Maximize Outcomes by Establishing a Release Driven Workflow, Upfront

It's two AM on an early Wednesday morning, the entire development team is rushing to figure out what it's going to take to roll out a release. For every problem that's solved, two new issues arise. Half an hour goes by in what seems like five minutes. Maybe the team gets lucky in time for the five AM cut off. Maybe they roll back and try again another day. Maybe things end up somewhere in-between and nobody knows it!

No organization that develops software wants to be in this position, but for many, it's a reality.

How did we get to this point?

- Schedules are tight, the entire 40-hour work week is dedicated to cranking out code.
- Pre-canned demonstrations of prototypes are well received.
- Demonstrations are hosted in development environments where conditions are controlled to impress.
- Developers carefully navigate demonstrations to avoid "incomplete" features.
- An overly optimistic atmosphere ensues and, inevitably, someone wants to push the timeline up. Things are going so well, why not?

Perhaps, some of this is familiar?

Ultimately, the reason why the release is a nightmare: there's no focus on what it will take until it's time to release! For whatever reason, the work to prepare and test isn't done in advance.

Why don't we prepare for releases?

The focus of developing software has become obsessed with what goes in to the process: inputs and tasks. Even with the best of intentions, discussions quickly digress to mock-ups, reports, interfaces, frameworks, databases, the cloud, notifications, integrations, etc. This focus on what the software will do, instead of what we're trying to accomplish, derails due diligence in determining the next appropriate actions. Software is never an end in itself, instead, it's a means to create value for customers.

Because the focus shifts so quickly to what the software does, the next "logical" step is to jump right into developing code. We're no longer focused on outcomes. We're focused on taking the next step down a not-so-well defined path. Every subsequent part of the path ends up neglected until it's too late. We're too busy writing code to consider what it will take to release it.

However, if we start with desired outcomes and work backward to determine what's needed to deliver software that achieves these outcomes, then, before we ever develop a single line of code, we'll realize that it makes sense to put the basics of the infrastructure to release the software in place first. We'll create the foundation of a framework that we can evolve. We'll establish a mindset that focuses on what it takes to accomplish outcomes, not on what the next line of code contains. We'll focus on the target and be much more likely to hit it.

What's the objective?

How do we shift the focus? First, we have to ensure a worthwhile outcome. Any necessary software must be framed in the context of an outcome. The software will exist as a means to achieve the outcome. The outcome should provide clearly defined value to the organization. If the outcome is unknown, defining it should be the starting point.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

Outcome

An outcome is not a set of requirements, it's an objective for what the software should accomplish. It's the target we're aiming at, the focal point with which we continually align our efforts. Without a clearly defined outcome, we'll find ourselves wandering in a desert of code.

An outcome might be: "We'd like to take on an additional 100 customers over the course of the next year. To accomplish this, we need a system that does X or it just won't be possible." See how we're already working backward based on what it will take to serve another 100 customers!

Recognizing the need to release frequently

Once we've identified the role software will play, the next thing we need is working software in the hands of users. This doesn't happen overnight. Nonetheless, it's the precursor to our desired outcome.

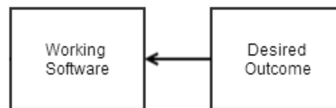

Figure 1 - The desired outcome constrains the software we create.

When we start with the code, it's no wonder many of us make the mistake of attempting to develop the entire system and release it all at once. However, our experience dictates that there will be many subsequent updates, often rushed, to get the software functioning. Logically, many of us have adopted the practice of iterative releases. Either way, we're going to release many versions of the software before we get exactly what we need.

The reason for this is simple, releasing is one of the best ways to get feedback. Whether we intentionally seek this feedback by releasing frequently, or we wait until we think we're mostly done. We're going to get vital feedback about how the software does, or does not, attain the desired outcome. Gathered frequently and used effectively, this feedback helps minimize the effort and maximize the likelihood of success.

What we really need is the capability to release as often as we want, to get the feedback necessary, to achieve our desired outcome. In order to release as often as we want, we need an efficient, repeatable process. Repeatable processes demand automation. Leaving this up to sparsely documented, manual repetition is a recipe for failure. Add in the need for efficiency, and automation is the only viable way to release as often as we want.

