Agile Flow of Change

Perforce Consulting Guide

Learn how to achieve scalability and increase productivity in a rapid environment.
# Table of Contents

Introduction ................................................................................................................................. 1

Defining the Flow of Change ........................................................................................................ 1
  Flow of Change Principles ........................................................................................................... 1
  Other Development Models ......................................................................................................... 2
  Accounting for Change ................................................................................................................ 2
  Retire Inactive Streams ............................................................................................................... 2

Benefits of the Flow of Change Principles ................................................................................... 3
  Promotes Collaboration ................................................................................................................. 3
  Promotes Transparency ................................................................................................................. 3
  Maintains Velocity ......................................................................................................................... 3
  Ready to Deliver ............................................................................................................................ 4

Flow of Change and Application Lifecycle Management ............................................................... 4

Scaling the Flow of Change .......................................................................................................... 5
  Transparency Revisited ................................................................................................................. 6
  Productivity and Automation ......................................................................................................... 6

Managing Dependencies .............................................................................................................. 7
  The Versioning Service’s View ..................................................................................................... 8
  The User’s View ............................................................................................................................ 8

Conclusion ..................................................................................................................................... 9

Learn More .................................................................................................................................... 9
INTRODUCTION

Achieving rapid successful development across an enterprise is difficult. One of the key challenges is maintaining a rapid, efficient flow of change: enabling concurrent development and managing complex dependencies, with minimal burden to developers.

Agile development processes aim to deliver increased business value via rapid, purposeful change. Agile teams focus on iterative development of working software driven by stories, or realistic requirements.

Many organizations start by adopting Agile for a small team working on a single product. After initial success, the next move is often to bring Agile to much larger teams working on bigger, more complex projects.

In this document, see how to use Perforce Software Version Management to achieve an Agile flow of change for teams and projects of any size, as well as to facilitate both the collaboration and transparency that are so vital to Agile development.

Understand how change flows, learn best practices for managing the flow of change, and see how to maintain a rapid, efficient flow of change in large, complex Agile environments.¹

DEFINING THE FLOW OF CHANGE

Picture streams of development as streams of water, and individual software changes as individual drops of water. If the flow is well managed, the flow will push each drop where it needs to go. Just as dams, floodgates and levees might be erected to manage the flow of water, similarly, software development teams need to understand and manage the flow of many changes that issue forth from the minds of programmers.

In general, streams are sets of files that evolve at the same time for a specific purpose, such as maintaining an older release or adding a new feature. Streams often contain one or more modules of a product, and their composition is defined in Perforce by stream views. The flow of change is the process of managing the propagation of individual software changes among a series of related streams of development. For example, a bug fix made in a release maintenance stream should be propagated in such a way that the same bug doesn’t reappear again in future releases; a new feature in a shared library must become available to all consumers of that library.

¹ For more on managing the flow of change with Perforce, see the Google TechTalk by Perforce VP of Engineering Laura Wingerd, “The Flow of Change” (October 2007).

Flow of Change Principles

The mainline model is a scalable, effective way to manage a variety of development practices. The mainline or trunk lives as long as the project exists, and contains a stable version of the source files at all times.² New streams start from the mainline directly or indirectly, and all change is eventually accounted for in the mainline.

Figure 1 depicts five streams organized according to the mainline model. The relative stability of streams is depicted in the hierarchy of the streams: More stable streams, such as the rel2.1 release stream, are shown above less stable streams, such as the dev2.3 development stream. The stream model also indicates the intended flow of change between the streams. The arrows represent the pathways for the changes flowing between streams. When change flows from a more stable stream to a less stable stream, we merge the changes together. Merging may require manual resolution of conflicts and evaluation of the suitability of changes for the target stream. Merging introduces the risk of instability, which we accept in the less stable stream. The merge pathways are indicated by the arrows with merge lanes.

When points of stability are reached, change can flow to a more stable stream. We copy or promote the changes to the more stable stream after first making sure that all outstanding changes in the more stable stream have been accounted for. Or in other words, we perform one last merge down before the copy up.

