MicroCODE Consulting Services
Application Development Notes

{2012-05} MicroCODE Software Support Process v008.docx

Subject: Application Development and Maintenance Process
Audience: MicroCODE Customers
Purpose: Explanation of recommended App support process
Author: Tim McGuire
Last Updated: Thursday, February 03, 2022

Background
Developing complex manufacturing software applications, and supporting time critical manufacturing, requires a proactive and highly responsive software support process.

Requirements
MicroCODE sees application development as a set of concentric 'Circles'*. An application 'Circle' consists of the following elements:

- A code base used to build all parts of a solution
- A documentation set describing the use of the code base
- A team, or allocated time for personnel within a team, to support the Circle activities

The following application Circles must be supported concurrently, at a minimum:

Internal Development Circle (ALPHA Code)
This Circle represents the '**next major release**' currently being development. The Business has presented requirements, they have been defined, documented and approved, and the application team is implementing, documenting and testing a new version of the application which fulfills these requirements. This is 'long term' activity.

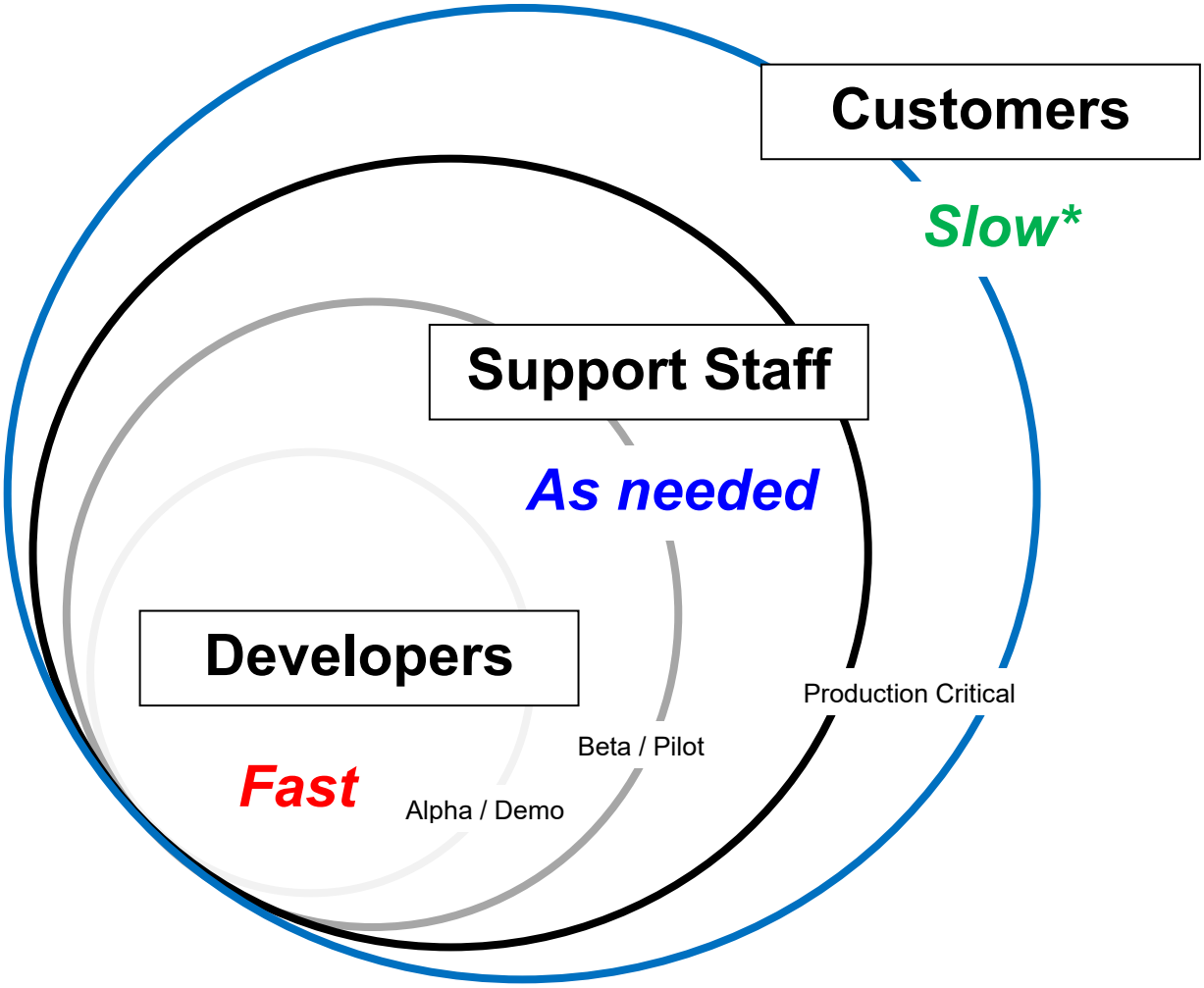
Plant Pilot Circle (BETA Code)
This Circle represents the '**current major release**' being tested and supported in selected customer sites. There could be multiple sites using the beta release in Production. There should only be one release in Beta Circle at any one time. A minimum of three (3) sites, varying in environment to stress the solution, must be completed to exit the Pilot phase.

