

Digital Notes



**AGILE SOFTWARE DEVELOPMENT**

**UNIT – V**

**Agile Methods with RUP and PRINCE2 and**

**Tools and Obstacles Development**



**Agile Methods with RUP and PRINCE2 and Tools and Obstacles**

* Agile modelling and RUP
* FDD and RUP
* Agile methods and prince2

**Tools to help with agile development**

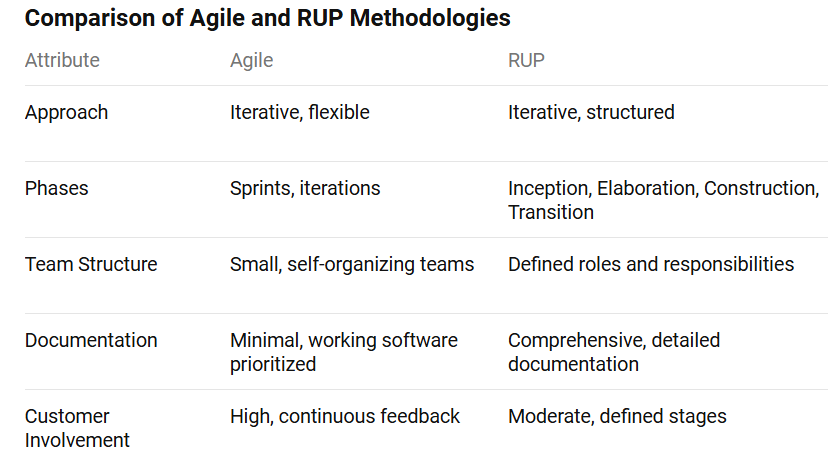
* Eclipse: An agile IDE

**Obstacles to agile software development**

* Management intransigence
* The failed project syndrome
* Contractual difficulties
* Familiarity with agility

**Agile modelling and RUP**

* Agile and RUP (Rational Unified Process) are distinct software development methodologies with unique characteristics and approaches

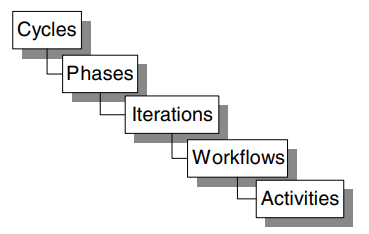




* The Unified Process or the Rational Unified Process (sometimes known as RUP) is a framework for handling the whole lifecycle of a software development project. It is referred to as a framework because it allows the overall suite of activities to be customised and modified as required for a particular type of application
* Why then is the Unified Process called a process and not the Unified Framework? It is called a process because its primary aim is to define:

1. Who is doing what?
2. When they do it?
3. How to reach a certain goal (i.e. each activity)?
4. The inputs and outputs of each activity

* It is thus an engineered process. In fact, it is comprised of a number of different hierarchical elements
* In terms of the agile movement, it is towards the heavyweight end of software methods and thus may not initially appear compatible with processes such as those that have come out of the agile movement.
* The Unified Process is actually comprised of low-level activities (such as finding classes), which are combined together into disciplines formally known as workflows (which describe how one activity feeds into another).
* These disciplines are organized into iterations. Each iteration identifies some aspect of the system to be considered.
* Iterations themselves are organized into phases. Phases focus on different aspects of the design process, for example, requirements, analysis, design and implementation
* In turn, phases can be grouped into cycles. Cycles focus on the generation of successive releases of a system
* This process is iterative and incremental and is adaptive in that it is responsive to changes in business or user requirements as well as to feedback from users.



