Agile Deployment

Best Practices for Risk-Free Deployment

http://www.buildmonkey.com

Table of Contents

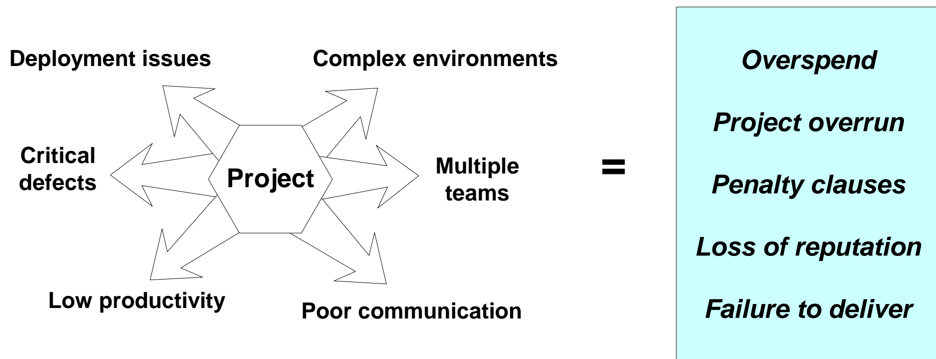
Table of Contents....2
Overview....3
The Problem To Be Solved....4
No Release, No Revenue....4
Software Tools Are Only Part of the Answer....5
Configuration Management Best Practices....6
It All Starts Here....6
Configuration Management is not Version Control....6
Source Code....7
Data....7
Database Structure and Contents....7
Application Configuration....7
Environment Configuration....7
The Point of Labelling....8
Don't Label If There Is No Point....8
Management Information....9
Build Best Practices....10
Building is not Compiling....10
Don't Throw Out The Baby With The Bathwater....10
The Architecture Drives The Build....11
Management Information....11
Deployment Best Practices....12
Deployment Is Not Installation....12
The Environment Is a Refactorable Component....13
Environment Verification Testing....14
Regression Testing....14
Configuration Management....14
Automate, Automate, Automate....16

Copyright © John Birtley, 2004
All Rights Reserved.

Overview

The cost impact to a company of a failed project can be severe indeed. The impact on the reputation of the project manager can be disastrous.

Software project management is not easy, and it requires considerable skill to successfully manage the many different risks that conspire to de-rail a project:



Numerous methodologies are available for mitigating these risks – PRINCE2, RUP, DSDN, eXtreme programming – and these have helped to some extent.

This document introduces Agile Deployment – a set of best practice and tools which automate software delivery.

In any software project, deployment is a milestone on the project plan that is usually associated with payment – or staged payment. Through the course of development, any number of problems can arise to derail efforts to reach this milestone.

Agile Deployment is based on many years of experience at the sharp end of projects, understanding what has worked and what has not, and the lessons learned from each.

Competent practitioners, and experienced project staff, will find resonance with many of the contents of this document and may find themselves saying "*this is just common sense*". This is certainly true, but the main problem with common sense is that it is not as common as people think it is.

Agile Deployment is an attempt to bring that common sense together in a single location, as a coherent set of best practices supported by proven tools to help you to ***release on-time, on-budget, with no defects***.

Copyright © John Birtley, 2004
All Rights Reserved.

The Problem To Be Solved

No methodology has yet focused on the component that all development projects share – the “build”.

One of the reasons for this is that the term “build” is interpreted differently by different people:

- The development team sees it as compilation and assembly;
- The integration team see it as the bringing together of all of the components in the application in a format suitable for release;
- The deployment team see it as something which produces the artifacts that they have to install and configure;
- The testing team see it as something which produces the artifacts that they have to test;
- The Project Manager sees it as an opaque step that nobody is entirely responsible for;;
- The end customer should not see it at all;

The BuildMonkey view is that the build is the combination of processes and technology that take software from design to deployment – where the return on investment starts to be seen.

It is clear that a methodology is required to de-risk development projects and to standardise use of the term “Build Management”.