Plant Production Sites (PRODUCTION Code)
This Circle represents the '**previous major release(s)**' which are running actually Production sites. Based on plant upgrade schedules and downtime availability there will be multiple versions in the field. All of these versions must be supported concurrently.

* Note: In 2017 I discovered that Microsoft refers to these 'Circles' as 'Rings', see…
https://blogs.office.com/2015/08/12/managing-office-365-updates/

Design

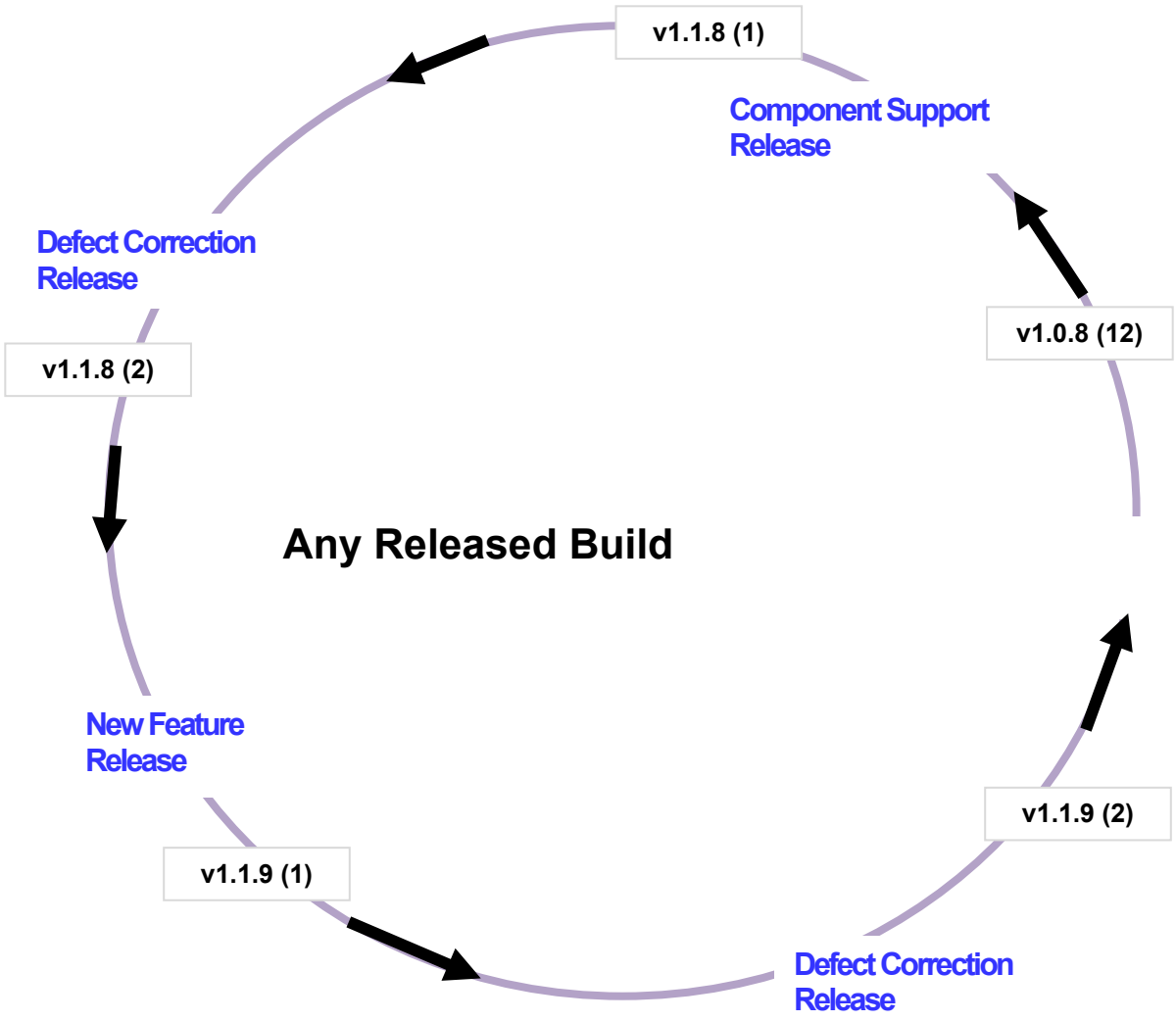
Application development must be viewed as parallel activities in order to respond to customers in a timely fashion. At the outside there are the **Customers** who are actively using the application, possibly multiple versions of it. On the inside are the **Developers** working to implement new features and functionality. And, between the two are **Support Staff** to control access to the expensive resources, prioritize defect resolutions, and provide the Customers with immediate work-around or training to get past non-application issues.



