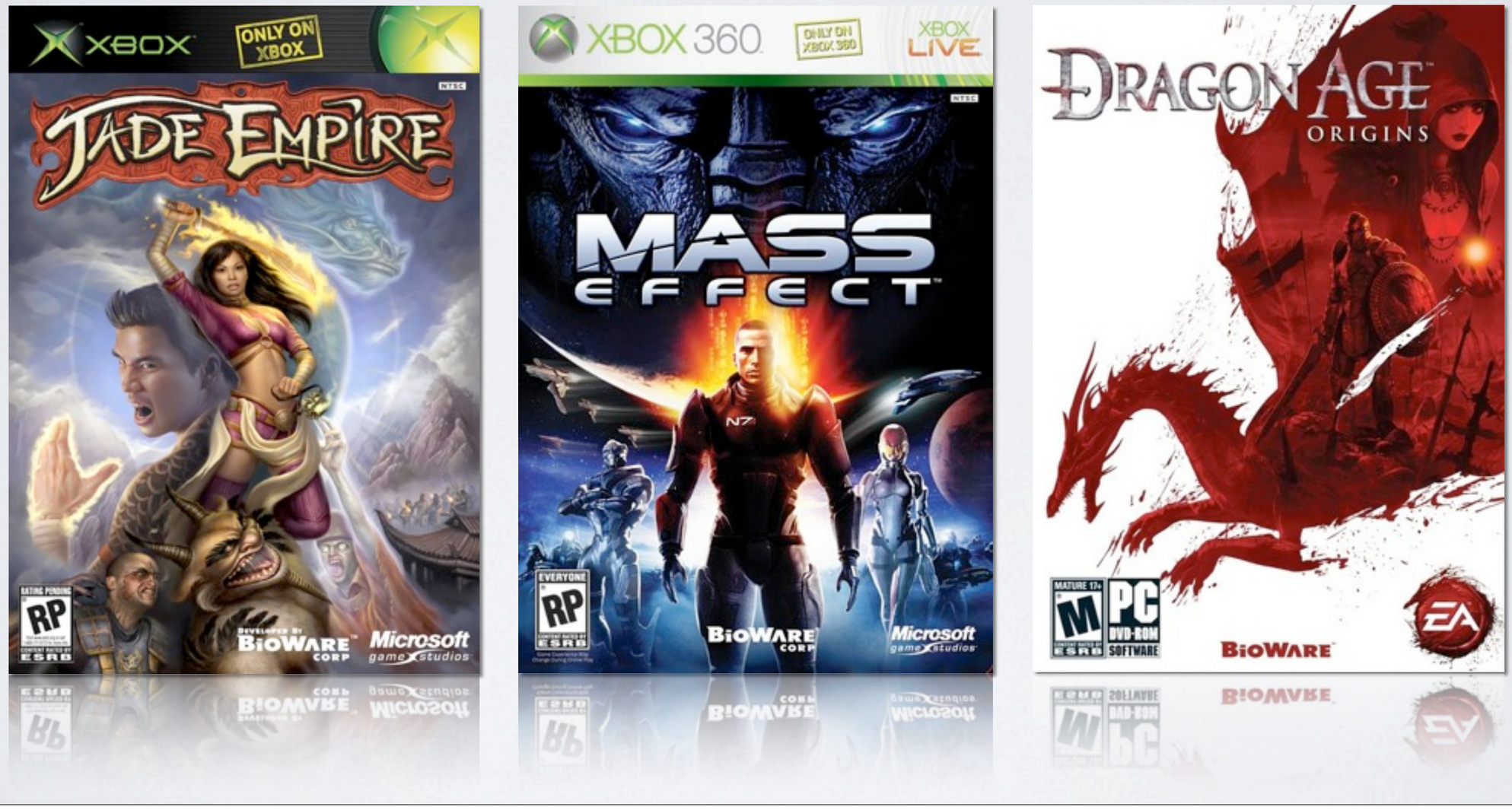
QA Engineering Lead, BioWare

BioWare

GAME DEVELOPERS CONFERENCE 2012

The Road Travelled

Lessons Learned in Testing and Automation



Friday, 9 March, 12

1) Jade: bot explored game, FPS, Mem
   Re-engineered daily, could've used telemetry
2) Mass Effect: More of the same, similar problems new tech
3) DAO: Technically mature and successful. TPM->Automated tests, distributed,
   abandoned due to maintenance costs

Jade Empire marked one of our earliest explorations into automated testing. We engineered a bot that would explore the game, from level to level, measuring performance across several vectors we were interested in (frame-rate, memory usage, loaded resources, etc.) While a marvel to watch, this solution rarely worked across multiple builds, and given the velocity at which the builds were coming in near the end of the project, the automation system needed to be reengineered almost daily to accommodate the rapid changes to the game designers were making. To further make matters worse, many of the measurements were inaccurate or invalid as the automation did not interact with the game legitimately, like a player would. In retrospect, the few benefits we did gain from the system could have been achieved with a much less costly telemetry system. Nonetheless, we remained nonplussed and continued our press for automation on Mass Effect, our follow-up title.