Best Practice:	**“*Build Management*” encompasses everything from compilation, all the way through to release to the customer.**

No Release, No Revenue

Any Finance Director knows that development is just an activity that needs to be tolerated in order to produce something that will deliver a return on investment.

It may sound strange, but a large number of software developers do not appreciate and embrace this basic truth. This is in part due to their closeness to the application being constructed.

A common problem faced by development projects is therefore that it is the software developers who manage the build. This creates a situation where the

Copyright © John Birtley, 2004
All Rights Reserved.

build is focused on the needs of development, and is not geared towards releasing the output of coding such that business value can be realised.

Build Management should therefore focus on the end result of development – a return on investment – and ensure that all of the inputs to the process are incorporated in pursuit of this goal:

Best Practice:	**Focus on the end, and accommodate the inputs**

Software Tools Are Only Part of the Answer

Software projects are a complex set of interdependent people and teams and can be likened to a convoy of ships. A convoy has to move at the rate of the slowest ship. Increasing the speed of a single ship in the convoy will not increase the speed of the convoy – it will simply increase the amount of wasted capacity in the speeded-up ship.

Speeding up the slowest ship will, however, have a positive effect since the whole convoy can now move faster.

Many Project Managers try to improve productivity by implementing some degree of automation in development projects – particularly in the area of the build – and often purchase "*magic bullet*" build software that provides this.

Simply using automated build software does not improve productivity any more than the example above improves convoy speed - as it only increases the speed of a single ship in the convoy.

There is no point in speeding up development, if the target production infrastructure cannot keep pace – this just increases the inefficiency. A lot of organisations make this mistake – highly agile development processes trying to feed into considerably less agile deployment processes. The result is inefficiency, waste and over-run.

Before considering using an automated build tool it is essential to ensure that the inputs to, and outputs from, the build can cope with the improved build speed. It is imperative to ensure that the processes and technology employed are geared towards taking the project to a successful conclusion – on-time and on-budget.

Best Practice:	**Don't rely on software tools alone, they may solve symptoms whilst creating problems elsewhere**

Copyright © John Birtley, 2004
All Rights Reserved.

Configuration Management Best Practices

Software Configuration Management (SCM) is a relatively mature discipline with much written about methodologies and techniques, and these will not be recreated here.

We will focus instead on leveraging the SCM repository, and the facilities that it offers, to further the goals of the project rather than to consider SCM in its own right.

It All Starts Here

The SCM repository - how it is managed and used – is the keystone of good build management and successful delivery of projects.

It is, and must be, the single Source of the Truth for all intellectual property and digital artifacts for projects and organizations.

The SCM repository is the slave of the project, not the other way round. It should be solid and reliable, yet flexible enough to accommodate the needs of new projects. Project Managers should not have to retrofit their planning to accommodate an inflexible SCM setup.

If used correctly, the SCM repository will enhance productivity, and minimize risk, through being able to provide exactly what the project – and project management – require. If used incorrectly, it can cause delay and slippage through having to do things inefficiently further down the chain.

Best Practice:	**The SCM repository is the slave of the project, not the other way round.**

Configuration Management is not Version Control

Most software developers regard the SCM repository as a massive storage area where they simply check versions in and out – a common cause of problems.

Simply checking things into an SCM repository is not Configuration Management any more than karaoke is opera.

A well-maintained SCM repository is so much more than version control, and should provide:

- The ability to recreate any identified baseline, at any time;
- Meaningful statistics on what is changing, when and by whom;

Copyright © John Birtley, 2004
All Rights Reserved.

- Management information, such as "*how many new defects were introduced by the refactoring of component 'X'?*"

In order to be truly effective in a project, the SCM repository should store all of the artifacts that form part of a baseline or a release.

Source Code

Most development projects simply store the code that is being developed and their use of the SCM repository is no more sophisticated than this.

Data

Most applications nowadays are not just source code. Take the example of a modern computer game – the vast majority of the code base is made up of artifacts other than code such as audio clips, pictures and movie clips.