By implementing this in advance of writing the first line of code:

- We can evolve the process to support the code we create.
- We can use and test the process to prepare for demonstrations.
- When the time comes to make our first release, there shouldn't be any surprises.
- We'll reap more value from the investment. The sooner we create it, the more we can use it.
- Challenges will seem surmountable.
- We're much more likely to establish a habit of maintaining an efficient, repeatable process.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

In the following sections, we'll look at what common aspects need to be considered to develop the capability to release as often as we want. Many of us put off releases under the assumption that it won't be that complicated. That's often true at the start of a project. But, the complexity exponentially accumulates as the project marches forward. And, there are many aspects that we don't really associate with releasing software until we take a more proactive approach. These also become more complicated with time. At the start of a project, we have the unique opportunity to make the minimal investment to get on the right path. Every day thereafter, the right path grows further and further out of reach.

Identify the Production environment

Released software has to have somewhere to run! First, we need to identify the environment in which we'll host our software. I'm going to focus on a web based application. Minimally, we'll need a server to host the application and, likely, a database system. It's typical to refer to this as the production environment. In another situation, say, a desktop or mobile application, we'd want to identify the most common environment(s) in which a user would run the application.

What it takes to get it running quickly in Production.

Based on the desired outcome, we should have an idea of what usage might be like and be able to consider the resources we'll need. Even if we don't put all of them in place initially, we can start the conversation. Once we've identified an environment to host our application, we'll need a mechanism to get it set up and running.

The following questions come to mind when we start to think about what it will take to get the application running quickly, in our production environment:

What configuration will the environment require to support the application?

What software needs to be installed? What settings need to be altered? Will the configuration change frequently? Perhaps, it makes sense to incorporate a Configuration Management tool to automatically configure the environment. Perhaps it doesn't. The point of working backward is to have these discussions before it's too late.

If you're leaning away from a Configuration Management tool, consider this. If the configuration is truly that simple, introducing a Configuration Management tool should be simple. Just having this tool in place will lower the barrier to leveraging the tool when we really need it. Either way we're going to have to document what we've done. Why not document what we've configured in a set of scripts that actually set up the environment?

How will the software quickly be installed into the environment?

Assuming the environment is configured. What will be necessary to install the software? In many cases it's simply a means of copying artifacts. If that's the case, we may set out to create a script to copy the appropriate artifacts. If we'll need an installation tool, we'll set out to create that tool. Perhaps the application can be delivered as a part of the Configuration Management tool?

How will we know the software is working in the environment?

This simple question, often overlooked, can be the most valuable to ask upfront. There are many things we can do to quickly add confidence when releasing software. Before we've composed any code, we can put a framework in place to support automated end to end testing. End to end tests help us verify that an environment is set up and running as expected.

In the first few weeks we can use end to end testing to create smoke tests. Tests that literally look for smoke when we turn things on. Smoke tests quickly run a few common scenarios to give us a decent level of confidence that our application is operational. They need not be comprehensive to provide significant value.

To put the framework in place, it's important to decide on a few tools to use upfront. Don't fret, we can always change the tools. But by putting this framework in place in advance, we eliminate a rather large barrier to developing the automated end to end tests. Without a framework, automated end to end testing is always doomed to failure. With a framework in place, we shift the mentality from "That's going to be a lot of work to set up" to "Now we don't have to check X, Y, and Z every time we release!"

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

Without automated tests every release will require someone, at a minimum, kicking the tires. As the need for more complicated verification emerges, manual validation will grind releasing to a halt, in short order.

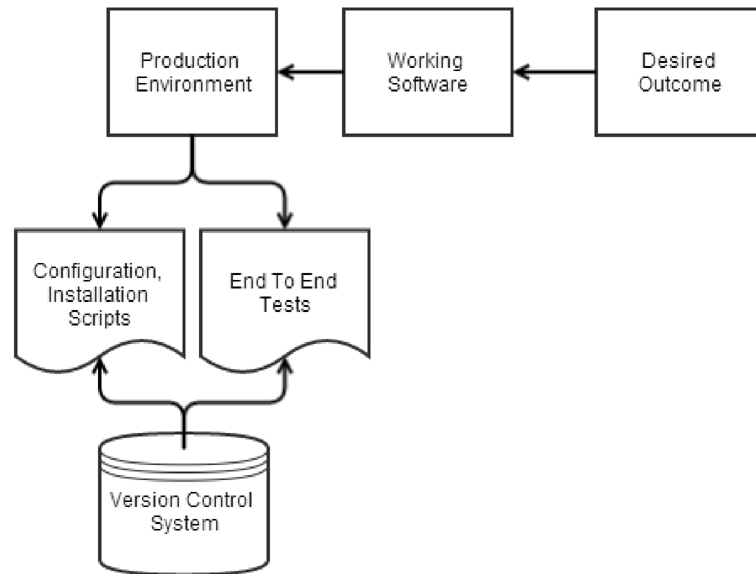

Figure 2 - Working software depends on a carefully configured and tested Production environment.

