Brainy Data Development Life Cycle

Contents

1. Introduction

In order to remain responsive to our developers' needs, we have always deviated from traditional development life cycles that limit new features to major releases. However, as our software became more complex and the use to which our software is put became more diverse, we have had to re-evaluate how we implement feature enhancements and software corrections based on the level of risk. Consequently, we have put in place a new system for releasing software based on a wider analysis of risk.

This document details our new policies regarding our development life cycle and our level of quality assurance. It will answer questions such as: What are the release modes? What are the levels of risk for each release? What are the limitations of our quality assurance? And what releases are covered by your maintenance versus support subscriptions? It will also show how we schedule new releases.

We hope this document clarifies our policy position and addresses all your questions. However, should you have further queries or comments don't hesitate to contact us.

2. Modes of software releases

There are four modes of software releases and these are reflected in the syntax of our four digit version numbers, as in A.B.C.D in the table below.

	A. Major i.e. **4**.0.0.0	**B. Minor** i.e. 4.**1**.0.0	**C. Maintenance** i.e. 4.0.**1**.0	**D. Patch*** i.e. 4.0.0.**1**
New Features	Major & minor	Minor only	none	none
Fixes	Major & minor	Minor only	Minor only	Minor only (single)
Alpha releases*	Yes	No	No	No
Beta releases*	Yes	No	No	No
Preview releases*	Yes	Yes	Yes	On request
Risk	Major	Minor	Minor	Isolated to minor
QA Testing	Comprehensive	Localised	Localised	Specific to localised

* these early access releases require one or more technical support subscriptions

In terms of dates, there are no fixed release cycles. As to when each mode is released depends very much on the demand and our progression through the development life cycle.

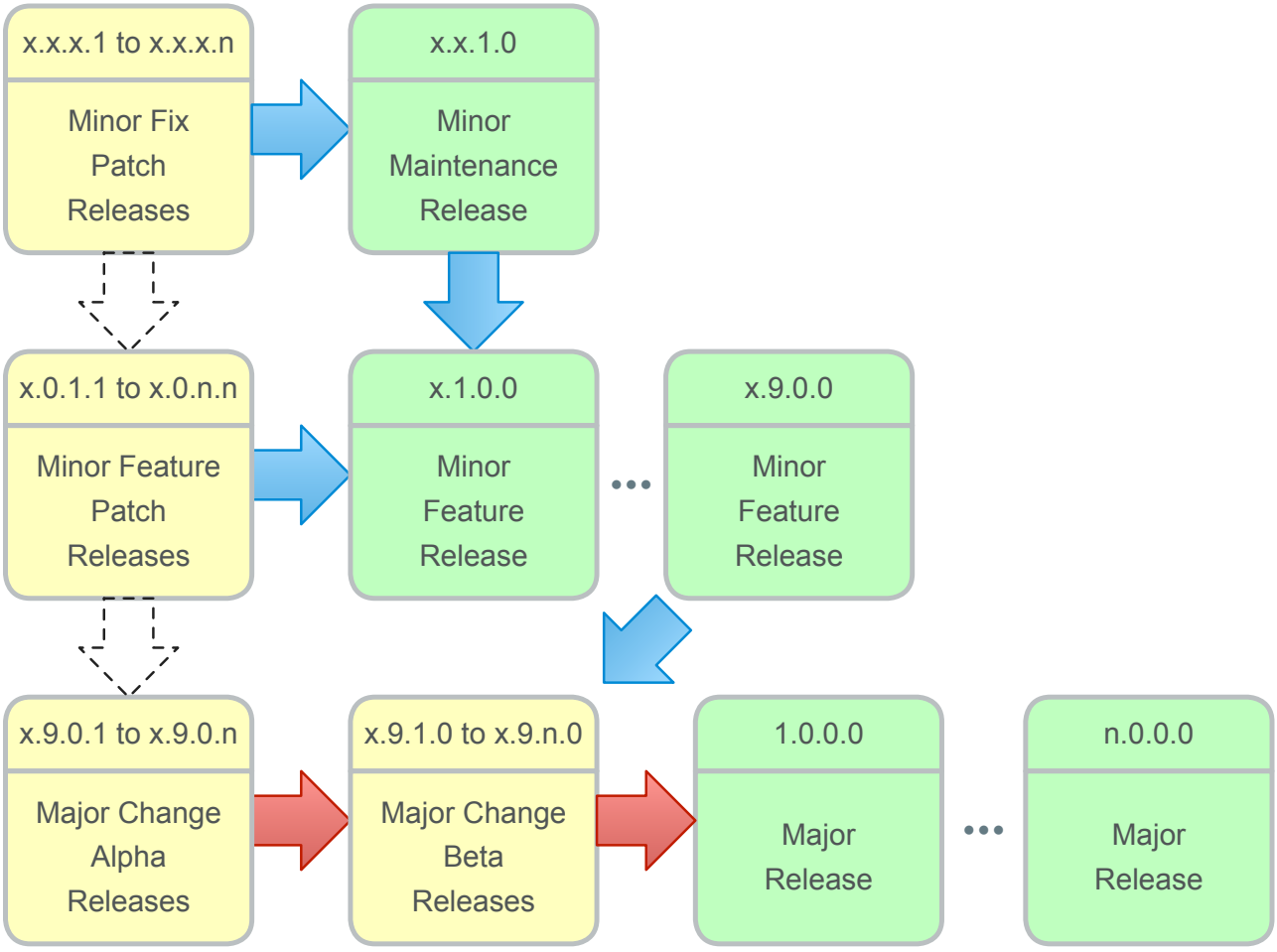
However, in terms of order of modes, each development life cycle begins with patch releases (D). A patch release is typically an advance through a single software change. Thus changes accumulate with each patch release. Patch releases are only available to developers with appropriate maintenance and technical support subscriptions.