<Mass Effect - more to come>

Dragon Age: Origins saw the complete maturation of our automation efforts. Figuring our testers, who owned our testing strategy, were the best ones to implement the automated test, we engineered a solution that allowed them to convert their test plans, piecemeal, into automated testing suites. The tests were then automatically registered with a scheduling server and could be run locally or distributed. The results would even report back into the test plans in real time. It had a great wow factor! Furthermore, we felt like we had finally engineered a winning solution. Because our testers wrote and maintained the scripts and they were being derived from existing test plans, the tests had to be useful. Right?

Examining the data after the game launched, we discovered the reality of our efforts wasn't quite as successful. Despite having a winning solution from an engineering point of view, the number of defects detected was laughably low. Furthermore, most testers ended up abandoning the system after a time as their automated tests were never able to keep up with designer changes. They found themselves spending too much time trying to keep their tests from reporting false positives.

We realized, at this point, that we had fallen prey to a number of mistaken assumptions...

Common Assumptions

- Quality comes from testing
- Testing is costly
- Testing can be easily automated
- Use as much automation as possible
- Defect detection is paramount

Friday, 9 March, 12

1) Quality can't be tested in
2) To a point, dev is more costly
3) haven't replaced, only added. something of value?
4) run 1000s of times, what say about strategy?
5) important but WNF, what makes quality?

1) Quality comes from testing - This is an easy trap to fall into. Testing is quality control not quality assurance. Quality can neither be tested-in nor tested-out, it comes from the combined effort of the development team and through extensive iteration. If your goal is to produce a quality game, then you going to need to look beyond just testing.

2) Testing is costly - Sure, to a point, but it's both necessary and unavoidable. How many testers can you hire for what you pay one of your senior engineers? This is the trade you are making. Test automation requires extensive software development (the test automation framework, scripting, maintenance, etc.) and development costs are always higher than testing costs.

3) Testing can be easily automated - There's nothing easy about automation and automated tests are qualitatively different than manual tests. You haven't replaced anything; assuming you haven't stopped testing by hand, you've only added to the pile. Have you added something of value?

4) Use as much automation as possible - If we can run an automated test easily, surely we can run it thousands of times. Think of all the cost savings! If this sounds right, what does this say about your testing strategy?

5) Defect detection is paramount - Absolutely, we need to find and fix defects in our software. This goes without saying. But is it the most important thing to worry about? How many issues get "will not fixed" at the end of the project? Would finding more defects really have a measurable impact on the quality of your game? How exactly do we measure quality for a game, anyway?

You Can't Test Quality In

Vis-a-vis QA and QC

- Most developers are unaware of the distinction between QA and QC
- Testing is an essential function of Quality Control (QC)
- Quality Assurance (QA) is the set of activities that review and improve the development process towards producing a quality game

The best games are made by development teams where everyone invests in assuring quality.

Friday, 9 March, 12

I'll simply support this slide with a recommend to explore the distinction further. There is a multitude of literature and webpages devoted to the topic. A quick Google search will start you well on your way

The Cost of Testing

ROI for 'Dummies'

Testing Cost = Preparation Cost + [Execution Cost X No. of Runs]

Manual Test	Automated Test
Low prep costs High execution costs	High prep costs Low execution costs

Friday, 9 March, 12

This is a common equation for measuring the cost of running a test, whether manually or automatically.

A manual test will have a low preparation cost, as the tester need only devise the test to carry out, perhaps as part of a test plan. Acceptance criteria may be determined beforehand, however, we can often trade on the tester's innate ability to recognize a failure and as such, need only concern ourselves with the cost of performing the test.

An automated test differs significantly. There is generally a large initial investment during the preparation phase. A test automation framework will need to be engineered to facilitate the test, a test script will need to be written (and maintained) and acceptance criteria codified, which can be difficult. On the plus side, after all this is in place, we can effectively run the test at little to no cost. Very alluring...but this is where we can get ourselves into trouble.