* **Slow**, except for critical Defect Resolution which must be Fast, and override other development priorities.

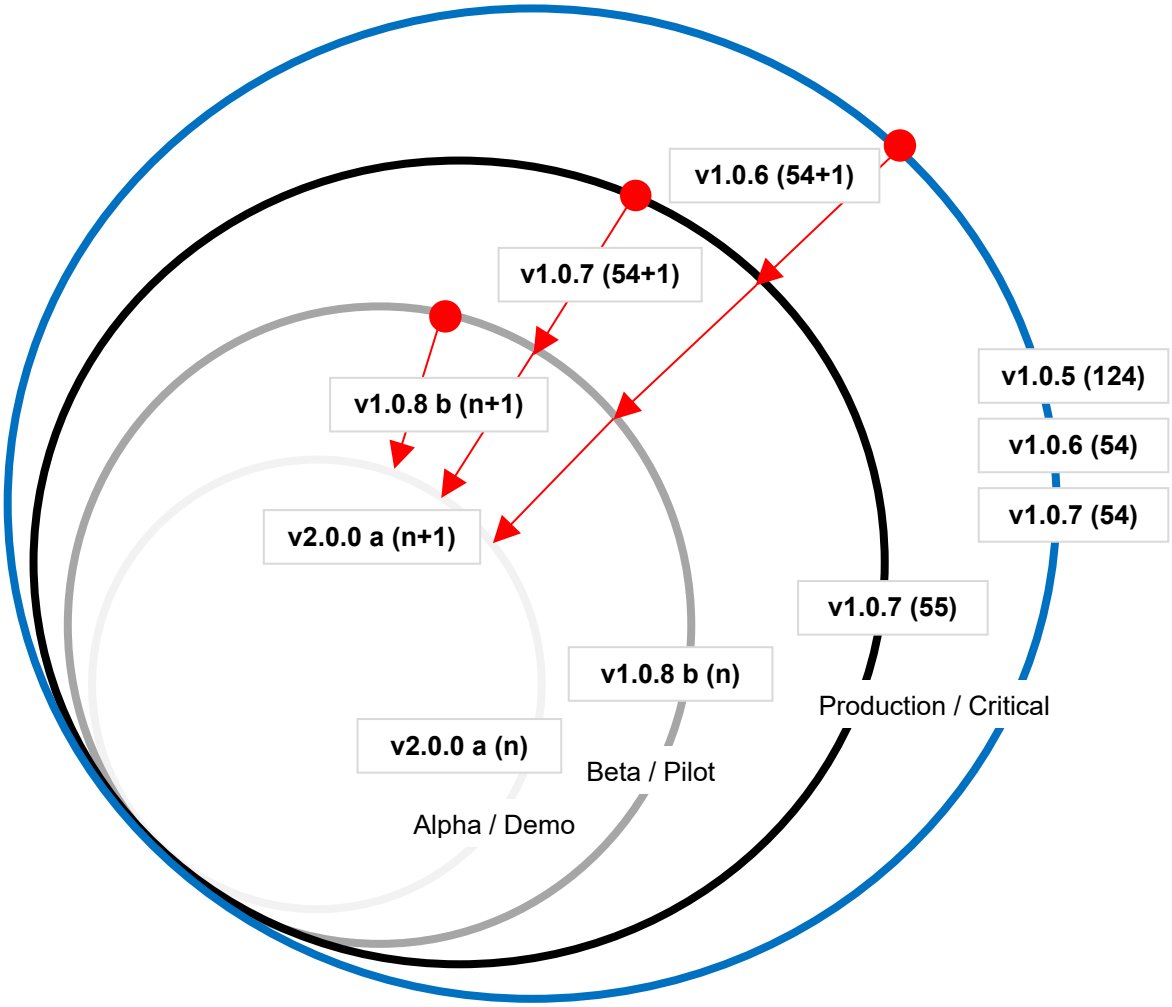
Activities

Why represent each version as a ‘circle’?
Because each released version will go through a cycle of multiple builds over its lifetime as a result of a healthy customer support process. This will continue until a Release goes to ‘End of Life’ and support for that particular Release is terminated, with fair warning given to all customers of course.



Version Control
Version control must support the parallel activities described on the previous pages. Versions in Production must be able to be patched for critical defects or critical enhancements to meet the needs of the customers. *Current SEP Actions CE App version number are shown as an example.*

The **red arrows** illustrate the need to fix defects **first in the version where they arise**…creating a new Build Number (nnn), shipping the resolution into Production, and then, **carrying that defect resolution down into all newer code Circles**, possibly with different implementations, or simple checking that the defect does not exist in the newer solutions.



