http://www.cs.pomona.edu/classes/cs181f/supp/relphases.html:

Release Phases and Criteria (cs.pomona.edu)

Contents

1. Introduction

There are several terms to describe system releases, and many processes to determine whether or not a product is ready for release. Ultimately, the goals for a particular product and release are a business decision; The release criteria should follow from those goals, and the release descriptions will be twisted to show the product in the best light.

Such relativism would seem to preclude precise definitions of release phases and the criteria they should satisfy ... and indeed definitive precision is not possible. There are, however, some commonly used terms, and it is possible to discuss what they are commonly inferred to mean.

Similarly, while there may never be a set of universal release criteria, it is possible to discuss the types of criteria that are commonly used, and rational processes for developing specific criteria for a particular product and release.

2. Release Taxonomy

2.1 Release Maturity

There are a few terms that are regularly used to describe the maturity (functional completeness, correctness, and confidence) of a release. The most common names for formal releases are:

- **Alpha**
  An alpha release is usually the first version of a system (or subset of the system) that it makes sense to put into the hands of potential users. It is to be expected that alpha software will be incomplete, buggy, and may experience significant change before it is released in its final form.

  Typical goals for an alpha release are:

  1. to gather feedback, from intended users (usually key customers), on newly developed features.
  2. to give partners early access so that they can start developing software that will complement or work with ours.
  3. to (re)establish credibility by showing how much progress we have made.

  Alpha sites usually require considerable support, and they are usually chosen very carefully:

  - we can depend on them to exercise the software and provide us with high quality feedback.
  - they are technically prepared to deal with the expected problems.
  - we can trust them to not share our software with our competitors or evaluators.
- **Beta**
  A beta release is much closer to what we expect to ship. They are usually expected to be fairly complete, and to have relatively few problems.

  There are a few schools of thought on the goals for beta releases:

  . We have a product that we believe is ready to ship, and we send it out to beta customers to confirm that it is ready before we move on to general availability.
  a. We have a product that we believe to be in final form and want to make it available to our partners as quickly as possible (to support their development of complementary software).

b. We have a product that we belive is not sufficiently stable for general use, but we want to release it anyway (so that we can claim to have made our date, to book some revenue, etc). In these cases the term beta is primarily a warning label to potential customers.

If the purpose of the beta release is to gather feedback, it is likely that the beta sites will be selected and managed almost as carefully as alpha sites. If the term beta is merely a warning label, the release may be made available to anyone who wants (or purchases) it.

- **General Availability**
  A general availability (GA) release is one that we believe is ready to be sold to all of our customers. Such a release may also be referred to as FCS (**First Customer Ship**) or **Revenue Release**. These terms distinguish paying customers (who purchase the product to use it) from evaluators (who may get the product for free in return for their feedback).
- **development build**
  All of the preceding terms refer to a formal release that might be the result of many months of planning, preparation and testing. When you get a "release" you should have reasonable expectations about what you will receive.

  Some products also make regular (e.g. weekly or even nightly) development builds available. Unlike releases, development builds may not have gone through a full gauntlet of system acceptance tests. These may be very popular with partners (who are tracking specific changes) but not commonly used by general users (who generally prefer the added packaging and testing that is done with formal releases). This should not, however, be construed to mean that all development builds are flaky. A build is as good as the people, organization, and processes that created it.

2.2 Release Incrementality

In addition to describing the expected completeness and stability of a release, there are also a set of terms for describing the degree of change that is introduced in a new release. Different organizations use different degrees of precision and formality in the application of these terms, but a reasonable set of expectations might be:

- **major release**
  A major release (often indicated by a change from 2.7 to 3.0) indicates major changes in the software, that may not be upwards compatible with previous releases.
- **minor release**
  A minor release (often indicated by a change from 2.7 to 2.8) indicates minor additions, enhancements, and bug-fixes to the software. While new features may be added, there is a reasonable expectation of upwards compatibility with previous releases. New versions of software should be able to process the output of older versions, and should still support all of the same features. It may, however, not be possible to take files created with the new versions, and process them with the older versions.
- **update release**
  An update release (sometimes called a point release and indicated by a change from 2.7 to 2.7.1) is usually expected to include only bug fixes. When installing a point release, we expect improvements in quality, with no changes in functionality ... and certainly no incompatibility issues.
- **patches**
  If a bug is sufficiently serious, the development organization may make updated versions of particular programs available immediately, rather than waiting for the next release.

In most major operating systems, a patch is actually a mini-package, containing only a few changed files. In days gone by, however, it was actually a script or program that would apply specific binary corrections to specified files (not unlike putting a patch on a worn pair of jeans).