Version control systems

At this point, to release to our production environment we've identified these three pieces:

- A skeleton of the scripts to quickly **configure** the environment.
- A skeleton of an installation script to **install** the application.
- A skeleton of an **end to end testing** framework that incorporates the tools and conventions we'd like to start out with.

Since we have actual scripts at this point, we'll want to keep track of them. To do this we're ready to introduce a version control system. We need to store all of these scripts in version control, they're just as important as the code we'll eventually create! Version control provides us with a clear history of every change to the scripts, why the changes were made, who made the changes, and when the changes happened.

A testing environment

After we have a handle on automating production releases, we're going to need an environment to do the following:

- Test changes to the application.
- Demonstrate changes.
- Support manual and exploratory testing of changes.
- Run automated tests to **practice** for production releases.
- Prove a release is ready for production.

We could do this in development environments. And for some projects that would work. But the reality is, if this environment doesn't closely match production, there are going to be a host of problems that arise only when we actually release. We wouldn't practice the flute to get ready for a piano concerto. All we need is an environment that is a close mirror of production.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

The best part, we can reuse the process we created to release to production. What better way to prove it works! We simply need to add a parameter to pass the location of a test environment to our configuration, installation and end to end tests. As we encounter impediments in releasing to our test environment, we'll remedy them in the same scripts we'll use when we move to production.

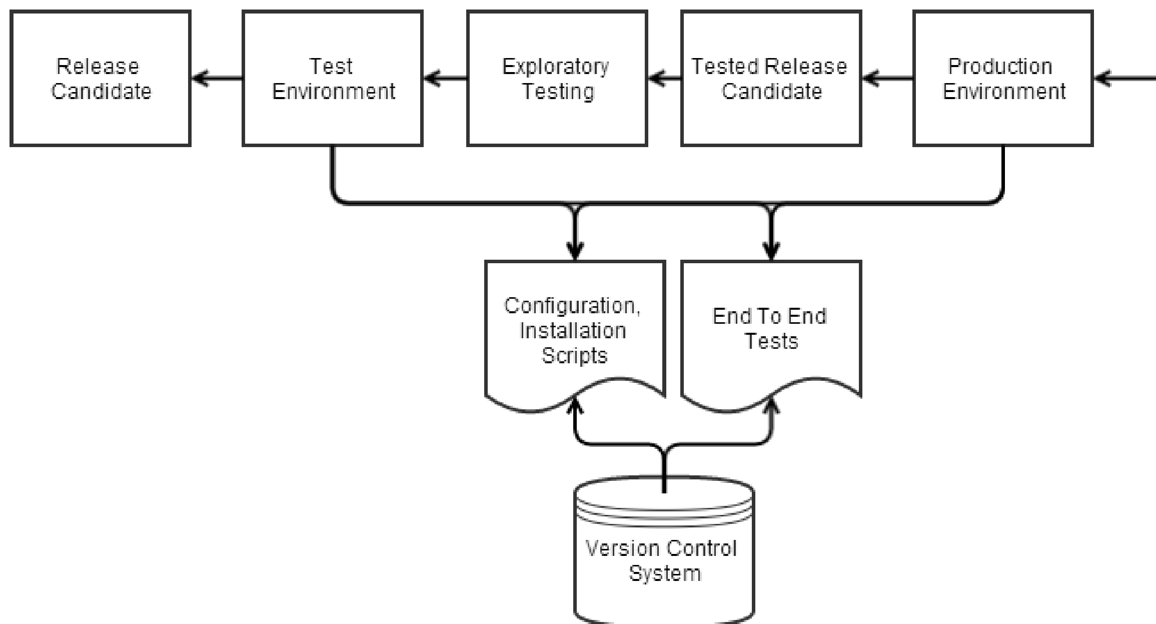

Figure 3 - Production releases rely on successfully tested release candidates. A Test environment that closely mirrors production is necessary to avoid surprises.

With a test server in place, we can compose a simple pipeline to push release candidates to the testing environment, run the end to end tests and gather the results. Once a test release passes automatic verification, exploratory testing can be done. Once a release candidate is proven successful, we can promote to production at the push of a button. Over time, we can move repetitive and burdensome delays in manual testing into automated end to end tests.

A tool to orchestrate and provide visibility

Our workflow is starting to garner quite a few steps. To maintain our ability to release quickly, we need a tool to compose and automatically run the steps. At the push of a button we need to be able to trigger the appropriate configuration, installation and verification of our test, and production environments!

We also need auditing of what's released and where, we need visualization of releases in progress, we need notifications of failures, we may need to limit who is authorized to release changes, and eventually we'll want some steps of the process fully automatic instead of requiring manual intervention to push a button. For example, we may want every release candidate automatically pushed to our test environment. And, we'll want the entire team to have access to this information.

