WHITE PAPER

Agile Practices and Installation Development

FLEXERA® SOFTWARE

Agile Practices and Installation Development

Both commercial and custom application development projects are increasingly using agile processes to deliver software. Agile processes are characterized by multiple iterations and frequent delivery of working software to users, who provide rapid and almost continuous feedback on the scope and direction of the application. This approach enables teams to successively refine the look and feel of software, while also making rapid changes to feature sets in response to changing business opportunities.

However, the ability to create and manage a flexible installation package is a critical part of meeting the goals of agile projects. Good installation development solutions enable teams to rapidly deliver new builds into a single package that allows customers to easily install and uninstall, supporting several of the most important goals of agile development practices. Agile teams focus on delivering software iteratively with a positive user experience and seek an automated solution for developing installation packages.

Business Drivers of Agile Development

The business environments that are served by the many software applications available tend to change rapidly. Business groups are driven by market opportunities, regulatory and legal changes, new business relationships, and staff changes. Often these occur with little or no forewarning, and windows of opportunity tend to close quickly. New applications or new features to existing applications are often needed in order to take advantage of these opportunities.

These realities apply to packaged commercial applications as well as in-house custom applications. Customer needs change quickly, and missing out on those needs may drive them to seek other alternatives.

An application can commonly take a year or more in development and test prior to release. Even adding new features to an existing application typically takes at least several months of effort. In traditional methodologies, requirements are frozen at the beginning of the process to make for a predictable development effort, with little or no further input from the user community until the software is delivered. While there is sometimes a period of beta testing just prior to final delivery, the purpose of beta testing is to identify defects, not change features or look and feel.

If user or business needs change in the interim, there is little that a development team can do to adjust to new requirements. In most cases, changing requirements or adding new requirements adds a significant amount of additional time and money to the development effort. In other words, traditional ways of developing software weren't designed for the needs of business today.

These business realities have led development teams to adopt agile techniques in order to better meet the needs of the user community. Agile development enables teams to:

- Get important features into the hands of users more quickly by iterating development on a few features at a time.
- Make mid-course corrections based on changing business needs, in response to changes in user needs during the course of the project.
- Fine-tune features and user interfaces with regular user feedback, based on the iterative releases.

Even teams building commercial packaged software find agile methodologies useful. In addition to being able to deploy a few compelling features quickly, it provides

for a mechanism to adjust the feature set based on new technologies or large shifts in their target markets.

Commercial application developers usually do not seek to satisfy a single customer or group of users. But timely feedback from multiple customers can help product managers and development teams make adjustments on subsequent iterations in response to feedback from early adopters willing to work with multiple releases.

Common Characteristics of Agile Processes

There are many different agile methodologies, including Scrum, Feature-Driven Development, and Extreme Programming. Choosing a specific agile methodology often depends on the previous experience of team members. In practice, many teams research agile methodologies and pick and choose individual techniques and practices that best match their skills and culture.

While agile methodologies vary in a number of details, there are several common characteristics. First, agile teams develop software in a series of iterations, typically ranging from one week to one month. This enables teams to focus intently on only a few features, for a limited period of time, and deliver those features to the best of their ability. Successive iterations are planned just prior to execution. They may be used to add additional features, make changes to existing features, or address defects or other limitations.

Agile methodologies place a premium on delivering working software at the end of each iteration. This means that the team is building early and often, finding and fixing defects in real time, and making sure that most builds result in software that can be installed and used. Working software requires enabling the user community to easily install and start using the software, and to replace it seamlessly once a new iteration becomes available.

Daily or continuous integration of each developer's code support the ability to deliver working software. Each integration is verified by an automated build to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

Agile methodologies also depend on talented and motivated developers. Often developers who make up agile teams are specialists who work in different geographic locations, and collaborate at a distance. These professionals are able to work independently on specific components while also ensuring that everything is able to work together when built.

Limitations of Agile

Despite the apparent benefits, agile methodologies can also create challenges development teams must overcome. Probably the most significant challenge is delivering software that cleanly installs and uninstalls at the end of each iteration. In many cases, Registry keys are either skipped entirely, necessitating temporarily setting paths or privileges. Or users are required to manually move files around and set individual Registry keys.

Frequently, either the team delivers a zipped set of files that has little organization or installation integrity, or it works an inordinately long time after each iteration to ensure that installation is safe and convenient for users. Neither alternative represents an efficient use of time or resources.

Because agile teams may be geographically distributed, development must be planned out carefully, enabling individual developers to write code to a defined set of interfaces. In most cases, geographically separate developers or small teams may each hold parts of the installation code, which must be put together seamlessly in order for the installation project to build and execute successfully. Even with close coordination, initial builds and installation success requires a lot of back-and-forth between team members.

