You can find more information about each of these techniques online via the list of references I have provided. There is a lot of online as well as offline text available on these methodologies. I personally prefer SCRUM development, which is a Chaos Theory based approach. In SCRUM, we have *sprint*, which is an iteration with a certain number of days (for example, 30) at the end of which the development team covers a certain set of use cases and lets the stakeholders see and test the application. During the next iteration, they cover more features, which are prioritized accordingly. Each iteration passes through a full software development cycle: planning, requirements, design, coding, testing, and documentation. The goal is to have an available release (without bugs) at the end of each iteration.

A major goal to be achieved by using this process is to allow the client to take their new product to market before it is completed in its entirety. We are also minimizing risk by developing highly-focused components in a short period of time. This development process will continue throughout all phases, promoting the release of components to a beta phase in a live environment as the life cycles are completed. The following are the major benefits or principles of the SCRUM method:

- Keeping things simple by chunking (or batching)
- Customer satisfaction through rapid, continuous delivery of useful software they can get their hands on
- Working software is delivered frequently (taking weeks rather than months)
- Working software is the primary measure of progress
- Late changes or additions in requirements are welcome and can be added to iterations with ease
- Close, daily cooperation between clients and developers
- Continuous attention to technical excellence and good design
- Regular adaptation to changing circumstances

Because the stakeholders can see and review the current application at the end of each sprint, it gives them a valuable opportunity to change anything they don't like. Changes made at a later stage (such as when all use cases are covered) would take a longer time to absorb into the application, and can sometimes derail the project completely.

Project Transition and Release

At the end of the last iteration, the project will be in *alpha stage*, which means that all of the main use cases are implemented. The alpha build of the software is the build delivered to the software testers, usually internal to the organization/community developing the software. Usually, no end users see this phase. Alpha stage software is never completely bug free, but functionally covers all use cases.

Once the alpha testing is over, the project moves to the beta phase, which means that external users/end users (outside the company or organization that developed the software) can now start checking the system and using it.

A **beta version** is the first version released outside of the organization or community that developed the software, for the purpose of evaluation or real-world testing. Beta level software generally includes all of the features, but may also include known issues and bugs of a less serious variety. Once the beta phase is over and all major bugs have been fixed, the project is in an RTM (Release To Manufacture) stage, or in the Gold Edition.

The following is a summary table showing all of the important project phases:

Project Stages	Project Pitfalls
1. Initiation: Understanding project needs from a very high-level perspective and conducting a small feasibility study	A poor feasibility study can hinder or block project progress later on; it is very important to see what really can be achieved and what cannot, given the current technological offerings
2(a) Planning: Understand the project needs comprehensively, develop business use cases, detailed project plan, high-level architecture, class diagrams, data model, sequence diagrams, prototype	Detailed planning and project management is the key here; without a well laid out plan the project is doomed to fail. Wrong estimates or an improper choice of architecture can sound the death knell for the project's progress
2(b) Development: Work iteratively on selected use cases, QA process follows	Lack of unit tests, deviations in architecture, patching and short circuiting code to avoid missing deadlines
3. Transition: Beta testing, release docs, deployment instructions, bug fixing	Without proper usability and integration testing, success is not possible
4. Support: Provide support after the Gold/RTM release	Good support is very crucial

Tiers and Layers

There is a misunderstanding that tier and layers are two different names for the same entity. The concept of tier and layers came into being with the need for identifying and segregating different parts of an application into separate connected components. This separation can be at two levels:

- Physical Separation
- Logical Separation