Database Structure and Contents

Where an application relies on a database this database will have a schema and structure that may change from release to release – this schema must be captured.

There will normally also be seed data for the database which should be considered as part of the baseline.

Application Configuration

In a large distributed application, the software being developed will sit atop a number of pieces of software (e.g. application servers, web servers and message queues).

The configuration of these underlying applications have an effect on the quality – or otherwise – of the software being developed and should, therefore, be considered part of the baseline for a release.

Environment Configuration

The underlying environment and infrastructure is a key component of the project, particularly in the case of distributed applications.

Such banal considerations as DNS zone files, user account information and system parameters have to be considered as some of the moving parts which affect the application and therefore be placed under configuration control.

This is of particular importance when there is more than one environment involved in the project (e.g. a development environment and a separate test environment) since the question of "*are these environments the same?*" crops up again and again.

Copyright © John Birtley, 2004
All Rights Reserved.

Best Practice:	Everything that can be changed, and affect the behaviour of the application, is a candidate for configuration control

The Point of Labelling

It is a common misconception that simply applying labels, or tags, to the SCM repository creates a baseline but this is only partly true without corresponding records of:

- What label has been applied;
- When that label has been applied;
- Why it has been applied (i.e. what milestone, or other external event, the label is associated with);

The use of a naming convention for labels can deceive even further. For example, a project that uses a date-based labeling convention (*dd_mm_yyyy*) will have several labels of the form (*03_05_2004*, or *09_06_2004*) and will reasonably assume that they have some kind of record of the baseline on those dates.

But what was happening in the project on those dates? Was the *03_05_2004* label applied immediately before a release to test, or immediately after?

Best Practice:	Labels should be used to identify and inform about events in the project

Don't Label If There Is No Point

This may seem like stating the obvious, but there should be a reason for a label being applied – the whole purpose of labeling is to identify some event in the development cycle that may need to be re-visited.

To this end, labels can be divided into two categories:

- Point-in-time
  Associates particular versions with a particular calendar date, or other event that is fixed in time (e.g. *MONTH_END_JAN_2004*, or *RELEASE_1_0_1*);
- Point-in-process
  Associates particular versions with events in the project that may recur at a later stage (e.g. *LATEST_RELEASE*, or *CURRENTLY_IN_UAT*);

Copyright © John Birtley, 2004
All Rights Reserved.

Best Practice:	**Every label should have a point, whether point-in-time or point-in-process**

Management Information

The job of the Project Manager, ultimately, is to bring the project to a successful conclusion. If this were an easy task that happened by default, then there would be no need for a Project Manager.

In order to be able to do this job well, a Project Manager needs information. He needs to know what is going on in the project – who is doing what, who is working on which components, and a wealth of information can be obtained from a well-managed SCM repository:

- What is changing in the environment – what has changed since a given point in time[1] or how often particular elements are changing[2];
- Who is changing things in the environment;
- Why things are changing in the environment;

Of course, the final item in the list requires that committers are using descriptive comments to indicate why they are marking a particular change. A well-managed SCM repository should enforce this.

Best Practice:	**The SCM repository should provide meaningful, and accessible, management information**

[1] Or point-in-process
[2] An often-undervalued metric – if there are three files which **always** have to change in order to fix defects or add new functionality then the code base may need refactoring

Copyright © John Birtley, 2004
All Rights Reserved.

Build Best Practices

As explained at the beginning of this document, the term “build” means different things to different people. The most common interpretation is the one used by developers, where the term “build” describes the compilation and assembly step of their development activities but this narrow interpretation is a common cause of problems and over-runs, on development projects.

Building is not Compiling

At the outset of the project, the Project Manager will ask the question “*how long to set up the build?*” and a developer – thinking of compilation and assembly – will answer something like “*1 day”* – a task and duration which is then duly marked on the project plan and development begins.

Later in the project, when it is time to start deploying and testing the application, this “build” needs to be refactored to accommodate the deployment and test tasks. In doing so, it turns out that the way the application is being assembled is not conducive to it being deployed or tested correctly – so the compilation and assembly staged need to be refactored as well.