A Mistaken Assumption

Automated Tests Have the Same Value as Manual Tests

- Manual testing will find knock-on defects
- Manual testing will evaluate the software holistically
- Many tests we wouldn't bother to run manually, yet these are the ones we most often choose to automate
- Your testing should be yielding useful and interesting information at all times

Many automation solutions exist in the absence of any real testing strategy.

Friday, 9 March, 12

1) 100,000 hrs at 20/hr = 2 million in savings!
2) are these good tests? would we have people do it?
3) no equivalency
4) go through points

A common justification made for test automation is that we save so much money in our testing budget by running automated tests. I find this to be a weak and inaccurate argument. The basic gist of the argument is as follows: "We can report 100,000 hours of automated tests run on this project. Our manual testing costs are 20 dollars an hour. That's 2 million dollars worth of testing for just a fraction of the cost!"

Think for a minute about the kinds of tests we automate. Does performing the same action thousands of times in a row constitute a good test? Would you task one of your testers with repeatedly loading the inventory screen for the next 6 months? I'll caution that not all automated tests are useless, but be cognizant that there is no value equivalency between manual and automated tests.

How many actual and unique defects were found? How much manual time was spent investigating them? Would they have been found by testers anyway? Were they fixed before you shipped? What bugs did you miss? These are just a few of the questions that immediately spring to mind.

Sadly, automated testing strategies seem to appear most frequently on projects that are a mess and lack any real testing strategy. If you feel that the work your testers are performing can be easily automated I'd encourage you to examine closely your testing strategy. **Your testing should be yielding useful and interesting information at all times, if it is not, then no amount of technology will fix that.**

There have been entire books written on test strategy development, so it is much too rich of a topic to get into here, however it is important that you have a test strategy in place that works first before you even start to consider automation as added value.

A Mistaken Assumption

Automated Tests Have the Same Value as Manual Tests

Manual Test

Agile
Adaptive
Variated
Holistic
Exploratory
Anticipatory

≠

Automated Test

Rigid
Predefined
Expective
Fragile
Discreet
Determinisitic

A tester's salary is pretty good value, in my opinion, for access to the full capabilities of the human mind it provides. Can your automation system bring to bear that full range of capabilities for a comparable price?

Cost Explosion

Diverted Development Effort and Upkeep

- Development effort is diverted to support automation rather than the game
- Developing an automation framework costs developer time
- Time spent developing the framework is time not spent developing the game
- Don’t forget about infrastructure costs

Your goal is to make a game and you have ~18 months to make it. Don’t forget that!

Friday, 9 March, 12

1) a good framework requires good engineers
2) costs a lot of time - usually throw juniors engineers at it
3) You may have rationalized this effort but the savings are suspect
4) supporting hardware isn’t cheap

Game development is software development at a breakneck pace. If you are lucky, you have about 18 months from idea inception until you need to have the product on store shelves. This is arguably why agile development methodologies are so popular in the industry. Do you have the time and resources to spend developing software that is “not the game”? You may have rationalized the diverted effort by reporting “tens of thousands of hours saved in testing” but this is a suspect measurement, as we just saw.

Upkeep and Maintenance

It's Never Just Run and Forget

- Test automation frameworks have defects too!
- Prepare to have at least one senior developer devoted full-time to supporting the framework
- Games change during development... a lot! Prepare to see your tests break... a lot!

Most defects that your automation might find will be found and fixed, *manually*, while hardening the framework.

Friday, 9 March, 12

1) time spent testing things that are not in the game - most detections were defects in the framework
2) game systems aren't developed with testing in mind
2.5) attempts to replace trivial actions with automation free up junior tester but tie down senior engineer
3) when churn has settled, game has shipped

Maintenance of the automation framework is a huge "gotcha." This can't be overstated!

Conceptually, it seems so easy and alluring. As engineers we observe testing as an obviously onerous and repetitive task that is ripe for automation. Wheels start spinning in our heads, and we start building a system to drive some bots through the game. Piece of cake! Of course, it's never that simple and we soon find ourselves constantly trying to adapt the system to account for all the unanticipated idiosyncrasies of the game. Chances are the game systems were not developed with testing in mind. They rarely are!

What we've seen, time over time, is that our effort to replace some trivial tests normally performed by a junior tester, have instead resulted in a senior technical analyst or, worse yet, a senior programmer regularly babysitting and maintaining the automated tests instead. To be clear, a simple test that can be performed in minutes is replaced with an automated test that breaks frequently and requires hours of investigation and repair.