Implementing this for each project would be a lot of overhead. Fortunately, we don't need to reinvent the wheel, this is where a good Continuous Integration/Delivery tool fits in. Simply making this tool available upfront provides a skeleton to automate the entire release process.

Packaging the application

We now have a pipeline of scripts to install and run end to end tests on our application. Now is probably the time to discuss what's necessary to quickly package the application as a release candidate to install into a particular environment.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

This will require compilation and transformations as necessary to produce the application we'll be installing. The end result is usually a set of files or some sort of installer.

We can use scripts to quickly package the application. We can plug these packaging scripts into our release pipeline. Since we've already established a version control system, our Continuous Integration system can trigger this packaging every time someone makes a change.

We can start with a simple "Hello World" application, make sure it packages successfully and then run our configuration, installation and end to end testing scripts to ensure it's working in our test and production environments.

Verifying the package

At this point, we could start creating the application. But, we're going to quickly find our release process hampered by issues we could have caught if we did some validation of our application when we packaged it, before it's deployed to the testing environment.

End to end tests help us validate if the end result is operational in a given environment. Another set of automated tests, often referred to as component or unit tests, can help us validate the components of the application before we install them into a testing environment. These tests often catch the majority of issues that would otherwise require complicated end to end and / or exploratory testing.

At this point we only need to agree on a testing framework and a set of conventions to decide how and when to test. We can always change this later. Then, we need to create the scripts to trigger these tests immediately after packaging.

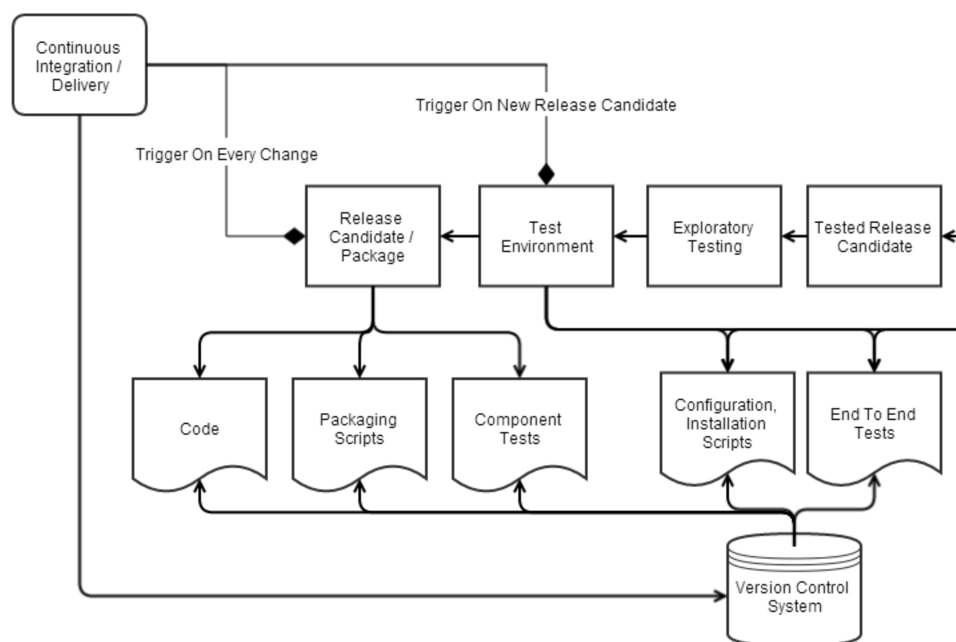

Figure 4 - A Continuous Integration/Delivery tool can create a new release candidate when any artifact in Version Control is altered!

Failing fast

Much of the release process is designed to find out about problems sooner rather than later, so we can do something about them. This is known as failing fast. We now have a framework to help us find out about problems quickly.

I highly recommend including a Continuous Integration/Delivery tool to notify everyone when something goes wrong. Every time someone makes a change to the application, it should package and validate the application. And, eventually it should automatically deliver validated packages to a testing environment to get the fastest feedback. If anything goes wrong, it should tell someone about it.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

If packaging release candidates is left to manual intervention: it'll easily be forgotten, we'll fail to receive timely feedback about issues, and our ability to release will suffer.

Time to code?

By working backward from the desired outcome, and establishing a basic release pipeline before we write the first line of code, we now have a process to help us confidently create and evolve our application. We'll know about problems sooner rather than later. We're much less likely to be surprised at two AM in the morning, pulling our hair out!

