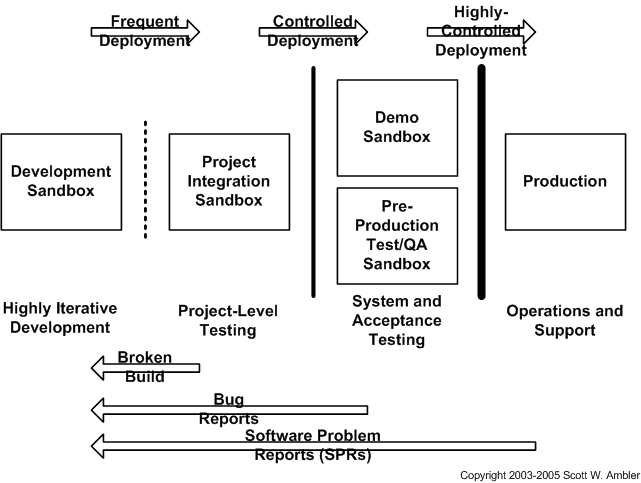
System deployment is a complex endeavor which is a critical aspect of the software development lifecycle (SDLC), an endeavor that is all but ignored by writers in favor of sexier topics such as distributed object development, components, or the latest version of an SDK. If you can't get software into the hands of your users then what is its value? Absolutely nothing. The following tips and techniques should help to make your system deployment efforts successful:

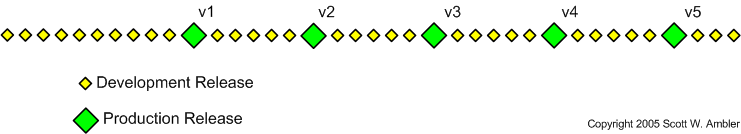
1. **Identify and understand your deployment audience**. There are at least three distinct groups that you need to consider: your end users, the [operations](http://www.enterpriseunifiedprocess.com/essays/operationsAndSupport.html) staff responsible for running the software once it is in [production](http://www.enterpriseunifiedprocess.com/essays/productionPhase.html), and the [support](http://www.enterpriseunifiedprocess.com/essays/operationsAndSupport.html) staff who is responsible for aiding your users with the software once it is in production. You need to identify the level of control that each group has over your actual deployment. Can one group stop your deployment if you don't meet their specific [requirements](http://www.agilemodeling.com/essays/agileRequirements.htm)? For example, it is quite common to discover that operations departments have defined criteria for the release of new software, criteria that your deployment approach must meet. I once worked for an organization where the users were unionized and any software that was deployed to them had to be first accepted by their union representatives, otherwise you couldn't ship. Early in your project you need to identify the deployment hoops that you need to jump through to be successful.
2. **Identify your deployment strategy early**. Will you run the new system in parallel with the existing system or will you perform a cutover? Running the system in parallel offers the advantage that you can easily back out to the original system if the new one runs into problems. However, parallel operations requires significant effort on the part of everyone involved: Your users need to do double entry, operations staff need to run both systems, support staff need to support both systems, and development staff may need to create integration code that temporarily works behind the scenes to synchronize data. For many systems, particularly ones supporting online customers via the Internet, a cutover is your only option - few customers would be willing to place their book order with both Amazon version N and with Amazon version N+1. With a straight cutover you will need to plan for a downtime period in which the cutover occurs, anywhere from a few seconds to a few hours, or even a few days depending on the nature of the system being deployed.
3. **Installation testing**. Just like you test your application, you should also test your installation scripts. A good way to do this is to develop your installation scripts as you develop your system, and use them to promote your software between your team integration sandbox into your [pre-production testing environments](http://www.agiledata.org/essays/sandboxes.html) as shown in [Figure 1](http://www.ambysoft.com/essays/deploymentTips.html#Figure1).