On Dragon Age this was especially true. First excited at the prospect of being able to automate much of their test work, our senior technical analysts developed extensive automated test suites. Remember, these suites were based on existing test plans - these were useful tests! Yet, to a person, they were all ultimately abandoned before project end. The analysts all reported the same thing: the test broke too frequently and they found they were spending more time investigating the failures than they would spend testing the game by hand. In fact, most failures were a failure of the test not being able to adapt to the rapid-iteration of many of the game systems.

Additionally, any in-game defects were generally found and fixed while developing the test and never returned, making the tests arguably useless as regression tests. It only makes sense to keep a test around as a regression test if it has little to no upkeep costs associated with it, yet as we've seen, in the rapid-iteration based environment of game development this is usually not the case.

We haven't even mentioned the cost of developing a test automation framework. A test automation framework is a piece of software, too. That means it has bugs and needs to be tested as well. We've just written more non-game code that we need to test. Are the potential gains worth it, or are we being self-defeating? An important question to consider.

SO?

New Assumptions

- Quality comes from iteration
- Testing is a small part of development costs
- Manual testing and automated testing are distinct
- Use automation sparingly but effectively
- Defect prevention is paramount

Friday, 9 March, 12

1) Quality comes from iteration - As it's put more succinctly by many of our industry's best developers: "When it's done!" This is how you get quality.

2) Testing is a small part of development costs - Is testing and QA 20% of your budget? 30% or more? That leaves 70% - 80% that isn't testing! Regardless, actual development is usually the primary cost. Just because the programmers may be able to improve bad processes themselves doesn't mean they have the bandwidth to do so; they are busy making the game. The QA Engineering team at BioWare has its hands in all sort of development including the build and resource pipelines. There's opportunity to apply "automation" everywhere.

3) Manual testing and automated testing are distinct - Recognizing the difference allows us to see where automation might actually be useful in our project. If we are not looking at it as something that can replace manual testing then it must exist as a value-add; this can help move us towards being successful with whatever strategy we adopt.

4) Use automation sparingly but effectively - Pick your fights. Delivering a couple of solid wins will add more value to the team (and the game!) than "almost" delivering on an ambitious automation plan. Measure your development process, too. Metrics can help you decide where to focus your efforts. Every game team is unique, maybe testing really is a problem for you so, in that case, ignore everything I say, but you might be surprised where your biggest bottlenecks actually are.

5) Defect prevention is paramount - If you agree that a quality game comes through iteration, then there is no better way to ensure a high level of developer velocity than by preventing defects. Defect detection is secondary.

Quality Comes From Iteration

What Makes Games Good

- You will find more bugs than you can fix, so focus on fun
- Support initiatives that allow changes to get into the game faster and more frequently
- Gamers can be surprisingly tolerant of ‘defects’

Improving developer velocity is like discovering additional time you can use for iteration.

Friday, 9 March, 12

While the topic of game quality measurements could fill another talk altogether, it is important to at least recognize that unlike in other software, games are measured by more than their satisfying of technical specifications. To be clear, unlike say in a spreadsheet application where the software is evaluated based almost solely on the execution of its feature set, a game is judged more holistically and experientially. Your customer is most concerned about how your game “feels”, and your best chance at delivering for this expectation is to spend as much hands-on time with your game as possible during development. This why iteration is key!

If your game is loved by your players, then a defect whereby a player’s vehicle may explode across the map due to a misplaced satchel charge might elicit cheers of delight and a multitude of YouTube playlists. This sort of thing is very common with video games, yet consider a scenario where your spreadsheet application unceremoniously explodes your data across the screen. It’s hard to imagine the users taking that very well.

Certainly, catching and fixing as many defects as possible is important (especially ones that would cause the players special frustration), but don’t lose sight of the fact that your game will be judged by how fun it is, first and foremost. Make sure you target your efforts with this in mind.

You will find more bugs than you can fix, so focus on fun

Support initiatives that allow changes to get into the game faster and more frequently

Time from dev-to-test is more important than test-to-dev!

Gamers can be surprisingly tolerant of ‘defects’

...so long as the game is fun; which is a holistic measurement

Defect Prevention

A Penny Saved...