In the meantime, the development team sits on its hands whilst the “build” is refactored to accommodate the needs of the project – valuable time is lost whilst the deadline continues to advance.

Best Practice:	**Know what will be required of the build before starting to write the scripts**

Don’t Throw Out The Baby With The Bathwater

From a build perspective, projects with similar architecture (both in terms of the team and the application) will have similar attributes. There will obviously be some changes required, but these will tend to follow the 80/20 rule to a large degree.

For example, a web application that is being developed by an in-house team and that will be deployed to a Tomcat servlet container and Oracle database will follow largely the same steps and require largely the same deployable artifacts.

A good SCM repository will enable the latest versions of boiler-plate build scripts for such an application to be found. These can be used almost off-the-shelf – meaning that development can start apace without having to wait on the build to be constructed for similar applications.

Best Practice:	**Well-crafted builds are re-usable and should be re-used**

Copyright © John Birtley, 2004
All Rights Reserved.

The Architecture Drives The Build

Following on from the previous section, it should be clear that the architecture of what is being developed – and the structure of the team(s) developing it – will dictate how the build should look.

There is little value to be gained in trying to retrofit build scripts for a computer game (developed in machine code by 12 people all in the same room) into a project to produce a large J2EE application with development occurring at six different sites around the world.

Best Practice:	**Well-crafted builds are flexible, but a "one-size-fits-all" approach can be costly**

Management Information

There are a number of people who need information that the build can provide:

- The Project Manager needs to track overall progress against defined milestones – number of outstanding defects, whether a committed release date will be met etc;
- Development leads need to be sure that the code is of the quality that they require – test reports, bug analysis patterns, code metrics, document and UML generation etc;
- The deployment team need to know that the artifacts will work in their target environment, and that the environment is as the application expects it to be. They also need to know whether two (or more) environments are "*the same*";
- The test team need to have confidence that they are testing against a known baseline, and whether defects that they see have been rectified in development (or whether they are re-appearing after already being fixed);
- Everybody needs to be able to communicate effectively using the same language, and have a common terminology for release versions – particularly if there are multiple threads of development;

A good build infrastructure will provide all of the above information, and more besides.

Best Practice:	**The build should tell all project participants what they need to know**

Copyright © John Birtley, 2004
All Rights Reserved.

Deployment Best Practices

Considering that it is generally an important milestone on a project plan, normally resulting in payment or a staged payment, deployment is one of the most overlooked areas of software development.

The normal course of events is:

1. Release artifacts are created;

2. Some installation and release notes are cobbled together in the form of a README;

3. The deployment team work frantically to install and configure the application – the testing team (or, worse still, the customer) are idle and unproductive in the meantime;

4. Some symptoms are found which are suspected to be application defects;

5. The development team blame the environment;

6. The deployment team blame the application;

7. Repeat (5) and (6) *ad nauseam*.

When a documentation team are also considered - responsible for creating documentation that the end user will need to install, configure and use the application – the situation becomes even more difficult.

This situation can be avoided by planning for deployment from the beginning. Deployment is an inevitable part of software development, yet it always seems to take people by surprise.

Best Practice:	**Know that deployment is inevitable, and incorporate it into the automated processes**

Deployment Is Not Installation

As part of normal development activities, artifacts are installed into sandbox environments – and test environments – many times. But this is not deployment, this is installation.

In order to get an application into its production environment, be that an internal environment or on hosted-infrastructure, a number of hurdles must be overcome:

Copyright © John Birtley, 2004
All Rights Reserved.

- The application must pass UAT;
- The application must be installed and configured correctly:
- All pre-requisites for the application must be satisfied;
- The end customer must accept the handover;

Deployment is that point in the life of an application where it starts to produce a return on investment. "*Launch*", "*Go-live*", "*Release*", "*First Customer Shipment*" are all phrases which describe the same event.