**Figure 1. Development sandboxes.**



1. **Release regularly**. Agile software developers typically deliver development releases at the end of each iteration into [pre-production staging area(s)](http://www.agiledata.org/essays/sandboxes.html) (see [Figure 1](http://www.ambysoft.com/essays/deploymentTips.html#Figure1)). A development release of an application is something that could potentially be released into production if it were to be put through your pre-production quality assurance (QA), testing, and deployment processes. Granted, this won't be true earliest development releases because you won't have delivered sufficient functionality to make deployment worth your while. Furthermore at the beginning of a project you often stub out interfaces to shared services - such as security, persistence, or even reusable legacy functionality - so technically you still have some clean up to do before you're ready to release to production. This is why in [Figure 2](http://www.ambysoft.com/essays/deploymentTips.html#Figure2) you see that the first production release often takes longer to deliver than subsequent releases; in the first release of a system you likely need to get a lot of the "plumbing" in place and your team likely hasn't "gelled" yet enabling them to become efficient at collaboration. The first production release may take you twelve months to deliver, the second release nine months, and then other releases are delivered every six months. An early focus on deployment issues not only enables you to avoid problems it also allows you to take advantage of your experiences during development. For example, when you are deploying software into your staging area you should take notes of what works and what doesn't, notes that can serve as the backbone of your installation scripts.