- Defect prevention and improving developer velocity go hand-in-hand
- Defects are introduced at the earliest stages of development; when the code is written and via the resource pipeline
- Most importantly, it's cheaper!
- Unaddressed defects accrue costly technical debt

"...quality is more an act of prevention than it is detection. Quality is a development issue, not a testing issue." - *James Whittaker, Google*

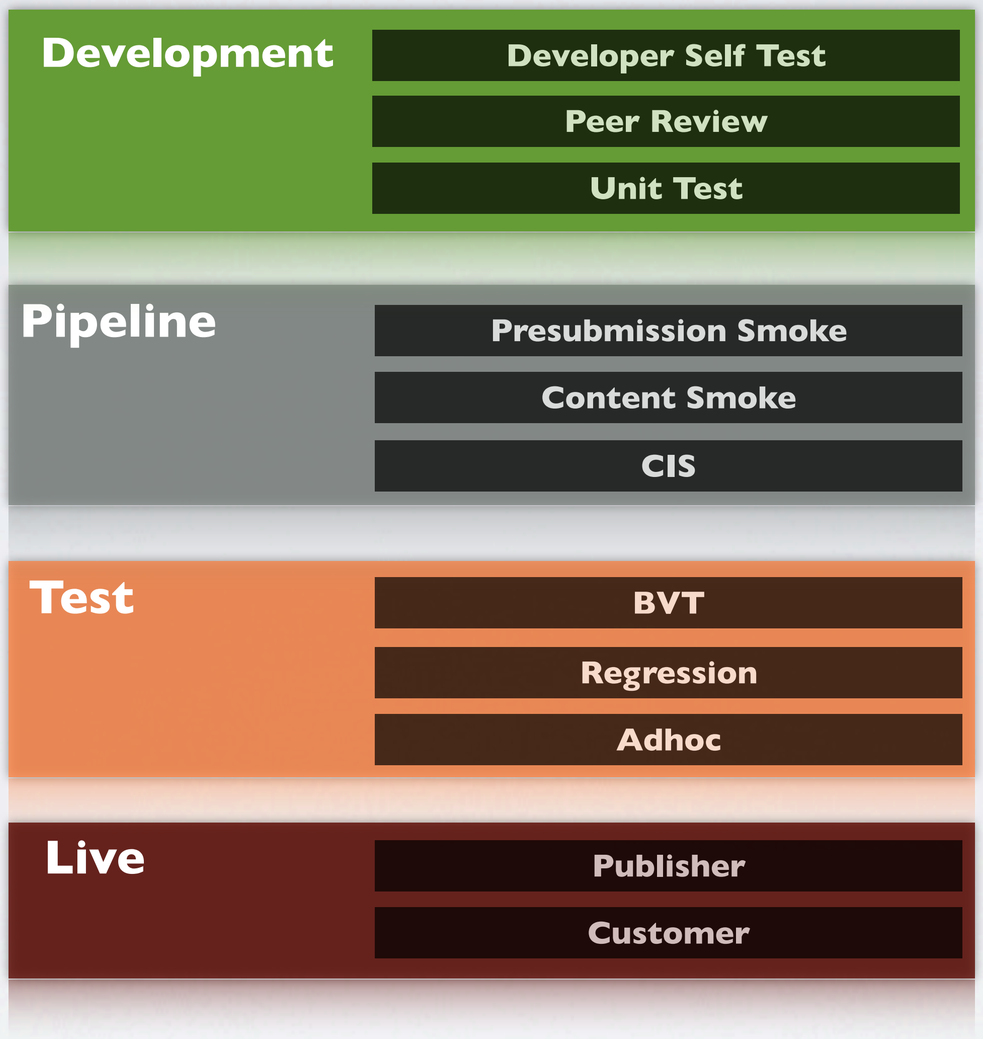
Friday, 9 March, 12

Favour defect prevention over detection!

Intuitively, everybody knows this, but as engineers we tend to "forget" when it is most critical. Don't let a defect trickle down through your development pipeline over a period of days allowing it to impact multiple development personnel and accrue further debt. By the time it gets caught by the testers, you've wasted valuable time that could have been spent making the game better!

A commitment to defect prevention, while looking good on paper, often falls by the wayside as the demands of your deadlines start to exert their toll. Realistically, this will always be the case. That's why it is often best to target your QA engineering efforts towards supporting your development staff, helping to relieve them of some of this burden.