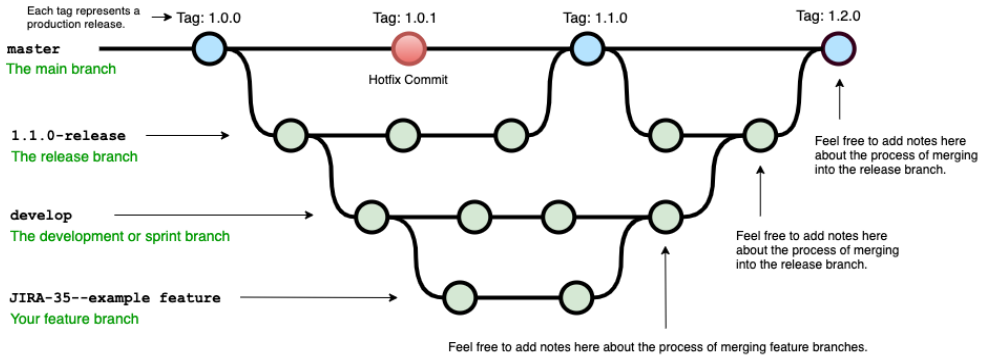
GitHub

This version control model can be easily supported by Git and GitHub. The beauty of Git is its flexibility, the danger of Git is its flexibility. Because it is so flexible it is very easy to branch yourself into oblivion without a well-defined process. **Git Flow** has a defined process very similar to MicroCODE's.

Example Git Branching Diagrams

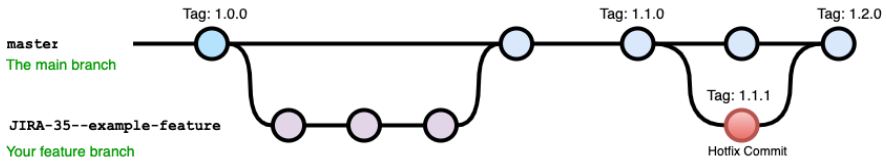
Example diagram for a workflow similar to "Git-flow" :

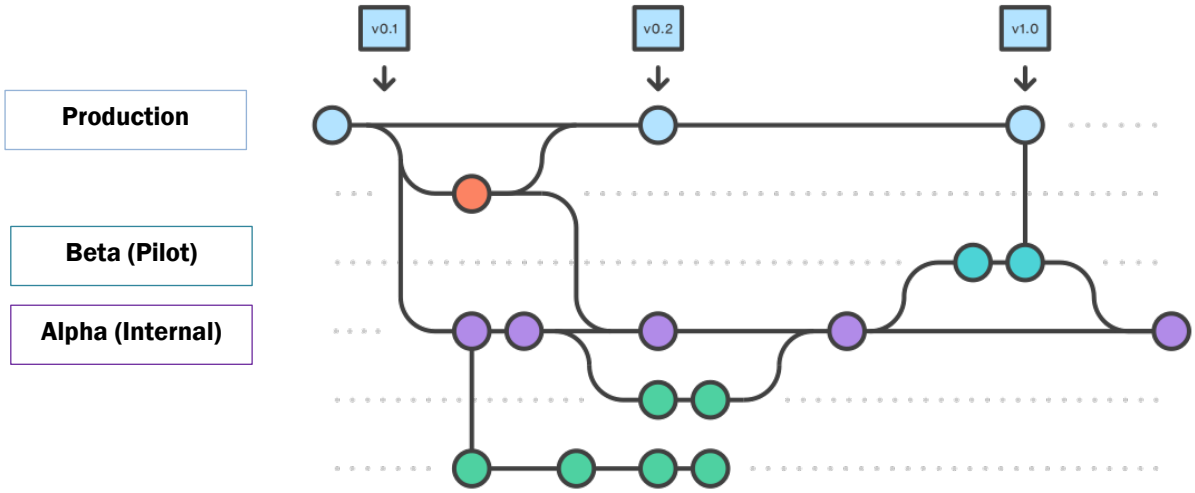
See: https://nvie.com/posts/a-successful-git-branching-model/