As patch releases accumulate, the accumulated patches will amalgamate into a minor maintenance release (C) and subsequently, with minor features added, into a minor feature release (B). This order is intended so that a stable C release always precedes a B release. Maintenance and minor feature releases will be available through early access to developers with appropriate maintenance and technical support subscriptions. Following early access, B and C releases are made available to all developers with appropriate maintenance subscriptions.

Alongside the B, C and D releases, the development of major fixes and features will be carried out in a branch of our source base dedicated to high-risk changes. Builds of this branch will be made available as early access through alpha (A.α) and beta releases (A.β) to developers with appropriate maintenance and technical support subscriptions. Following the beta cycle, A releases will be made available to all developers with appropriate maintenance subscriptions.

Because development of our A releases is more long-term, B to D releases may supersede A.α or A.β releases. Consequently, B to D releases may spawn new A.α or A.β releases to absorb the latest minor fixes and features.

The following image outlines our development life cycle.



The light-yellow boxes indicate pre-release versions that require technical support. The light-green boxes indicate main-stream release versions that are available to all who have the appropriate maintenance subscriptions. Red arrows indicate the high risk development path. The dashed arrows

indicate that some changes from patch releases may filter down to alpha releases, although, beta releases will always include all minor changes of all prior versions.

As can be seen in the diagram above we may be working on as many as three code branches at any one time.

3. Features and maintenance ranking

In which code branch a requested feature will be implemented or a reported software issue will be addressed, depends on the request ranking. When a request is submitted via our online form, you, the developer, may assign an initial ranking. However, following our initial investigation, we may adjust this ranking based on criteria such as the risk involved as perceived by us and the urgency of the request as perceived by your users (i.e. the frequency and the pervasiveness of the issue).

Low-risk urgent cases are likely to be addressed within early access patch releases, whereas high-risk non-urgent cases are likely to be addressed for the next alpha or beta versions of the major release branch.

More specifically: Isolated new features that can be turned on or off and do not alter existing functionality pose a low risk and are likely to be addressed in the minor streams. Equally, fixes that are very localised and only impact minor features will also be addressed within minor streams. Whereas, a change that impacts core behaviour will be addressed within the major stream. However, a serious issue affecting many end-users may be dealt with within a minor stream even if it is risky. Such a critical high-risk change when applied in a minor stream will, in all likelihood, move from patch release to maintenance release in isolation of other changes.

4. Early access releases

As I have already mentioned, early access releases (patches, alphas and betas) are made available to developers with technical support and appropriate maintenance subscriptions. The purpose of early access releases is two-fold. Firstly, they are a targeted way of distributing fixes and features to developers who requested them. Secondly, early access allows developers to test the changes within their own applications and have compatibility issues addressed prior to the actual release. Thus, early access provides an invaluable mechanism to ensure fast and safe implementation of requested fixes and features.

Providing the added value of early access is costly and consequently requires a technical support subscription. If you employ or work within a larger programming team, we recommend that you subscribe to our TEAM or PREMIER technical support subscription as it will allow you to register multiple technical support contacts to cover periods when some developers are absent or to cater for multiple development streams that involve our software. Providing support for larger teams is more costly and these higher value subscriptions will go some way to meet these extra costs. Please contact sales@brainydata.com for further details.

5. Risk and QA testing

Due to complexity, any changes we make carry a certain amount of risk, i.e. there may be undesirable side-effects. In the table in section 2, we provided a mere guide to the expected level of risk that each mode of development caries. I can hope to do little more here except to add the following.

According to the table, patch releases (D) start with the least risk. However, as D releases accumulate, the risk will widen although it should remain localised to the functions for which the changes were intended. Nevertheless, as patch releases accumulate towards the level of maintenance (C) and minor (B) releases, so does the potential risk of undesirable side-effects. **Localised internal testing is essential prior to releasing your software.**

Major (A) releases carry the greatest risk and should be thoroughly tested during the various alpha and beta stages. We do not recommend releasing your software without comprehensive internal testing.

Regarding our own testing, this will be limited to the functionalities exposed in our example libraries that typically accompany our software. As you are likely to use our software in ways we may not have anticipated, **it is vital that you develop your own testing procedures so that you do not purely rely on our QA procedures.**

6. Conclusion

This document has outlined the natural progression of our development life cycle, starting with patch releases that amalgamate into maintenance and minor feature releases, and which eventually conclude in a major release. Because we make many new features available in minor releases, major releases may not carry as many new enhancements as is typically seen in other software. Our main aim is to restrict risky changes to major releases so our meaning of a major release is more about high risk changes and less about new features. This development life cycle is reflected in the way we price and license our software, i.e. why we charge for maintenance and why, in the absence of a maintenance subscription, our software upgrade prices also apply when upgrading between minor versions.

Of course the modes of development as described here are our intention in best case scenarios and it may be that in some circumstances we deviate from these modes.

Finally, I would like to say that I believe our approach to gradually improving our software with new features being introduced slowly but steadily in minor releases has the benefit of safety and the benefit of not being counter balanced by the cheer volume of changes in typical major versions, which in the normative development life cycle is common place, and which is often driven by time constraints due to revenues being restricted to major release cycles. This brings to mind the old Aesop's Fable of “The Tortoise and the Hare”.

Document History

02 February 2020: first publication