2.3 Other Release Related Terms

- **golden master**
  A "golden" image (typically of a CD) is the one that will be sent to manufacturing for reproduction (or put on the web for down-loading).

  It should be (as close as possible) to identical to the last system that passed system test, but we will test it once again to make sure that there has been no regression. After it passes that test cycle, we will give manufacturing the go-ahead.

- **Release Engineering**
  This term describes an activity or a group of people who perform the activity. Their role usually includes:
  - starting with a clean official build environment,
  - extract the official versions of everything from the version control system.
  - do a complete product build, using standard build procedures.
  - organize the build software into **packages**.
  - create ready to install media (or downloadable images).
  - verify that the created media installs and passes a smoke test.
  - publish the new build to all interested parties.

3. Release Criteria

About 30 years ago, Orson Welles, as a celebrity spokesman, made famous the Paul Mason motto:

> *We will sell no wine before its time.*

This reassured premium (jug) wine purchasers that they were making the right decision, without every actually stating how Paul Mason was making the critical determination of whether or not a wine's time *had come*. I, personally, don't know enough about wine to be able to assess the verity of Paul Mason's claim. I do, however, know a little bit about software, and I am pretty certain that many of the products I have used were sold well before what I would have considered to be *their time*. The sad fact is that, while many products are carefully *released*, a distressing number of software products *escape*.

While it is (fundamentally) impossible to attempt to define universal criteria for software readiness, release criteria can be defined rationally, and on the basis of objectively ascertainable information.

3.1 Goals

When is software ready to ship?

> When we have *adequate* confidence that it will achieve *our goals*.

Thus, any rational discussion of release criteria must begin with a clear statement of our goals for the release in question. Releases can have many different goals. Typical examples might be:

- To provide new functionality to existing customers with no disruption to their work processes.
- To add a new features that will enable us to gain new customers.

- To enable partners to start developing new functions that will be based on our software.
- To get new functionality into the market in time to capitalize on current interest, and to gain mind-share by being the first to release it.
- To get a prototype of a new concept into early adopters' hands to gather feedback and generate buzz.
- To silence competitors who are telling potential customers that we can't do this.
- To avoid being fired, which is what we have been promised will happen if this release is not on time.

Obviously many others, and combinations, are possible. The above list, however, should be sufficient to illustrate how wide the range of release goals might be. There is a comparable range of release readiness criteria that might be implied by each of these goals.

It is tempting to state general and noble goals (because they make us look noble and visionary), but it is important to be as honest and specific as possible. Release goals are the highest level of requirements. Most of the other requirements we gather are simply trying to understand the necessary conditions to achieve our goals. Like any other requirements ... if we state them incorrectly, they will lead us to specify and design the wrong product, and to make many other "wrong" decisions along the way.

Some of our goals may be sufficiently venal that we may not choose to widely publicize them ... but that doesn't mean we should ignore them. Clarity on our goals is a prerequisite for a good plan to achieve them. Ambiguous goals and decorative non-goals will give rise to bad plans.

3.2 Common Types of Criteria

Once we understand what our goals for the release are, we can start trying to articulate necessary conditions for success. These conditions typically involve some combination of functionality, quality, and delivery time. Many of these conditions will be shaded (with some things being absolutely required, some being highly desirable, and some being merely nice-if-possibles). We may find that we can finely sub-divide functionality, and assign different priority, quality, and time requirements to each of the sub-sets. Ultimately this should give us a set of requirements that can serve as the basis for release criteria.

Functionality and date requirements are easily specified and tested. If the functionality is reasonably well specified (which it needs to be before we can build it), it should be easy to ascertain whether or not the product provides the required functionality. When we can state what we want in measurable terms, life is good. Where we get into trouble is when we do not know how to state what we want in measurable terms ... which is often the case with quality criteria:

- quality is, for the most part, not directly measurable. Thus, we are forced to propose surrogates, whose correlation to quality may be (at best) poorly understood.
- we would like to express quality requirements in terms of expected customer experience, but we cannot measure customer experience until after the product is in customer hands. Thus, we are forced to use predictive measures, whose predictive power may be (at best) poorly understood.

The challenge then is to find:

- measurable product characteristics
- that we believe to be well correlated with our goals
- that we believe to be valid predictors of customer experience
- and rephrase our quality requirements as quantitative statements in terms of these surrogate measures.

