TECHNICAL TESTING

Technical testing is traditionally the job of professional playtesters in the gaming industry: testing the game to make sure it works. Its primary goal is to verify that the game meets its specified design requirements, and runs correctly.

- The game installs properly.
- The game runs smoothly, with a good frame rate and does not crash.
- The game runs on all expected platforms (hardware, software) and with a reasonable number of background programs open (chat windows, music software, office software, browsers, etc.)
- All features work as intended, all art assets appear correctly and all sounds and music play correctly.
- The game meets all required standards, legal and otherwise.
- The game does not have any unexpected or gamebreaking behaviors in it.

While this sounds short and succinct, beware. There's a lot of work lurking in those bullet points, and plenty of testing to be done so that the development team finds the bugs before the users do. Good technical testing means using every feature, touching every asset, finding every way the user can possibly do something - anything - in game, and doing it multiple times.

THE TESTING PLAN

To do thorough testing of any sort requires a plan. You need to know what in the game needs to be evaluated, *how* to evaluate it, who does it, and an estimate of *how long* each piece should take. A good plan also allows the QA lead to track what has been done, so that the most coverage is possibly in the least time. Finally, a testing plan should prioritize the evaluation tasks, so that if something must fall off the testing cycle, it's checking the spelling in the design document, not the overall project stability.

Every milestone the test plan should be reviewed and updated.

When a test plan first gets made, all the data it wants doesn't exist yet. For example, if the script hasn't been written, there's no way to know what audio files will exist! If the art asset list hasn't been finished, it can't be checked. Also, some checklists don't make any sense for some games - if the audio in a game consists of a single soundtrack played in the background, there's not much point to having an audio checklist. All it really needs is a checkbox on the overall feature list: does music play, y/n? On the other hand, a game with a complex vocal script where conversations with the in-game characters depend on what actions a player took earlier needs a lot more detail!

ELEMENTS OF A TEST PLAN

1. Plan of attack - roughly how much effort should be divided between technical testing, free play, checklists, user testing, and focus testing.
2. Communication/Reports: What does the team want out of the testing? What does the product owner want to know? How are you going to get them that data? What tests do you need to do for it?
3. Make checklists.
    - Know what's on the GAMBIT checklist for each milestone - you can't pass a milestone without passing that checklist!
    - You should also be creating checklists for your game - at the very least, a Feature Checklist and a Hardware/Software Checklist.
        - Feature checklist needs to include all the working features in your game, and ideally

will cover all menu and screen functionality as well as all assets (art and sound.)
    - Compatibility Checklist should detail all the software the game is expected to run on/with: OS's, Browsers, background programs such as music player, IM programs, and office programs.
4. How hard will it be to test your game? Which features will be quick, and which ones will take a lot of time?
5. Focus Testing -
    - What do you want to learn from your focus test?
    - Who should you be using as your focus testers? (and)
    - How are you going to get them into the office?
    - When is it a good time to do user testing with in-house testers (not your primary audience, but at least they serve as a good
    - Borrowing team members from another project can be done on short notice, and makes a great way to do a quick double check of an idea or implementation - just remember everyone else has a busy schedule too.
    - Arranging for naïve testers takes time! Post posters, send out emails, or otherwise start your recruitment drive well in advance of the planned testing time/date.
6. Release Testing: ***A successful release is, in the end, the most important part of QA's job. Everything you do leads up to this: Releasing a game that is easy to run, play, and use.*** You're responsible for finding another QA lead to verify your milestone releases, and verifying at least one other project's milestone. Make sure you're not wasting anyone's time.