As a result, agile teams tend to struggle with installation. This is especially a problem in agile development, with its many delivery requirements and little ability to move around tasks when delays occur. These limitations call for a better way to develop installation projects for applications that are developed iteratively and delivered to users in several successive releases.

Best Practices for Application Installation

Installation is a vitally important component of an agile development effort. Without the ability to quickly and easily install and uninstall successive versions of an application, it is impossible to obtain user feedback on the quality and direction of those features. Further, difficult or time-consuming installation and uninstallation will cause users to become reluctant to continue to use the software, depriving them of the immediate benefits of new features.

Software teams who practice agile development, therefore, desire best practices in order to offer the best user experience, deal with fewer support calls, and achieve greater acceptance of their software. Teams should plan for installation as a part of scoping and designing the application, with a determination of the files required, privileges and locations of those files, required Registry keys and locations for those keys, directories used, and components to conditionally install. Once the design is complete, install should be developed concurrently with one of the first iterations.

As development teams strive for the first successful build of a new project, they must also focus on the complete install package for that build. Even adding files or other objects at a later time should be straightforward as long as the fundamental architecture is solid.

Because agile teams are likely employing continuous integration practices, installation should also be a part of that practice. In addition to integrating, building, and testing all software on an ongoing basis, the team should also integrate and build the necessary installation packages.

Finally, teams should leverage automation where possible. According to the principles of the Agile Manifesto, simplicity in learning and use is essential. Therefore any automation has to be simple, flexible, and configurable. By choosing the right tools to enable automation of specific activities, an agile process can be significantly improved.

Agile Installation and InstallShield

Packaging and installation should support the organization's business objectives. In many cases, this means making multiple applications and components available through a single installation process. Packaging and installation should also simplify development, licensing, and maintenance, rather than making it more complex. Flexera Software's InstallShield, the longtime leader in installation development for Windows applications, enables traditional and agile development teams to address their installation authoring needs so they can deliver software iterations quickly.

InstallShield provides the flexibility required by fast-moving agile teams in creating and offering application installation options. For example, release engineers can use the InstallShield Standalone Build to automate nightly builds of installations and help development teams to implement continuous build integration. Because nightly or continuous builds are an essential part of any agile practice, the Standalone Build ensures that the project is ready to deploy once an iteration is completed.

The teams can set up the nightly or continuous build process to include not just the application builds, but also the installation builds. The build can include a single application, or, with the new suite support in InstallShield 2012 Spring, a suite composed of several separate applications, as well as associated prerequisite components. The installation is ready for regular testing throughout the development process, and when development and testing are complete, is ready for distribution.

The InstallShield automation interface enables team members to use a script to add new files, add or delete features, initiate the build process, and change product name and upgrade code, release settings, summary information stream items, release flags, and similar information.

The newly enhanced InstallShield 2012 Spring Collaboration add-on enables individual development team members to contribute to the development of the installation simultaneously throughout each iteration. Each developer or tech writer can create on one or more developer installation manifests (DIMs)—feature-sized collections of related items such as application files, shortcuts, registry entries, and other elements that together make up a discrete, logically separated portion of the installation. Teams can integrate DIMs into multiple installation projects, reusing them as needed, enabling efficiency.

The agility of InstallShield is demonstrated by the ability of teams to create an integrated package that serves the needs of a wide range of customers and users. These include the ability to easily bundle multiple products together into a single, unified suite installation eliminating the need to develop a complex custom launcher or bootstrapper application, incorporate both 32-bit and 64-bit install packages, give end users the option of running suite installations with a user interface or silently, allow the installation developer to specify whether to show a single entry for the entire suite in Add or Remove Programs, and provide that developer with complete and centralized control over the user experience design.

With these features, agile teams can easily set up installation projects concurrent with initial iterations, and keep installations up to date as they plan and execute subsequent iterations. A typical install development scenario starts with individual members of the team working on installation units that correspond to their development responsibilities. As the team members are working independently, their installation components have to work together when integrated. Those components combine as units of the integrated installation package.

InstallShield features and capabilities enable these teams to meet their application delivery needs on schedule and with high quality. Whether the project is an individual application or a complex suite of products and components, InstallShield enables teams using agile or traditional development processes to quickly and accurately build installation packages, integrate those packages if needed, and meet business objectives by providing customers and users flexibility in selecting and licensing only the needed applications.

Summary

As agile development teams seek to deliver the right software to users early and often, installation can no longer take a back seat to features and bug fixes. Teams can plan for installation as they begin the project, and use automated tools such as InstallShield to prepare and manage the installation package for iterative releases. The result is the ability to deliver a great initial user experience with multiple iterations of an application, while easing the workload and stress on the development team.