We now have the capability to add code, sometimes even a single line of code, and quickly push that code through to a production environment with a high level of confidence that everything will be OK.

Incrementally improve

This scenario is just a sample of what we may want to establish. For example, we'll likely want to incorporate a database system pretty quickly. When this happens, we may ask the following questions (working backward):

- What database tools will we need?
- How will we install and configure the database in production and testing environments?
- What scripts should we create to quickly release changes to the database?
- What will it take to validate those changes?
- How can we store these changes and scripts in our version control system?

Because we have a release pipeline established, we can easily incorporate the components necessary to support a quickly-evolving a database. Something that is traditionally very cumbersome and error prone!

The upfront effort is negligible

Some of this may seem complicated. But, when this is established up front, it's much less daunting than it seems. All we're talking about is doing a little bit of development to support our development. Isn't that what we're experts at doing for other people?

Not to mention, we're doing all of these steps manually already! Wouldn't we rather have these tedious manual tasks automated? We already know what it takes to do all of this; we just have to have the discipline to prioritize automating things upfront. The upfront effort required often pays dividends in releasing even the first iteration of an application.

A mentality

By investing in the process upfront, we'll be much more likely to establish a mentality that prioritizes releasing and thus, quickly reap the value which that software provides.

If we put off releasing until the first release, we're much more likely to suffer through pulling our hair out to hack a release together. Because we waited until the last minute, we won't have the time necessary to pay down the debt we've already accumulated in building the first release. Most projects that don't prioritize quickly releasing, suffer the following:

- Surprises when configuring production environments.
- Surprises when installing software into production environments.
- Surprises long after the software is installed in production due to a lack of effective testing.
- Surprises over time as features are lost and/or broken.
- In time, fixing surprises begins to dominate resources. Changing the application becomes difficult and time consuming.
- A vicious cycle of accumulating debt that keeps us from ever investing in an effective release process.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com

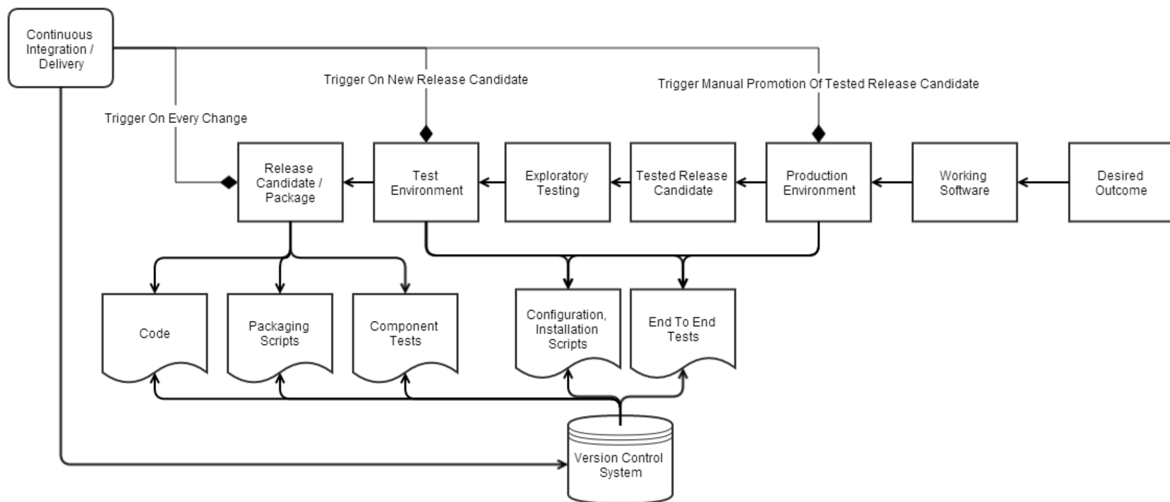

Figure 5 - The entire release workflow.

Good intentions or upfront action?

Imagine what impact this simple framework would have on any of your existing applications.

- What if you could easily add component level tests?
- What if you could easily add end to end tests?
- What if you knew, every time the application was released, that all of these tests would be there to support you?
- What if you had a trusted system to quickly find problems?
- What if you could release the application, at any time, to any environment, at the push of a button?
- What if releases only took a few minutes?
- What would this mean to you?
- What would this mean to your organization?
- What value would this provide to your customers?

We all have good intentions of putting these practices in place. But we all know what road is paved with good intentions. Instead, we should demand "**Not this time!"** Whether the project is new or old, we should put down the code, focus on the outcome, and work backward to establish a process to quickly create and deliver that outcome. A process that supports and gives confidence in what we're doing.

Copyright © 2014 Wes McClure www.wesmcclure.com / www.fullcitytechnology.com