MAJOR AREAS OF EVALUATION (WHAT AND HOW)

- Design - Confirms the game is playable and fun!
    - Overall playability
    - Game balance
    - Individual features - do they work? Are they fun? Are they functioning to spec?
    - UI evaluation - making the UI as intuitive as possible, from the pop up window that asks "Do you want to install" all the way down to how the player controls his actions in game.
- Documentation
    - Includes reviewing design documents, ensuring they are followed in game, and updated as necessary - may even include updating documents.
    - Does the game need documentation? For what? How should it be packaged?
    - Confirm any documentation (whether hard copy or a text file) associated with the game is correct and useful.
- In game text is accurate, useful, with correct spelling and grammar.
- Manuals, installation instructions, release descriptions, etc
- Technical - Ensures that the game plays as designed.
    - Confirming that every feature is present, and works according to spec.
    - Game is free of defects/unexpected behaviors
    - Game stability (no crashes, freezes, etc.)
    - Hardware/Compatibility - Guarantees the game runs where it is supposed to.
- Game runs on all expected platforms (Mac OS, XP Pro, XP Home, Vista)
- Runs on various browsers: Safari, Explorer, Firefox, Opera, Chrome
- Game runs with any expected software/add-ons (Music players, chat programs, Office programs, etc.)
- Works with any/all potential peripherals
- Installs and uninstalls cleanly
- Regression Testing

        - Once a team member reports that a defect has been fixed, and a new build made available with the fix, the defect needs to be tested and confirmed as fixed.
            - It is best for a defect to be regressed by the original locator.
        - Regression testing is also used to confirm that previously working features (for example, features done in a previous sprint) still work.
- Release
        - Ensures the release build is stable, and can be started, exited, installed, and uninstalled cleanly.
        - Confirms that all features expected to be in the release are, and are functioning as expected. If a feature is not fully implemented, that should be documented.
        - Testing the physical release - be that a DVD, a cartridge, an .exe file uploaded to a clean computer, accessing a game via the Web, etc. - in the exact same methods that the intended recipient will use.
        - Makes sure part of the release includes documentation as to all features present in the build, especially those which are new for this release, and includes clear and detailed instructions on how to install and run the build.

A FINAL WARNING

CLASSIC QA MISTAKES, PART 1: TESTING COMES AT THE END

Because QA tasks are dependent on the development team plan, it is easy to assume 'QA comes at the end.' Don't fall into this trap. The later formal project evaluation starts, the more serious the problems are, the longer they take to solve, and the worse off the project will be. On the first day of the sprint, the team is doing something, and the QA Lead is too - unless the test plan is still being created. By the second day of the sprint, it is time to put the test plan into action.

Is someone writing a design document? Read what they've wrote, think about it, ask questions, point out parts that don't make sense, aren't consistent, won't make for good game play. Is there an audio script? Review it. Read it out loud. Comment on it. Did a pile of audio files just get delivered? Listen to them. Did a new feature just get checked in? Use it. Did new art just get put in the game? Go look at it. And, like all the other developers, track what has been checked and what hasn't been checked. Talk to the team, find out what they are working on, and if there is an area they are worried about.;

CLASSIC QA MISTAKES, PART 2: IT'S THE QUALITY ASSURANCE LEAD'S JOB

Overseeing testing is the Lead's job. Actual testing, however, is the responsibility of everyone on the team. Code needs to build and run before it is checked in. Assets - be they art, audio files, or UI screens - need to be finished, in the correct place, and ready to be integrated into the game with a minimal (and standardized!) amount of effort. There should always be a usable build checked into source control. Would a small coding feature make the testing process significantly faster and easier to do? Talk with the team, and see if it can be added to the schedule.

CLASSIC QA MISTAKES, PART 3: STOP FINDING (SERIOUS) BUGS, OR WE'LL MISS THE RELEASE!

Teams that make mistake #1 and mistake #2 frequently follow them up with the final error of assuming that they should run out of bugs just because they are running out of time. Alas, it doesn't work that way, and no amount of wishing is going to make an unstable project stable. When the serious bugs are found and corrected as they occur, releases can happen on time. If the problem can't be corrected, it can at least be isolated, and the feature backed

out so that an acceptable version can be released. Developer who wait to fix a serious problem until the last minute often find yet another serious problem lurking behind it - either because testing was blocked by the defect, or because another problem has been unmasked by the fix.