After the last merge down, which has resolved any potential conflicts, the less stable stream represents exactly what we want in the more stable stream, so the copy up, or promotion process makes the more stable stream identical to the less stable stream. The copy pathways are indicated by the arrows without merge lanes.

This merge down, copy up principle is the guiding principle for managing the flow of change. In the simple stream model, development streams merge down from the mainline and copy up when stable milestones are reached (see Figure 2). Bug fixes on release streams merge down to the mainline and changes are rarely merged up to release streams. In more complex models with a deeper stream hierarchy, the merge down, copy up principle still guides the flow of change.

The implementation of an agile flow of change can be managed manually for small teams and projects, but requires more powerful tools when scaled out to larger teams and products. We will revisit this point in detail in a later section.

² Laura Wingerd, Practical Perforce, (Sebastopol: O’Reilly Media, 2006).
Other Development Models

Some development efforts, such as website development, do not follow the mainline model. Instead, a promotion model is used where change is promoted rapidly from a development stream to a published or live stream, perhaps with a review stream as an intermediate step (see Figure 3). Changes are rarely made directly on the live stream.

Suitable for the circumstances of many websites, this model still follows the merge down, copy up principle for propagating change.

Accounting for Change

The mainline model and the merge down, copy up principle account for changes but do not imply that we always accept all of the changes when merging. As part of accounting for a change, part or all of the change may be ignored, either as part of the merge process or by building rules into the stream model.

Retire Inactive Streams

Clearly communicating when a stream is inactive avoids the unnecessary work of propagating change to streams that no longer need it. In Perforce streams, you can prevent a stream from accepting new changes.

For example, a bug fix on an older release may no longer be relevant to the mainline, so the bug fix can be ignored when merged. If an entire module of the older release is obsolete, it can be isolated via the stream view.
BENEFITS OF THE FLOW OF CHANGE PRINCIPLES

Although the principles for managing the flow of change described so far work well in any environment, they have particular benefits for Agile development.

Promotes Collaboration

An effective stream model allows developers to work in isolation when necessary, letting individuals or teams focus on their tasks without interference. When changes must be shared, the merge down, copy up principle makes the process as effective as possible, and keeps risk in less stable streams.

Notification tools, as provided by the Perforce review daemon or external systems, further assist collaboration, as team leads and release managers can easily stay in touch with active development.

Promotes Transparency

Having streams where work can proceed in isolation encourages developers to work within the system, rather than managing work locally. Other developers and teams can inspect, compare, and merge the work in a stream; release managers can quickly see if changes must be propagated from one stream to another.

Visualization tools such as the Stream Graph provide an easy grasp of the stream model and the flow of change, which encourages the use of streams and further promotes transparency.

Developers can also drill down into the evolution of a single file using the Time-lapse View tool (see Figure 4).

Maintains Velocity

An effective flow of change maintains velocity. By creating a release stream for stabilizing a release, you allow development of new work to continue in the mainline. By creating development streams for different featuresets, you allow each team to work as rapidly as possible, with no delays if another team’s schedule slips.

Creating new streams is rapid and easy in Perforce’s streams framework, and allows new projects and features to be developed quickly. It only takes a few clicks of the mouse for the release manager to define and create a new stream. The development team then quickly creates workspaces and starts working.

Shelving gives developers the confidence to utilize multiple streams, as work-in-progress can be saved before switching to a different task or stream.
Ready to Deliver

Keeping a mission-ready stream at all times is an important tenet for Agile developers. This could be the mainline for classic release-and-ship workflows, or a production staging stream for a hosted application workflow.

Keeping a stream mission-ready involves the application of continuous integration technologies to test each software change as it is made. Once suites of unit, smoke, regression, and other tests are available, they can be applied to streams as needed. The continuous integration system should allow different combinations of test suites to be easily targeted at any stream.

FLOW OF CHANGE AND APPLICATION LIFECYCLE MANAGEMENT

Several parts of the Application Lifecycle Management (ALM) toolchain help to maintain a rapid flow of change. Integrations between Perforce and other ALM tools are available in leading integrated development environments (IDEs) like Eclipse and Visual Studio, which encourage adoption and use by developers, and help to maintain team velocity.