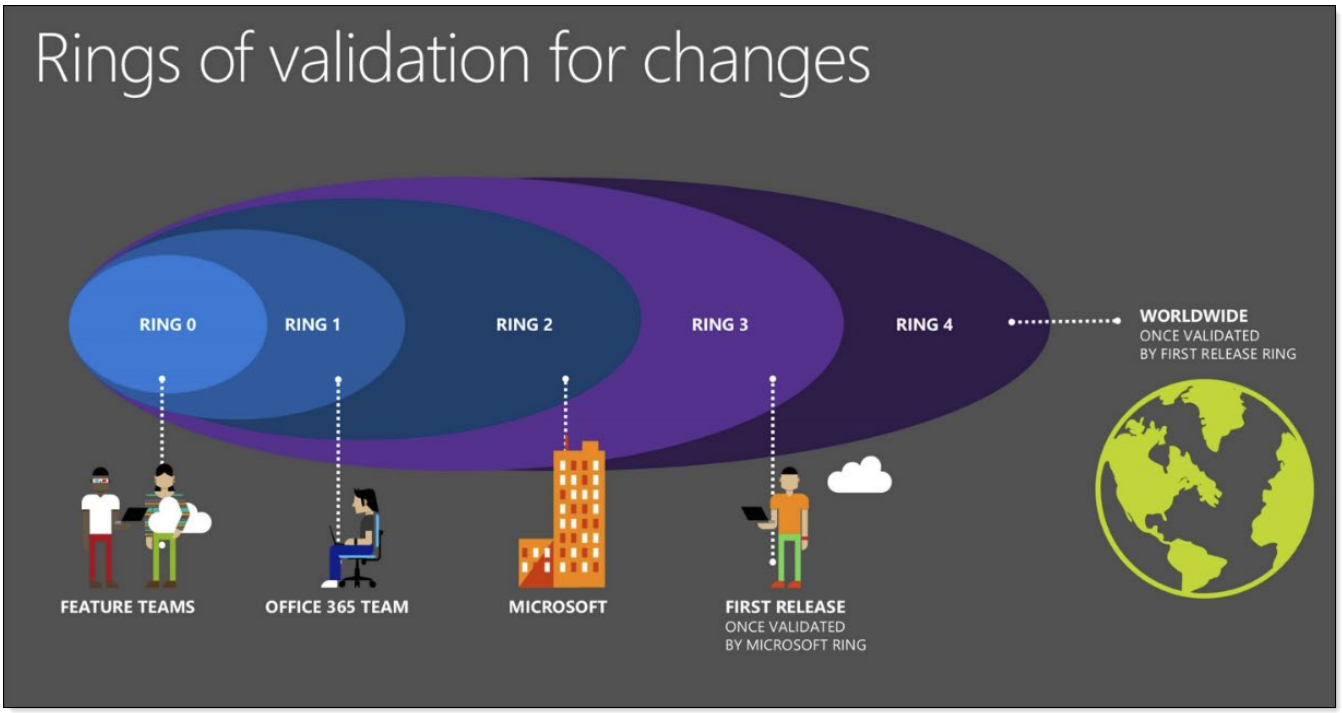
Example diagram for a workflow with a simpler branching model:

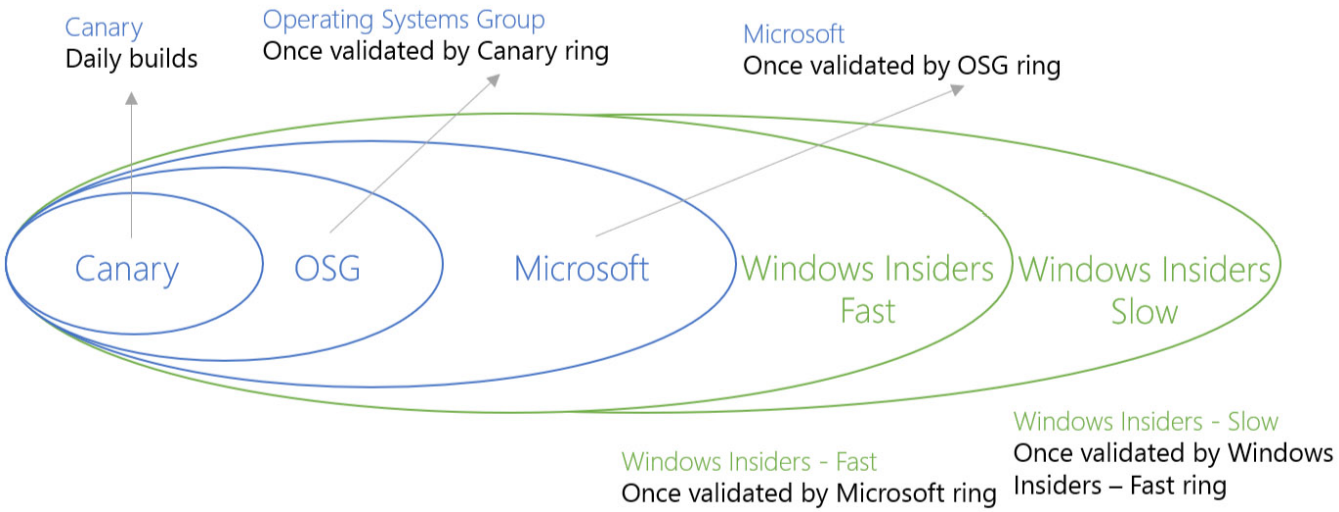
See: https://gist.github.com/jbenet/ee6c9ac48068889b0912 or https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow





I've been drawing this picture since the 1980's, here is Microsoft's version in 2017:





Application Version Numbers

The application software version numbering is defined as follows…

vM.m.R Circle (B)

M = Major software version; represents application architecture, underlying technology, etc.
Incrementing this number is associated with a '**Major Release**'.

m = Minor software version; represents new components within the application.
Incrementing this number is associated with a '**Component Support Release**'.

