

Great Expectations (of a Build Engineer)

Jeff A. Bowles
Piccolo Engineering, Inc
jab@piccoloeng.com

June 2003

Abstract

A handoff is the process of preparing work into a form usable by others. It is an Engineering task with formal underpinnings, much more than a software automation role.

This document describes the role of a build/release engineer when the emphasis is placed on the ENGINEERING: with what groups does he or she interact (and in what ways), what are the requirements of a software handoff, etc. The key is to understand who will receive the work product, and what they need to receive in order to proceed efficiently.

Implementing CM Everywhere, Change, Configuration & Content Management

1 INTRODUCTION

On the path between the software developer and the customer are several stops: the person building and packaging the software, the one testing it, the one who will eventually support it, and the one who will train customers to use it.

At some level, a person, process, group, or role will be responsible for moving it along this path. This is sometimes a managerial position in an *Integration, Development, or Quality Assurance (QA)* group. Often it falls to an engineer who doesn't have training for the position, or worse, who will juggle this responsibility while trying to do other work.

This paper tries to identify some of the groups on this path and their expectations. No list of those expectations would be exhaustive. It is more helpful to poll those groups and then write automated scripts and reports that satisfy their needs.

1.1 Why spend time on this?

Time is money; it is easy to waste someone's time without realizing. Whenever a QA/Testing engineer or Technical Support analyst has to interrupt his/her work to ask for information, someone else has to stop their own work to provide it. If software is redelivered as part of this interruption, additional costs will include reinstallation and retesting of the new software.

1.2 What is the cost of not doing this?

Every Technical Support engineer has seen a graph that shows the cost of fixing a bug: a small amount when found in the Requirements Phase of a project, a larger one when found in the Development Phase, far greater when found by the QA/Testing group, and a huge cost to fix once the product is deployed.

If a group is squandering effort, by redelivering software due to avoidable issues, then those resources are not available to shake down the product. The quality of the final product suffers, and bugs are not identified until more expensive stages of the product's life.

2 DOWNSTREAM GROUPS – NOT ALL THE SAME NEEDS

The closer you are to the customer, the less direct power (and status) you have to help that customer. This results in customer support groups and testing groups that are given insufficient information to ascertain the stability of a product prior to release. The quality of the product suffers.

There are many groups that are "downstream" of a software development group:

1. OTHER SOFTWARE DEVELOPMENT GROUPS. For example, three groups in the same company might develop components for each other to use: Java™ packages, subroutine/method libraries, tools. They might exchange their "products" internally with no packaging or information, issue "internal releases" as needed, or formally release the products internally with no real distinction between that process and an external (to "real customers") release.
2. THE QUALITY ASSURANCE / TESTING GROUP. This group usually has its own machines, a pristine environment, and schedules that are set by external factors.

3. THE DOCUMENTATION GROUP(S). This group sometimes receives a copy of the product, but approaches it as a sophisticated end-user who asks questions such as, "How do I use this?"
4. THE TECHNICAL SUPPORT GROUP. This group is expected to be the expert on all usage questions including, "Is this behavior a bug?"
5. A MAINTENANCE OR PATCHING GROUP. Often called "sustaining engineering", this is a role within one of the above groups or the original development organization. This group will decide to make changes to a released product to distribute to an individual or subset of customers. The engineers in this group often need to rebuild the product exactly as originally built, but with a small patch/delta.
6. PRODUCT MARKETING. This group often receives an internal release early in the process in order to prepare marketing campaigns.

2.1 The Road Map

The standard sequence of software development has this form:

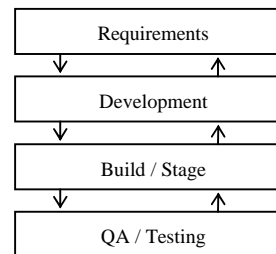


Figure 1: Standard development path.

This is a simplified diagram, to show that certain things are "upstream" of others. Such graphs can be misleading, because they do not reflect that once a group has handed off its work, it is busy working on the next release or next handoff.