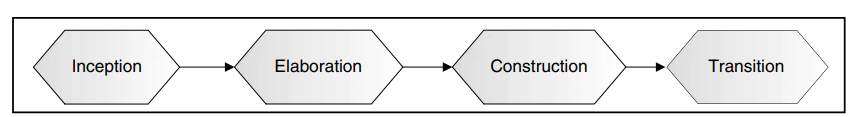
Key building blocks of the Unified Process



* **Overview of the Unified Process – The Four key elements of RUP**
* There are four key elements to the philosophy behind the Unified Process. These four elements are:
  + Iterative and incremental,
  + Use-case driven,
  + Architecture-centric,
  + Acknowledges risk.
* *Iterative and Incremental*. The Unified Process is iterative and incremental, that is, the design process is based on iterations which address either different aspects of the design process or more the design forward in some way. This does not mean that the Unified Process is a process based on rapid prototyping. Any prototypes that are developed in the Unified Process are used to explore some aspect of the design. This could be to verify some architectural issue, different design options, assess a new technology, etc. Indeed, the use of an iterative and incremental approach in the Unified Process requires more planning as compared to approaches such as those based on the waterfall Essentially, the following holds with the iterative approach in the Unified Process:
  + You plan a little.
  + You specify, design and implement a little.
  + You integrate, test and run.
  + You obtain feedback before the next iteration.
* The end result is that you incrementally produce the system being designed. While you do this you explicitly identify the risks to your design/system Unified Process front and deal with them early on (see later). Note that this does not mean that you are hacking the system together nor are you carrying out some form of rapid prototyping (you are not). However, it does mean that a great deal of planning is required, both upfront and as the design develops
* *Use-Case Driven* The Unified Process is also use-case driven. Remember from earlier that use-cases help identify who uses the system and what they need to do with the system (i.e., the top-level functionality). Thus, use-cases help identify the primary requirements of the system. One problem with many traditional approaches is that once the requirements have been identified there is no traceability of those requirements through the design to the implementation. Instead, designers must refer back implicitly to the requirements specification and make sure they have done what is required of them. This is then verified by testing. In the Unified Process, use-cases are used to ensure that the evolving design is always relevant to what the user required. Indeed, the use cases act as the one consistent thread throughout the whole of the development process.
* For example, at the beginning of the design phase, one of the two primary inputs to this phase is the use-case model. Then explicitly within the design model are use-case realizations that illustrate how each use-case is supported by the design. Any use-case that does not have a use-case realization is not currently supported by the design



* **Lifecycle Phases**
  + The Unified Process is comprised of four distinct phases. These four phases focus on different aspects of the design process. The four phases are Inception, Elaboration, Construction and Transition.
  + The four phases and their roles are outlined below.
    - Inception. This phase defines the scope of the project and develops the business case for the system. It also establishes the feasibility of the system to be built. Various prototypes may be developed during this phase to ensure the feasibility of the proposal. Note we do not focus on the development of the business case in this book; it is assumed that the system to be designed is required and a business case has already been made.
    - Elaboration. This phase captures the functional requirements of the system. It should also specify any non-functional requirements to ensure that they are taken into account. The other primary task for this phase is the creation of the architecture to be used throughout the remainder of the Unified Process.
    - Construction. This phase concentrates on completing the analysis of the system, performing the majority of the design and the implementation of the system. That is, it essentially builds the product.
    - Transition. The transition phase moves the system into the users environment. This involves activities such as deploying the system and maintaining it.

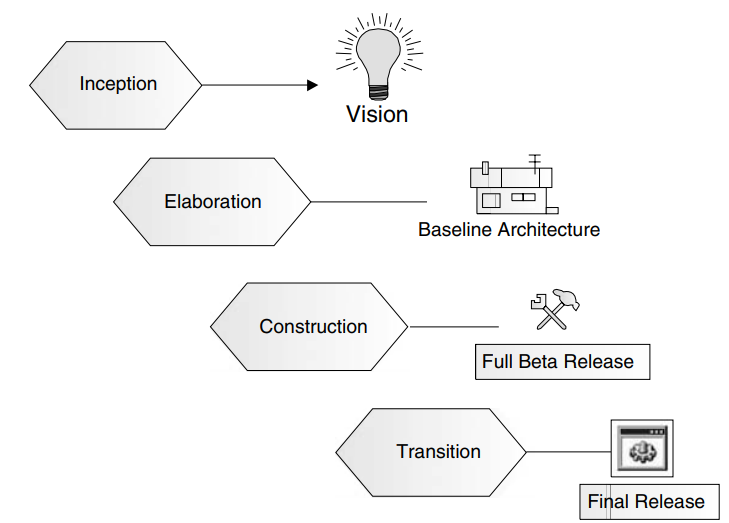


Four phases of the Unified Process.