R = Incremental Release Number; represents collections of new features within the application.
Incrementing this number is associated with a '**New Feature Release**'.

Circle = Development Circle as in ALPHA/DEMO, BETA/PILOT, or PRODUCTION. In the case of PRODUCTION, the Circle label is removed.

Changing this label is associated with a '**Code Circle Promotion**', i.e.: Internal Build Promotion. This is a rebuild/relabeling only no code is changed. E.g.: v2.0.0 Beta (017) → v2.0.0 (001).

B = Build Number. This is the internal build number of the application from within the development group; any time code is changed **and released into the Support Staff** this number must be incremented, no matter how small the change.

Incrementing this number is associated with a '**Defect Correction Build**' or '**Critical Enhancement Build**'.

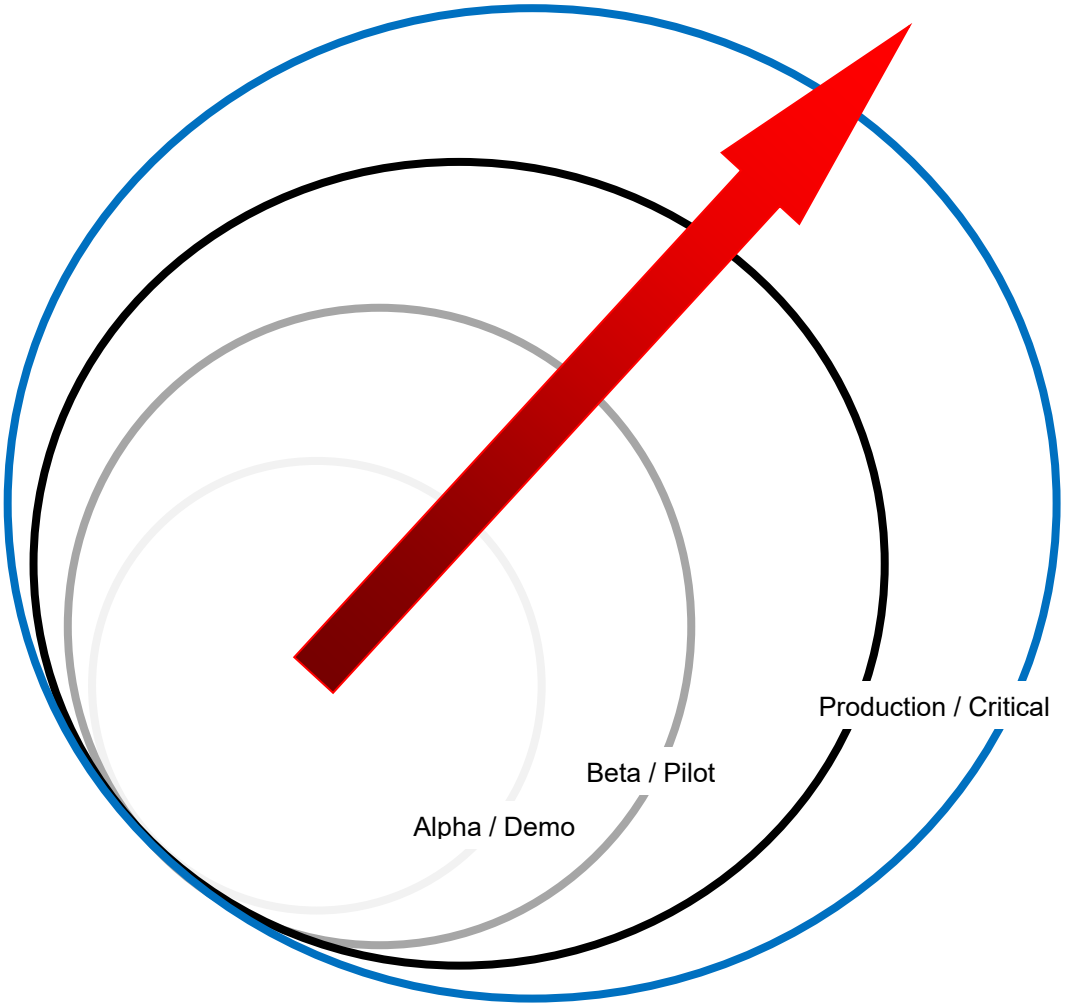
Examples:

App v1.0.5 (124) – Production Version found in Customer Sites.
App v1.0.5 (125) – A defect correction or critical enhancement to (124).