- Achieving transparency often starts by integrating software version management with
- Continuous Integration (CI) systems to build, deploy, and test software.
- Workflow management/defect tracking systems to provide insight into sprint status.
- Code review systems to help encourage technical best practices.

Linking software version management with CI engines that build, deploy, and test software helps to catch problems early and produce builds quickly. Some of the testing is narrowly focused at the functionality associated with specific backlog items. Passing such targeted tests is needed to change the status for a backlog item from a red to a green indicator. Sophisticated use of CI tools may even include automatically promoting code to more stable streams when tests pass. Working in the Perforce streams framework, this sort of automation is easily accomplished.

Providing meaningful status on each backlog item is essential to promoting transparency. Sprint management tools help with status and task delegation and enhance the build and test status reported by the CI engine. Integrating software version management and workflow or defect management systems within the Eclipse IDE helps to track the reason why work is done, allows for a better understanding of the impact of tasks on source code, and provides insight into the progress of a sprint with status updates of work submitted and tasks completed (see Figure 5). Code reviews can be a time-consuming and intensive process. Code review systems facilitate collaboration and transparency by exposing work to a team of subject matter experts. Integration with software version management allows much of the mechanical work to be automated, increasing productivity and acceptance of code reviews. For less formal processes, Perforce facilitates informal code reviews via shelving.
SCALING THE FLOW OF CHANGE

Agile development is characterized by rapid iterations that deliver concrete chunks of usable software that provide business value. Overhead—anything that does not directly contribute to business value—must be minimized.

Consider a team working on a small software project called Jam. The team has one release in maintenance, and is working on the upcoming release. In this case, it is fairly easy for the team to manually keep the streams in sync. Bug fixes for the older release are merged to the mainline and then to the development stream at regular intervals, and stable development work is promoted to the mainline for the next release. The overhead of the flow of change between the streams, as the arrows between streams, is fairly small (see Figure 6, left).

Now consider a related product called PeanutButter (PB), the perfect complement to Perforce’s Jam product. PB involves several teams who are maintaining older releases and working on several new features in parallel. The need to keep an always-ready stream at all times is one of the drivers for a stream model that includes features like integration branches and patch branches (see Figure 6, right).³

As the PB teams adopt Agile, they want to maintain the same rapid iteration and low overhead as the Jam team. When a sprint ends for the “db” team on the upcoming 2.0 release, the release manager must do several things:

• Merge the latest bug fixes from the 1.0-r into the 1.5-r, then main, 2.0-int, and down to the dev-db stream.
• Have the db team build and test with the latest bug fixes.
• Promote the stable milestone from the db stream.
• Build and test the code.
• Merge the integration stream into the dev-gui stream.

These steps must be easy, routine, and automated as much as possible. Otherwise, at least two teams will be unproductive as they wait for the end-of-sprint cycle to complete.

³ For more on the streams structure of complex development environments, see blog articles by Tom Tyler, Using Branch Diagrams to Scare Your CEO (June 2010) and Perforce Directory Standard (PDS) (June 2010).
Transparency Revisited

Visibility into the project is particularly important when dealing with large teams and products. The Stream Graph quickly provides several pieces of information (see Figure 7, left):

- The intended flow of change from oldest release to newest development.
- A visual indicator if there are any pending changes that must merge from stream to stream.

This transparency helps the release manager and other developers understand the steps that must happen when a sprint ends or a release patch is coming through the pipeline.

Several other Perforce features provide similar functionality. P4Eclipse’s MergeQuest provides a stream or branch model in Eclipse (see Figure 7, right), while the Revision Graph focuses on the evolution of a single file (see Figure 8).

Productivity and Automation

Besides transparency, the Stream Graph also provides easy access to the Perforce operations necessary to maintain the flow of change. These operations are easily scripted using Perforce’s Command-Line Client or supported APIs. Automation reduces the time and effort involved in these routine supporting tasks, preventing the buildup of technical debt in large projects.
Technical debt accrues when changes originate in one stream and are not rapidly propagated. This stagnation can eventually slow the pace of development, as time must eventually be devoted to propagating old changes. As changes stagnate, the work involved in merging them increases. Even worse, failing to propagate a bug fix may lead to fixing the same bug multiple times in different ways.