**Figure 2. Regular software releases.**



1. **Start planning early**. Deployment can be quite complex, especially when your user base is physically dispersed or there is a wide range of system configurations, you often find that you need to start planning early in your project lifecycle to be successful. There is nothing worse than rushing to finish software on time only to have it put on the shelf to await installation due to lack of deployment planning.
2. [**Take an agile approach to planning**](http://www.ambysoft.com/essays/agileProjectPlanning.html). Keep your plans high-level at first, focusing on the major dependencies. The people doing the work should plan the details, and you don't need to worry about the details until you're about to do the work.
3. **Reduce your pre-production testing efforts safely over time**. It's of little value to deliver working software on a weekly basis if there is a three-month pre-production testing process which it must go through each time. With agility you'll find that you can reduce the amount of time invested in your pre-production testing efforts, and thereby speed up deployment. Why is this? Agilists produce high quality artifacts and have the regression test suite to prove that their system actually works. This higher level of quality implies that you'll be able to do less testing during system deployment. Once you get really good you may find that you can in fact deploy working software into production on a weekly basis - I've seen this at a very large financial institution and a dot-com so it is possible, regardless of what the traditionalists may tell you.
4. **Well-tested software is easier to deploy**. [Object-oriented testing](http://www.ambysoft.com/essays/floot.html) is a complex and difficult endeavor, but one that you must undertake. I have several fundamental philosophies regarding testing. First, test early and test often. Two, if you can build it you can test it. Three, if it isn't worth testing it likely isn't worth building.
5. **Your plan must reflect your organizational culture**. You need to understand whom you are deploying software to, their current work practices and policies, the amount of change they are willing to tolerate, and how your software will affect them once deployed. I've found that the larger the organization the more difficult it is to deploy new software due to cultural inertia. Most people have a tendency to sit back and wait to see who else is using a product, which is often contrary to the nature of most software developers who will jump at the chance to work with new software.
6. **Update your deployment plan regularly**. This helps to set realistic expectations by both your project team and your deployment audience.
7. **Work backwards when deployment planning**. Envision the system in production - users are working with it, operations professionals are keeping it going, support staff are helping users to work with the system, and developers are potentially evolving the system to address defects and to implement new requirements - and ask what needs to occur to get me to that point. Then compare that vision with where the current environment is today, something called deployment gap analysis, to determine what needs to occur to successfully install your system. My experience is that forward planning generally isn't sufficient when it comes to system deployment because your requirements often do not reflect issues pertinent to the production phase of your system, the need to operate and support your software is often assumed and therefore not explicitly reflected in your requirements.
8. **Have go/no-go decision points during the installation process**. A good deployment plan includes go/no-go decisions points during the installation process. If at defined times during the installation you have not reached a certain point in the overall installation process you will rollback your efforts and try to install again at a future date. This is a critical concept for projects that have very stringent deployment requirements, typically software that is replacing existing mission-critical systems that currently run your business.
9. **Develop "de-installation" scripts**. The ability to back out becomes critical with this approach, requiring investment of additional resources to develop and test de-installation scripts. In fact, many deployment efforts fail miserably because the development team didn't bother to consider how to back out their new system, and then discovered that they had to do so because of an unforeseen problem during installation. Never assume that your deployment efforts will go smoothly.
10. **Remember the data**. Data conversion is often a key task for the deployment of a new software system, a complex effort that should be started early in your project. [Analysis](http://www.agilemodeling.com/essays/agileLegacyIntegrationModeling.htm) of your [legacy data](http://www.agiledata.org/essays/legacyDatabases.html) needs to be performed: the legacy data sources need to be identified, the legacy schemas should be modeled, and official sources of each attribute need to be chosen for data stored in several places. You need to understand your existing legacy data so that you can convert it to your new schema
11. **Negotiate with legacy system owners early**. During development you will discover dependencies between your system and other systems, dependencies that should appear on your deployment model. These dependencies affect your deployment plan because they imply the order in which updates to the various systems (if any) must be installed. You will need to negotiate deployment efforts with the other project teams that own the systems you have dependencies on, effectively a cross-project effort. If the interface to your system changes, or if you require new ways to access their systems, then they may also need to release all or part of their systems before yours.
12. **Announce the deployment**. You should announce the anticipated deployment schedule, including both the expected training and installation dates. Your announcement could be an email to your customers, a press release, or even advertisements on television, in magazines, or on the Internet. Release announcements are important because people want to know how the new system will affect them, if at all, and when it will affect them. Respect the fact that your deployment efforts are disrupting the lives of your stakeholders.
13. **Hold regular release meetings**. The closer you get to deployment, the greater the need to hold regular release meetings with key players involved in the actual deployment. This includes quality assurance staff to advise everyone of the testing status of your software, developers to advise you of the current rework status, operations staff to inform you of current production issues of the existing environment, and support and user management to inform you of their training statuses.
14. **Understand the associated risks of deploying systems**. Because software projects have a tendency to be delivered late and over budget, there is often spectacular pressure exerted by senior management on the project manager to forgo training. Without training, your users won't be able to effectively use your software, your operations staff won't understand how to operate your software, and your support staff will have difficulty properly supporting your system. Documentation is typically at risk on most projects, once again due to being late and over budget as well as a general lack of knowledge among developers as to how to write effective documentation. Poor documentation results in significantly higher maintenance and support costs for your system. The greatest risk is often data conversion, a very complex and critical task that is often ignored until it is too late.
15. **Don't underestimate the opportunities for reuse**. Other projects may have already developed [deployment models](http://www.agilemodeling.com/artifacts/deploymentDiagram.htm) that reflect your target platform, models that you can evolve as needed for your own use (ideally your [enterprise architects](http://www.agiledata.org/essays/enterpriseArchitecture.html) should be able to provide these models). Furthermore, templates for deployment plans, installation procedures, and templates for operations, user, and support documentation can also be reused from other projects. Remember, [you can reuse more than just code](http://www.ambysoft.com/essays/typesOfReuse.html).
16. **You may need to upgrade your user's existing environments**. These upgrades could include changes to existing hardware, operating systems, databases, or middleware. If you don't know the current status of your technical environment you will also need to plan for activities to perform the required legacy analysis. If you are upgrading your database, you may be upgrading the database software itself or the schema of your database, you will need to include data conversion activities in your deployment plan. Physical considerations should also be considered when performing environment upgrades. Is sufficient power available? Are working areas such as desks and cubicles in place? Does sufficient room exist in the building(s) that you are deploying to? Do sufficient network drops exist for new workstations?
17. **Training is an important part of deployment**. Remember that your stakeholders may need training beyond that of learning how to work with your application. For example, this may be the first time that some users are working with a PC, a browser, or even a mouse. Similarly, this may be the first time that your operations staff is working with a new technology that your system users, such as an EJB application server, and therefore will need to be trained and educated in that technology to qualify them to work with your system.
18. **Develop supporting documentation**. Another important effort is the development of operations, support, and user documentation, the potential artifacts of which are summarized in [Table 1](http://www.ambysoft.com/essays/deploymentTips.html#Table1). Depending on your corporate documentation standards you will likely find that you need to include one or more technical writers on your team to develop this documentation (hopefully in an [agile manner](http://www.agilemodeling.com/essays/agileDocumentation.htm)).

**Table 1. Potential documentation requirements.**

|  |  |
| --- | --- |
| **Customer** | **Potential Needs** |
| Operations | * Backup procedures * Batch job and printing requirements * Data extraction/sharing requirements * Installation procedures * Resource requirements * Release notes |
| Support | * Contact points within development and operations * Escalation procedures * Support call recording process * User documentation (all versions) * List of new features |
| End Users | * Reference manual * Support user's guide * Tutorial manual * User manual * List of new features |