There are two costs associated with reworking a handoff: the delivering group has to interrupt the next thing it is working on, and the receiving group has to bear the cost of another handoff (which includes reinitializing build and testing environments).

Redelivery to a "downstream" group is an expensive operation, especially when it interrupts work-in-progress.

This makes it important to avoid unscheduled handoffs when possible.

2.2 The Questions to Answer

Different groups will have different needs, as their questions reflect.

1. The Quality Assurance (QA) group will need to know: what is in the product; how is it different from previous handoffs; how is it installed; how robust is it (i.e., what tests has it passed or failed?); and what are schedules for other handoffs.

Implementing CM Everywhere, Change, Configuration & Content Management

2. The Technical Support group will ask a variation of those questions: how to report a problem, and against which version string; how a patch can be made and delivered; a request for the source, to see how it works; how to enable run-time debugging options.
3. Often, the “maintenance” role falls to Development or Technical Support. In either case, the group doing maintenance has another set of questions added in: how to rebuild the software to recreate the build of the original product; how to modify it slightly for a rebuild/patch; and how to fold those modifications back into official product, when appropriate.

In each case, the point is that these engineers need the answers to start their work. More importantly, the “downstream” engineers in such groups need to be the ones who provide the list of questions to be answered. These are examples to show them, nothing more.

3 REQUIREMENTS

3.1 Common Requirements for a Build Engineer

The build engineer bridges the Development and post-Development groups. Not only must he/she create an integrated product, built from components, but he/she must also contend with many expectations:

1. The build process should be repeatable, and automated so that overnight builds/regressions provide a daily “heart beat” to Development groups telling them the status (“health”) of the current product being developed.
2. The final content, to include this feature but exclude that feature, is not known until late in the Engineering process.
3. What is given to the QA/Testing groups should be packaged as it will be for a final customer. E.g., if it is to be sold on CD-ROM then all QA/Testing handoffs will be on CD-ROM. (*This is often difficult since the contents of the final release are not known yet. Often, a build/release engineer will start with the packaging of the prior release/handoff – the scripts that build that package, that is – and refine/augment using those scripts as a base.*)
4. This process will be repeated enough times that automating it is practically a requirement.

Admittedly, this is sketchy at best.

3.2 The trap: unspoken requirements

There are other requirements that often are not mentioned, but which the build engineer will need to address or find that a rebuild/redelivery is necessary:

1. The product to test should be compiled as a “production build” but that a debugging or instrumented internal build is expected, also.
2. The engineer will be maintaining the tools used to build the product in a software developer’s environment, and the tools used to build it for the production handoff. Often, the same tool/script is used for both needs, to avoid discrepancies that can occur.

3. The engineer should be able to recreate the same build from identical source-code files, in an auditable way. This speaks directly to the integrity of the product.
4. The delivery to QA/Testing needs to be identifiable as such, and that unofficial builds cannot be mistaken for official ones easily. Often, the product carries an embedded version string that distinguishes it from a “developer build.” This avoids committing QA/Testing resources to unofficial releases that are not known to be reproducible.
5. Similar versions for two hardware platforms have similar behavior. This can vary between companies and products, and often the source trees are structured to support software builds for multiple platforms in the same subtree.
6. There should be an owner. (Whose phone rings if the build does not work?)

4 ...BUT WHAT IS A HANDOFF?

Any handoff to another entity (the QA/Testing group, another development group, a customer) must meet this requirement:

There must be an unambiguous agreement that the handoff has happened and that any further updates will comprise a new handoff.

There is no formal definition, that a handoff must be a CD-ROM, handed to the right person, or a certain document, signed appropriately. Each organization must decide what comprises a handoff that satisfies its needs. Note that a “mushy” handoff, in which someone installs software onto a QA/Testing machine and tinkers with it until it works, then declares that the handoff has happened, does not meet this criterion. (If further tinkering is required, will it really be referred to as a new handoff?)