QA Maturity
Increased predictability
Lower cost
Increased developer velocity

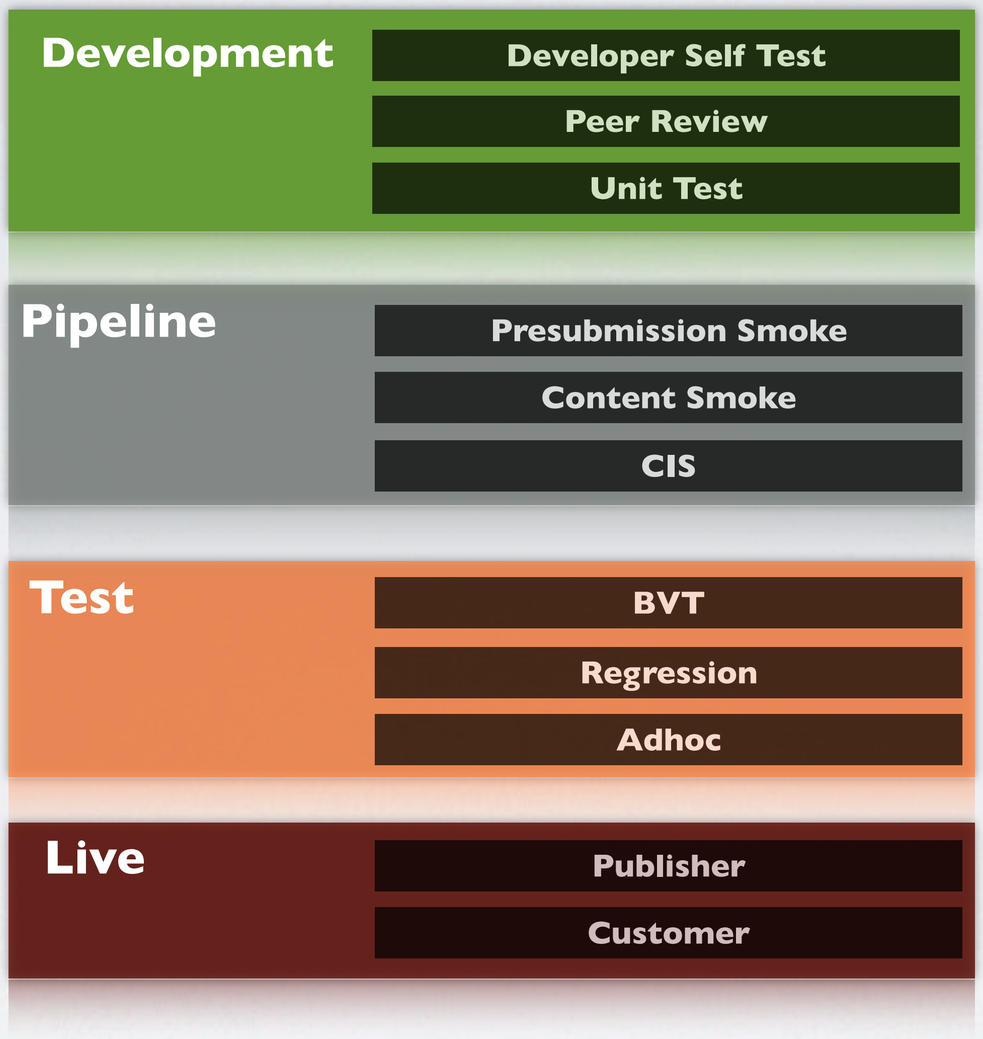


Technical debt
Reduced predictability
Increased cost
Compound defects
Less iteration time

Defect prevention benefit timeline visualization.
Modified from original. Original provided by Philippe Branchu – Electronic Arts.

Catching defects here increases developer velocity across the board.

This is the biggest bang for our buck.



Yet, the tendency is to focus our engineering effort here.

Defect prevention creates iteration opportunity.
Remember, iteration creates a quality game.

Tester Augmentation and Process Automation

Building 'Rock Stars'

- *Tools assisted testing* **will often yield more value than** *test automation*
- **Expand your testers' capabilities**
- **Find processes that can be supported with technology throughout the studio**

"...creating an infrastructure to make test analysis easy for humans is much more effective than trying to engineer tests to automatically 'understand' the UI." - *Patricia Legaspi, Google*

Tools assisted testing **will often yield more value than** ***test automation***

- *Support testers with technology that makes them more effective*

Consider expanding a tester's capabilities

- *Can you deliver a technology to let a single tester perform multiplayer testing?*

Look for processes that can be supported with technology throughout the studio

- *Test is just one part of development; you can inject quality anywhere!*

Little Automation, Big Effect

Don’t Overreach; Focus on Being Successful

- **Automation is a big investment**
- **Set modest goals that will return meaningful results within the project’s scope**
- **Determine what challenges your particular project faces and target those**

Friday, 9 March, 12

Automation is a big investment

If you are not prepared to fully commit the resources necessary, you are better off focussing your effort elsewhere... like on the game!