Best Practice:	**Deployment is the point where an application starts to provide a return on the development investment.**

The Environment Is a Refactorable Component

This point cannot be stressed enough, particularly in large distributed applications.

Every application, large or small, has a runtime environment in which it operates. In a simple desktop application, this is a standalone machine (such as a PC). In larger applications, this will be a combination of machines (e.g. an application server and a database) operating together to provide the runtime environment.

In either case, the application expects certain facilities to be available from the runtime environment and will function incorrectly – or cease to function – if these are not present.

The environment itself, whether standalone or a network, contains many moving parts that can be independently configured. IP addresses, or hostnames, can be changed. User privileges can be modified or revoked. Files and directories can be removed. Each of these can have an effect on the way that the application behaves.

In an environment that is owned and managed internally this can be bad enough. In an environment that is owned and managed by an external third party, and where project success is contingent upon successful UAT in that environment, this can be disastrous.

Best Practice:	**Be able to identify whether the deployment environment is as prescribed, and "*fit for deployment*"**

Copyright © John Birtley, 2004
All Rights Reserved.

Environment Verification Testing

One of the most common questions that arises in development projects containing more than one environment is, simply, "*are these environments the same?*" and its answer can be elusive.

It is essential to be able to answer that question – quickly and accurately – so that any perceived defects in the application can be categorised as "*defect*" or "*environmental*".

This ability becomes particularly poignant where on or more of the environments are owned by different teams, or organisations.

Best Practice:	**Be able to prescribe what the deployment environment should look like, and have a capability to test it quickly.**

Regression Testing

The environment, as explained earlier, is a refactorable component. It can be changed, and parts can be moved or deleted. However, unlike application code, changes may need to be made to the environment in response to external events (e.g. hardware failure, or security policies).

Applications, particularly complex ones, use regression tests to ensure that observed behaviour after a change is exactly as it was before the change was made. The same should be true of the environment.

Best Practice:	**Automated regression tests for the environment that will compare observed behaviour both before and after changes are made.**

For example, suppose that a number of operating system patches or service packs are applied to an environment where the application has been, or will be, deployed. How are these tested? Do you wait for users, or testers, to start calling to say that there are problems?

Or do you make sure that you know what problems have been introduced before your users do?

Configuration Management

As stated earlier, the SCM repository should be used to store any artifact that can be changed and that may have an effect on the environment.

It may not seem obvious, but some of the most obscure environmental changes can cause an application to fail:

Copyright © John Birtley, 2004
All Rights Reserved.

- Hostname resolution;
- Non-existent user or group accounts;
- IP and network connectivity;
- Existence, or otherwise, of files and directories ;
- Application or operating system configuration files;

It is essential that these environmental variables be placed under configuration control and able to be identified as part of a baseline.

Best Practice:	**Environmental artifacts that are not part of the application should be part of the baseline**

Copyright © John Birtley, 2004
All Rights Reserved.

Automate, Automate, Automate

Every single task that is performed as part of a development project – throughout the entire lifecycle – can be placed into one of two categories:

1. **High Value Work**
   Tasks which require some form of human judgment. Spending more time on these tasks results in more output or better quality. Development and testing are examples of high-value work;

2. **Low Value Work**
   Tasks which do not require human judgement. Spending more time on these tasks does not result in more output. Deployment is an example of low-value work;

Tasks which fall into the first category can use some degree of automation, but should stop and wait for human intervention wherever judgment is required.

Tasks in the second category must be automated. There is no value in having expensive resources employed to do mechanical or repetitive tasks that a computer could do more quickly, accurately and consistently.

Best Practice:	**Automate anything that does not require human judgment**

A note of caution - it may be tempting to think that automation will increase productivity on its own, but this is not necessarily the case. Automating an inefficient process will simply magnify its inefficiency – as explained in the section on ***Software Tools are Only Part of the Answer***.

This, and it is worth repeating, is a common error – to assume that automated tools alone will improve productivity.

Best Practice:	**Do not automate inefficient processes, or you will only maximize the inefficiency**

Copyright © John Birtley, 2004
All Rights Reserved.