In general, the handoff is a small set of items: *a version string* to uniquely identify the handoff, when describing it in various reports and bug lists; *software* to evaluate/test; *corresponding user documentation*, such as product documentation, installation notes; *internal documentation* that describes the contents of the handoff, such as bug-fix lists, final copies of requirement specifications, and results of “smoke-test” regressions.

Often, these items can be assembled as part of a build and regression script that automates the retrieval, assembling, and delivery of the handoff. An additional benefit of this strategy is that handoffs become easily repeatable.

4.1 When to hand off to another group

There are two types of handoffs: the one that was scheduled in advance, as part of the Engineering schedules to issue a release or stage a product or project; the one that was not scheduled, but is necessary for an emergency update to let a “downstream group” continue work.

The first case is straightforward. The main rule of thumb is,

Pass along what you know, as soon as you know it. Early knowledge about a delay or product issue gives “downstream” groups time to plan for contingencies.

The second case takes more care. Even if the Development group wants to reissue a handoff, the choice to issue it should

Implementing CM Everywhere, Change, Configuration & Content Management

be an Engineering decision driven by the ability of the receiving group to use another handoff. In that situation, a development engineer or manager could convince his/her counterpart in QA/Testing of the gravity of a problem. The result of the conversation would be a request from the QA/Testing group for a new handoff. This puts the **responsibility** of testing and the **authority** to control the testing environment with the same people, instead of dividing it between groups.

Let the downstream groups, such as the QA/Testing group, drive requests for new handoffs.

If the product is in the QA/Testing phase, it is the QA/Testing personnel who control what they're working on. (For example, often the Testing group might respond, "exactly how bad is this bug?" and ask to delay a new handoff until they've finished a particular cycle of testing that is unrelated to the newly modified code.) The QA/Testing group is more aware of the cost, to the QA/Testing group, of receiving a new handoff than any other group.

4.2 What to Document, and for Whom?

Groups have different needs. A good guideline for internal documentation is the rule for gift-giving, *Give what you would want to receive and a little more.*

This means that the QA/Testing group needs build logs, bug-fix lists, source librarian "check-in" logs, and the results of any regressions run as part of the delivery process. For white-box testing, it will need access to the source code; for black-box testing, it will need all of the end-user documentation. This level of documentation can be stifling, and unless there is an easy way to deliver it to the QA/Testing group, software deliveries will rarely include it.

The Technical Support group needs information about the stability of the product and early warnings about areas of the product that have changed or that will generate customer questions. Informal documents, such as written strategies for trouble-shooting and lists of fixed bugs and known anomalies, can save hours of customer time.

*There is a fear of **process**, because many engineers believe that it is a synonym for **paperwork**. When automated and delivered electronically in a format that is helpful to the downstream group, however, this process makes the engineering stronger.*

5 CONCLUSIONS

The developer who is given the task, to "build this and hand it off to QA, someone else will do it next time," has little continuity to build upon. Each time someone does it for the first time he/she is starting from scratch with no process or organizational memory. **This is a very inefficient (and costly) way to do business.**

This argues strongly for automating and documenting those mechanisms, to create that "organizational memory." He/she cannot improve something the first time because of the learning curve, and automating the process of build (including release, package, and regress) makes this possible because it allows the next iteration to build a "scaffold" upon prior work.

Part of the engineering effort is the engineering itself. By strengthening the ties between groups, and remembering to ALWAYS ASK THE DOWNSTREAM GROUPS WHAT THEY NEED IN ORDER TO DO THEIR WORK EFFICIENTLY AND WELL, the process is fortified.

Without this formalization, it is hard to keep the roles straight. Formalizing the handoff criteria does not force the groups to be more distant from one another, and does not bring them closer together. What it does, however, is make it possible to let the Development group concentrate on developing software, and the QA/Testing group concentrate on testing software, with little interruption/interference during the focused phases of the engineering work.

© 2003 Piccolo Engineering, Inc. All rights reserved. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.