Set modest goals that will return meaningful results within the project’s scope

Experience shows that favouring development over test will get you the biggest wins!

Determine what challenges your particular project faces and target those

Your major risks might not be what you think they are!

Some questions to consider when developing a tool or automation strategy:

- **How long will the new tool or process be useful?** e.g. if the project ships in 2 weeks or the engine is being retired soon, it may not be worth building. If this is something you will use for several months or across multiple projects that is a good sign.
- **How much effort will be saved by the new tool?** E.g. if people would discover the problem anyway as part of their normal work, are you really saving anything? If this uncovers issues long before they’d be noticed, that’s probably useful.
- **How complex is the task?** If there are a small number of distinct error causes/messages, this may be a great candidate for automation. If the issue is generating a large number of different errors it may be more difficult to automate.
- **How much would it cost to perform the task manually?** Often tasks that are only done over small windows of time or that would be very complex to automate are good candidates for manual execution.
- **What will be done with the tool’s output?** If the output isn't going to be read or used there is no point in doing the project.
- **How long will the tool take to develop?** This also depends on the complexity of the task.

Automation Axioms

Four Rules to Live By

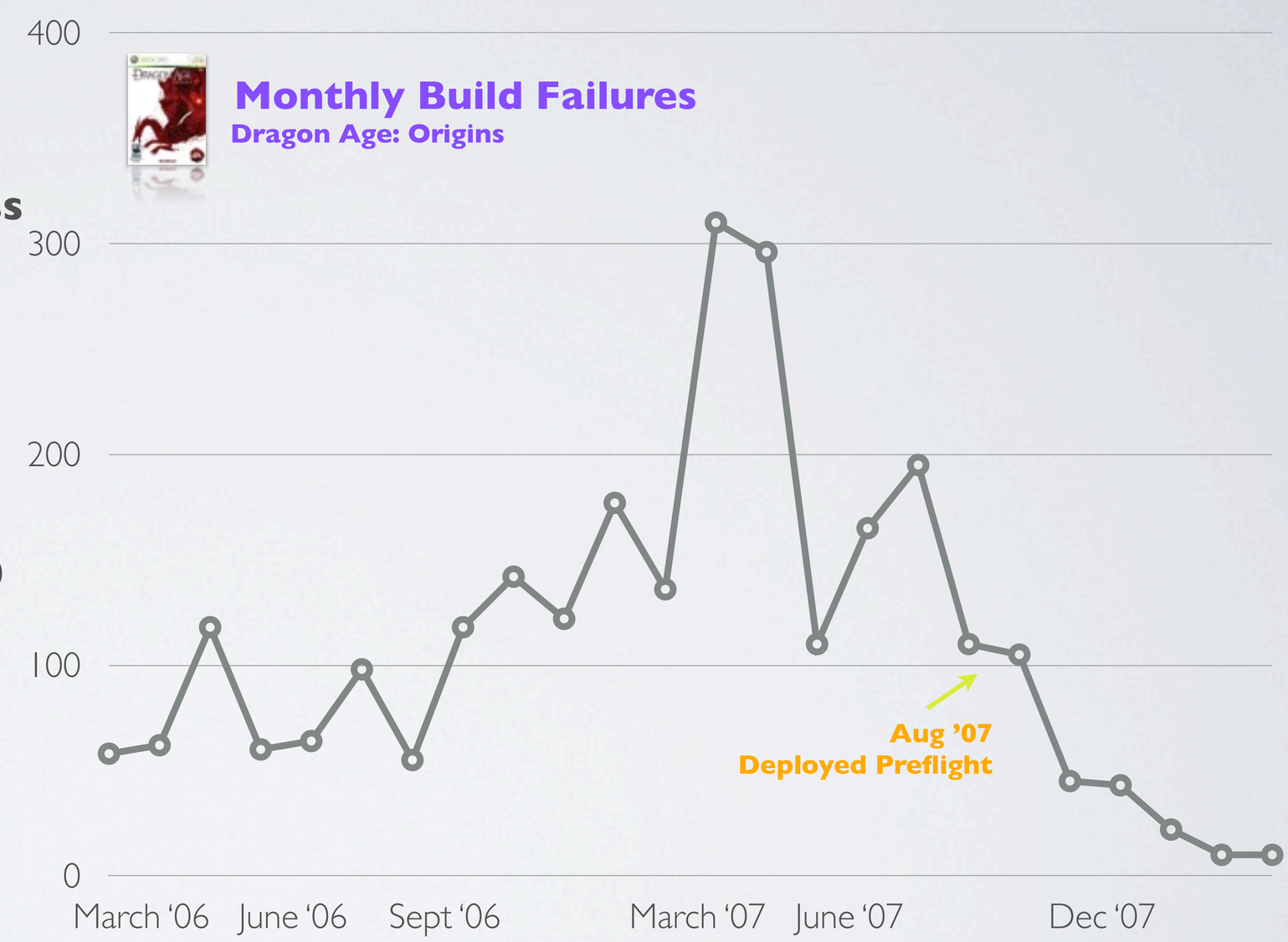
- Automate processes, not people
- Automate tasks that people cannot perform
- Favour early stages of development
- Augment manual testing; don't supplant it