There are several metrics that are commonly used in quality criteria:

- known defects
  It is common to define release criteria in terms of known open bugs ... since these give direct testimony to the quality of the product. A typical set of such criteria might be something like:
  - no priority 1 bugs
  - no more than 10 priority 2 bugs, each explicitly approved by the product owner.
  - all severity 1 and 2 bugs require explicit approval from customer support
- defect discovery rates
  It would be naive to assume that the only defects in the product are those that we already know about, and there has been a great deal of research into ways to predict the number of remaining (undiscovered) bugs. One of the most obvious predictors is the recent discovery rate. If we are still discovering a hundred bugs a week, it is a safe assumption that there are many bugs yet to be found.
- quality assurance process
  If we believe that requirements, architectural, design and code reviews to be good sources of confidence, we may require those processes be completed at the appropriate points in the development process.
- tests passed
  If there are known test suites of proven value, we may specifically enumerate the suites that must be run and what a passing score is. In the absence of known validation suites we could specify criteria such as "successfully passes all required test cases specified in the test plan" ... but this merely punts the definition to someone else, who still has to figure out what test combination cases would give us the confidence we seek.
- code coverage
  Where no existing test suite is already known to be adequate, some projects impose code coverage requirements on the new test suites to be developed. These may be stated in terms of absolute numbers (e.g. 100% statement coverage, 95% path coverage), or it may merely be stated that achieved coverage must be approved by a specified review process.
- hours of testing
  For exploratory, usability, or load and stress testing, our confidence increases with more hours of testing. It makes sense, therefore, to require satisfactory performance to be demonstrated for a minimum number of hours or sessions.
- feedback from limited availability trials
  One reason to do beta testing is to confirm product quality before putting it out for General Availability. It is common to require x weeks of beta testing with y sites and a minimum feedback score of z% before a product can go to General Availability. We may have internal testing requirements (e.g. running on 200 internal desktops for three months) before we can go to beta.

Some of these criteria are incomplete, and empower others to make some of the decisions:

- the process of bug prioritization is seldom deterministic, and trusts people to diligently act in good faith in prioritzing and evaluating bugs reports.
- deferring test case definition to a test plan gives another body of people the authority to determine what tests are required to determine whether or not the product is ready for release.
- there are good reasons for setting different code coverage criteria for different modules, but we trust the people who make those determinations to do so carefully and deliberately.

Such flexibility is necessary because it is impossible to fully specify (in advance) all of the criteria that must be satisfied. It makes sense to set general expectations at a high level, but many of the details are best worked out on a case by case basis. There are also risks associated with such delegated decision responsibility. People who have an interest in shipping (as opposed to achieving release goals) may be motivated to under-prioritize bugs or approve inadequate test plans. There are certainly cases where release criteria are abused.

It is very difficult to design a set of criteria that provides adequate flexibility to deal with unknowns, while preventing abuse. Also, the sad fact is that much abuse is actually encouraged. People are quite perceiptive, and quickly figure out if an executive is paying lip-service to quality while focusing on date. Ultimately it comes down good people, and clear and honest communication.

3.3 General Considerations

Like most other project plans, release criteria are living documents, subject to continuous evolution. The fact that they are subject to change, however, does not mean that it does not matter what we start with. We should work hard to define clear and reasonable release objectives, and release criteria that will testify to their satisfaction. As they evolve they should become more specific and better justified. Arbitrary changes (inevitably in the direction of lowered expectations) must be critically scrutinized.

It is critical that release criteria be established early in the project, when people are focused on goals and plans. If we wait until we are near our proposed ship date to establish the release criteria, there will be strong pressure to set the bar at "wherever we currently are".

Setting criteria that will only be measured when the final ship decision is to be made is a formula for disaster. The criteria should be pushed (in graduated steps) back to earlier milestones, so that acceptability is measured throughout the entire development process, and we are able to continuously assess our progress towards our ultimate release criteria. This makes it possible to monitor progress and detect problems as early as possible, giving us more opportunity to recognize and respond to those problems, and a greater likelihood of success.

3.4 When Push Comes to Shove

There are very few absolutes in this life, and difficult decisions are seldom made in complete accordance with written guidelines. Often, when a ship decision is made, there are arguments for shipping and arguments against it ... and neither set of arguments is decisive. If they were, the decision would be clear.

Ultimately, the decision to ship or wait is a strategic risk management problem. Do the expected positives outweigh the expected negatives? This decision is usually left to the product owner. Not the engineers who built it, not the managers who oversaw its development, not the people who have evaluated it, but the person who is responsible for the business unit, service, or laboratory.

Everybody will want to influence the decision maker, but what they really need is not "decision advice". What the decision maker needs is:

- the best possible understanding of the state of the product.
- the best possible assessment of the practical implications of the known issues.
- a good set of options, with clear costs and risks for each.