Automating software version management systems with powerful merge engines, such as Perforce, reduces stagnation by automatically merging changes along certain pathways. For example, the merge pathway from a release stream to a mainline stream is likely to contain only bug fixes. This implies that changes along that particular change propagation path involve relatively small chunks of text, with very little refactoring or complex changes. Such changes tend to merge well with automated tools. In those cases, a strategy of using automated merges is employed successfully by some organizations.4

The ease of workspace switching is also of great benefit to the release manager and other developers. When performing the merges, the release manager must work in different streams. The stream graph allows her to quickly move her workspace from stream to stream with a drag-and-drop sequence, after which the versioning service efficiently updates her workspace to match the new stream. If a typical workspace contains many thousands of files and gigabytes of data, this workspace reparenting saves considerable time and disk space compared to having one workspace per stream. Of course, a developer who needs to switch streams to fix bugs or help with a merge enjoys the same benefit.

4 For a more detailed discussion of the merits and risks of automated merging, see blog article by Tom Tyler, Living on the Edge—Automated Merging (September 2009).

MANAGING DEPENDENCIES

As Agile spreads to different teams in the organization, the natural dependencies among different teams, modules, and components must be managed effectively. Different modules of a product may be produced at different intervals, so often they are produced independently and later assembled into the final product. This allows for more rapid, stable iterations of some modules.

Consider four modules produced by different teams, and their relationships (see Figure 9):

- PB is a complex product, and each sprint lasts 4-6 weeks. PB is the actual product that ships.
- Jam is a smaller, simpler module, and each sprint is only 2 weeks.
- Jamgraph is an older library and is required to build PB, but is in maintenance mode, and the maintenance team does not use Agile. A new Jamgraph version is available infrequently.
- The web team produces online documentation used as product help. The web team follows a promotion model, with live versions promoted nightly.

Before adopting Agile methods and letting PB and Jam develop independently, the Jam team was developing more rapidly against a moving PB target. The Jam team would introduce several new features, which could not be released until the PB teams were ready for their release.
Now the product as a whole is more Agile—it is iterating more rapidly and delivering business value more quickly. Each time a Jam milestone is reached, a new version of PB can be released that incorporates those new features built against a stable PB framework.

The Versioning Service’s View
On the service, each of the four modules has its own location in the repository, and evolves independently (see Figure 10).

![Figure 10: Service’s view of modules.]

The User’s View
A developer needs a coherent working copy of the four modules in order to build and test effectively (see Figure 11, left). Building and maintaining that view must be easy and fast, in order to let developers work and build efficiently against the correct set of modules.

In the stream view that translates the versioning service view to the developer’s view, the stream view allows a product architect to define the relationships to other streams and other modules (see Figure 11, right). For example, the PB architect can indicate that the db development stream uses:

- The gui component from the integration stream.
- The 7.4.2 version of the Jamgraph library.
- The latest released Jam module.
- The nightly web pages.

Similarly, the release manager for the 1.5.1 patch release can periodically adjust the stream view to include a newer version of the Jam module.

Once the stream view is defined, all the developers receive a correct, updated workspace view automatically. Any child streams also inherit the proper dependencies. The overhead of dependency management is minimized, increasing developer productivity and team velocity.

The stream view can easily be visualized using HTML5-based applets (see Figure 12).
CONCLUSION

As Agile processes spread across the enterprise, maintaining a rapid flow of change is critical to delivering business value. The challenges in maintaining that flow of change increase as teams and projects grow in size and complexity.

Perforce provides solutions for problems commonly encountered when scaling an Agile flow of change. Concurrent development and complex dependencies require a software version management system that handles routine tasks easily, offers automation, and integrates with other tools.

By following these suggestions for an Agile flow of change, teams will spend more of their time implementing stories and tasks and delivering business value.

LEARN MORE

Read about Perforce Streams in P4 Blog articles:
perforce.com/blog.

Watch Perforce videos at perforce.com.

- Streams in Perforce (35:16 mins.)
- Perforce Streams: Adding Intelligence to Branches (2:34 mins.)
- Perforce Streams: Solving Developer Challenges (3:23 mins.)
- The Flow of Change (56 mins.)