A COUPLE EXAMPLES

Preflight

Presubmission Code Checks

- **Automated testing changelists across multiple SKUs and configurations**
- **Stopped bad code from reaching the CIS**
- **No. of monthly broken builds decreased from ~300 at peak to < 30**
- **Devs and QA always had a working, up to the minute, build**
- **Significant increase in developer velocity**



Friday, 9 March, 12

Here's an example of a solution that clearly satisfies axioms 2) and 3), by favouring early stage development and performing a task a person couldn't otherwise be expected to perform.

During the development of Dragon Age: Origins we encountered a situation where the number of monthly build breakages was increasing astronomically. This had severe negative impact on the team and developer velocity was ground to a near stand still. Consider a situation where you have a large project with maybe 60 programmers. If each developer only breaks the build once a month, that's still 2 broken builds on average each day! It's not hard to see how a small problem can be amplified.

In order to combat this, we developed a presubmission automated code checking solution. Rather than submitting their code directly to source control, developers would submit it to the Preflight system that would distribute the task of building and executing the code across multiple SKUs and platforms. If it passed the most basic of smoke tests, the changelist was automatically submitted into source control, if it failed, it was bounced back to the developer to fix. This ensured that we never had broken code making it's way into the CIS. In fact, the only broken builds we experienced were when the preflight system was circumvented, so we effectively reduced the number of broken builds to zero!

Ensuring we always had a working build meant everyone was able to do their job. Furthermore, they were always working with the bleeding edge latest code, so development iteration was at its peak!

B4Bug

Transparent Bug Filing

- **Single-click bug filing system**
- **Every bug is filed with all the essential information needed by developers**
- **Ensures a minimum bug quality**
- **Testers keep their 'head in the game'**
- **Non-testers file bugs too... because it's easy!**

Friday, 9 March, 12

Here's an example of a solution that clearly satisfies axioms 1) and 4), by supporting a process and augmenting manual testing.

The B4Bug solution was first developed after Jade Empire shipped for an unannounced title and was later ported to Mass Effect and Dragon Age: Origins.

Providing developers with the information necessary to resolve a bug can often be problematic. Even if the tester remembers to include all of the pertinent information, that doesn't ensure it is always correct. How many times have you failed to repro a bug because the build number was incorrectly specified, or how many times have you returned a bug as "need more info" because it didn't include a screenshot? B4Bug ensures that every bug has screenshots, accurate information and more. Furthermore, all of this comes with a single press of the 'B' key (where the tool gets it's moniker), requiring that the tester only title it and add some repro steps. This has the added benefit of keeping your testers focussed on playing the game and not on the accompanying "paper work".

The talented team at Electronic Arts CDS have since productized the B4Bug solution for the broader organization, and while we no longer use the tool directly at the BioWare Edmonton studio, we continue to employ a one-button bug filing solution as part of our telemetry solution (adding even more pertinent information to the bugs!).

Final Thoughts

Caveats and Further Considerations

- **Absolutely, test automation can have value, but...**
  - *...make sure you have a solid test strategy in place first!*
- **Understand your project's needs and target your automation accordingly**
  - *Test is probably not the top priority.*
- **Game quality measurements, ROI calculations, and developing test strategies are *huge* topics worthy of further examination**

And Finally a Word About Data-Assisted Development...

- **Development telemetry is critical to supporting your efforts**
  - *Much of what benefit we hope to get from automation we get more easily from telemetry*

Contact Info

twitter: @ProjectZen

e-mail: alexl@bioware.com