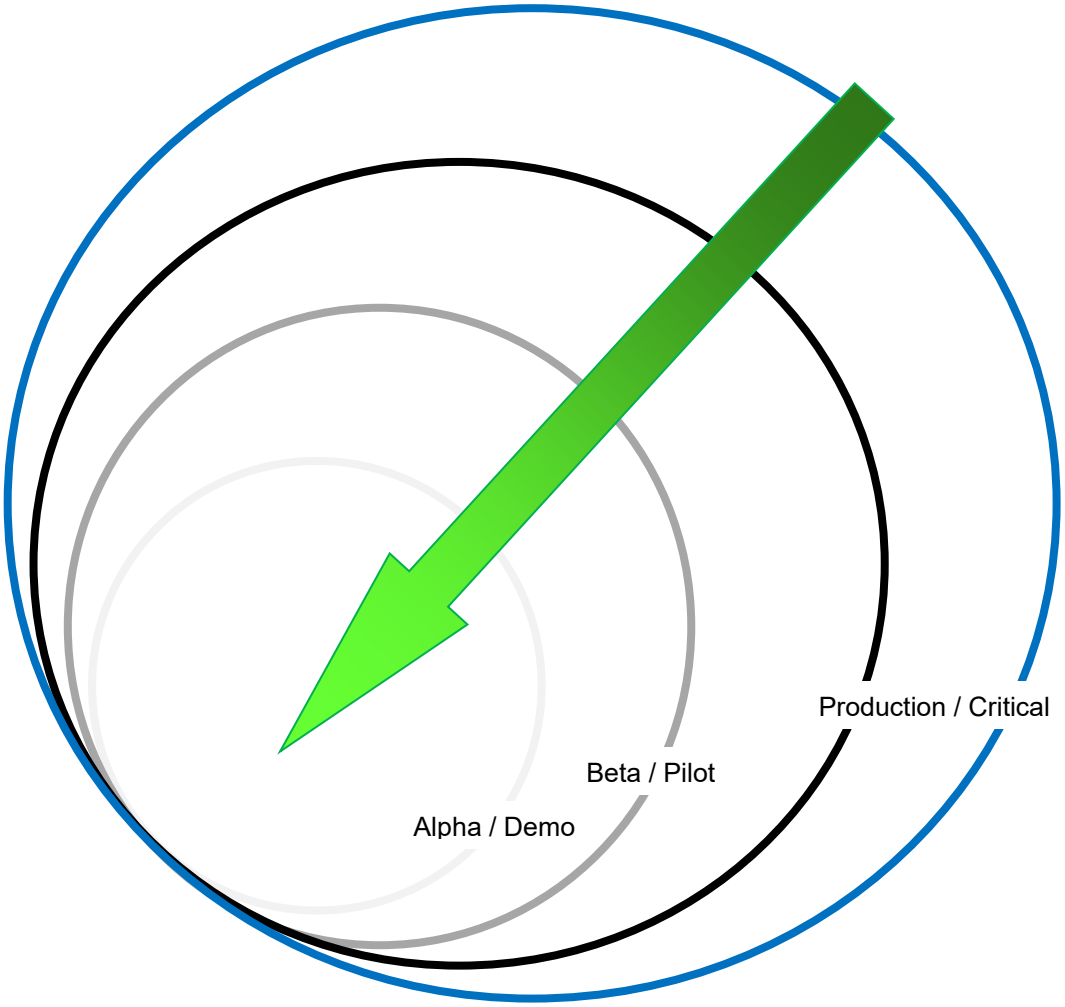
App v1.0.6 Beta (007) – Beta Version found only in Pilot Sites and Support Staff Lab.

App v2.0.0 Alpha (1001) – Alpha Version found only on Development Machines or in the Support Staff Lab.

Risk, Expense, and Urgency
The further out you go in the Circles greater the risk and expense associated with a defect or defect correction.



Flexibility, Time, and Options
The further into the Circles you go the greater the time and flexibility associated with any software change.



Defect Severity Definition

In order to prioritize defect resolution a severity must be associated with every System Issue logged by the Support Staff, and the severity must be agreed upon with the Customer / Production Site. By definition every call into the Support Staff must be logged as a Support Issue (e.g.: A 'Ticket Process').

These are MicroCODE's definitions, impact statements, and recommended responses.

Severity	Description	Impact	Response
1	System crash, complete loss of a major system component	Lost Units or loss of vehicle integrity	Immediate defect resolution and Build Release (**Hours/Days**)
2	Function fails, no work-around is possible, or missed defect or buildable job that was not able to be error proofed	Lost Units or loss of vehicle integrity	Immediate defect resolution and Build Release (**Hours/Days**)
3	Function fails, work-around is possible, or opening erroneous defects on vehicles	Loss of vehicle integrity	Defect resolution and Build Release (**Days/Weeks**)
4	Function fails, no impact to Production, no work-around is necessary, or defect is caused by reconfiguring a Station during Production	Local Support required	Defect resolution in next Minor Release (**Weeks/Months**)
5	Display, Report or Tool problem - no impact on system operation	Possible time lost in the future due to poor information	Defect resolution in next Minor Release (**Weeks/Months**)
6	Annoyances	Loss of resource time, delays	**Business Case** required for resolution

Once severities and responses are defined then support processes can be defined and be held up to those standards for customer support reviews.

Examples of these processes are described follow the next pages…

Defect Resolution Process: 'Defect Correction Release'

Defect resolution releases are defined as those required to correct Severity 1, 2, or 3 defects *or* the Production Site can present a Business Case showing a significant loss of time, money or resources due to a Severity 4, 5, or 6 defect.

Situation: Version **v1.1.9 (22)** of an application is in Production. A defect arises that is deemed Severity **2**.

Step 1: The Support Staff stage a copy of the Production Site in a lab—or visit the customer site—to reproduce the problem. Staging a copy of the Production Site requires all affected components be set to the same application versions as the Production Site, the site's data be loaded (possibly), and then a simulation run to recreate their issue.

Step 2: Once the issue is re-created the Developers are called in to recreate the issues in debug mode to isolate the code issue(s), engineer a fix, test the fix, document the resolution and create a new Build, in this example **v1.1.9 (23)**.

Step 3: This Build would be turned over to the Support Staff for regression testing of the affected components *which would have been identified in the Developer's defect resolution documentation.*

If regression testing of the affected functionality fails the Support/Developer interaction repeats until the Production Site defect and all affected components pass regression test, each cycle producing a new Build increments the version's Build number. So, the version shipped to the Production Site—and posted for use in other sites—may be **v1.1.9 (23)** if several Developer/Support Staff cycles were required.

Step 4: At that point the updated application components are delivered to the Production Site with required documentation included: **Release Notes**, **Installation/Upgrade Procedure**, etc.

Final Step: The Defect Resolution is carried into all inner Code Circles, i.e.: into all other Builds in Production or in development stages in the Lab.

Minor Release: 'Component Support Release' Process

A minor release would by definition be the addition a new component to an existing version of the application but no changes to the overall application structure. i.e.: Support for a new I/O Device.

Step 1: An existing version of the application, already in use at the Production Site requesting support for the new component is chosen with agreed from the site. E.g.: **v1.0.5 (124)**.

Step 2: Support for the new component is added, tested, documented and regression testing of any affected components is performed. A **Beta** Build is created for the Pilot site. E.g.: v1.**1**.5 Beta (001).

Step 3: The **Beta** Release is piloted at the requesting site. Defects are handled via the standard **Defect Resolution Process**. After a defined time period with no Severity 3 or higher (2, 1) defects the release promoted to 'Production' status and published for any sites to acquire and deploy. E.g.: v1.1.5 Beta (027) → v1.**1**.5 (**001**).

Final Step: The Component Support is carried into all inner Code Circles.

Incremental Release: 'New Feature Release' Process

A new feature release would by definition be the addition a new feature into an existing version of the application but no changes to the overall application structure. i.e.: An Operator Screen for the Torque Tool Action.

Step 1: An existing version of the application, already in use at the Production Site requesting support for the new component is chosen with agreed from the site. E.g.: v1.1.5 (014).

Step 2: Support for the new component is added, tested, documented and regression testing of any affected components is performed. A **Beta** Build is created for the Pilot site. E.g.: v1.1.**6** Beta (001).

Step 3: The **Beta** Release is piloted at the requesting site. Defects are handled via the standard **Defect Resolution Process**. After a defined time period with no Severity 3 or higher (2, 1) defects the release promoted to 'Production' status and published for any sites to acquire and deploy. E.g.: v1.1.6 Beta (007) → v1.1.**6** (**001**).

Final Step: The New Feature is carried into all inner Code Circles.

Major Release: 'Major Release' Process

A major release implies changes to the overall application structure, like movement to a different Hardware or Software platform. E.g.: Support for Windows 7 Embedded and .NET 4.0

Step 1: The newest existing version of the application, already in use at the Production Site(s) is selected for the development effort. E.g.: v1.0.8 (61).

Step 2: The application is migrated to the new platform, tested, documented and regression testing of **all functionality** is performed. An **Alpha** Build is created for the Support Staff site. E.g.: v2.0.0 **Alpha** (001).

Step 3: The **Alpha** Release is passed through System Acceptance Testing (SAT) until it passes all required Software Test Cases (STC). Defects are handled via the standard **Defect Resolution Process**. After all STCs are passed with no Severity 3 or higher (2, 1) defects the release promoted to 'Beta' status and a **Beta** Build is created for the Pilot site. E.g.: v2.0.0 **Beta** (001).

Step 4: The **Beta** Release is piloted at the volunteer site. Defects are handled via the standard **Defect Resolution Process**. After a defined time period with no Severity 3 or higher (2, 1) defects the release promoted to 'Production' status and published for any sites to acquire and deploy. E.g.: v2.0.0 (**032**).

Final Step: There are no inner Code Circles during a 'Major Release' cycle.

After 30 years of software development and customer support we believe this is the only way to properly support Production Critical Software.