* Each phase has a set of major milestones that are used to judge the progress of the overall Unified Process. The primary milestones of the four phases are illustrated in below Figure





****

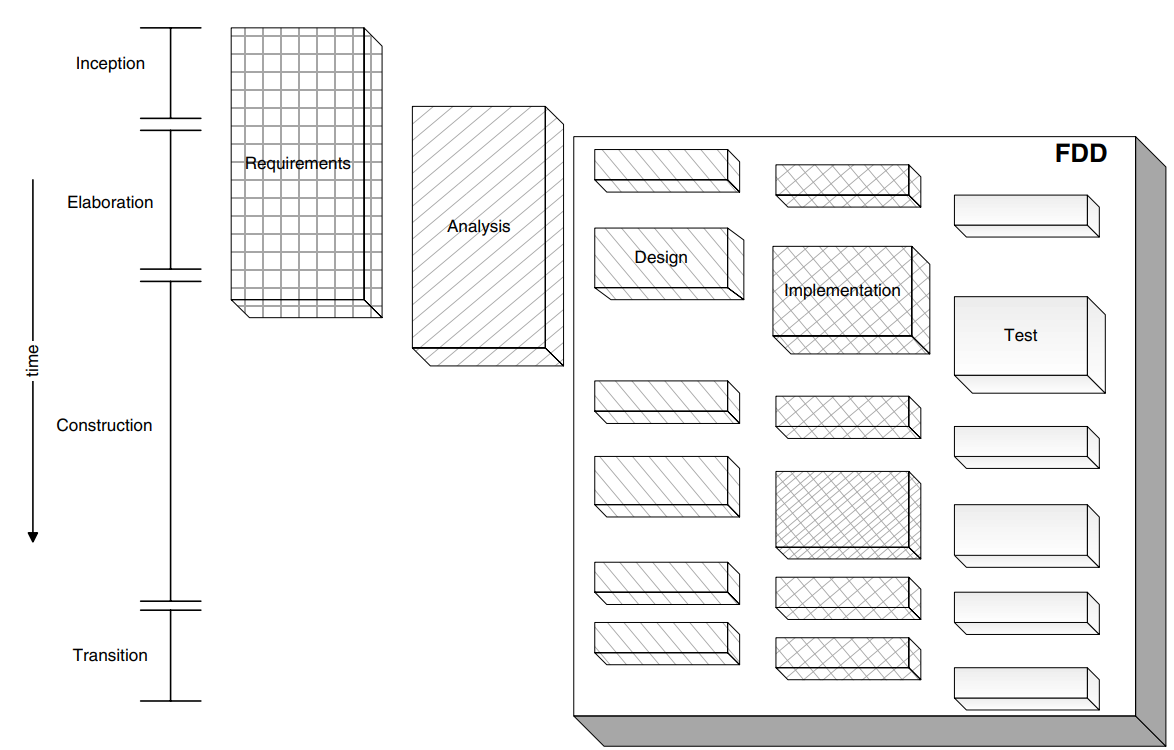
Major deliverables of each phase

* The primary milestones for each phase are:
  + Inception. The output of this phase is the vision for the system. This includes a very simplified use-case model, a very tentative architecture and the most important or significant risks are identified and the elaboration phase is planned.
  + Elaboration. The primary output of this phase is the architecture along with a detailed use-case model and a set of plans for the construction phase.
  + Construction. The end result of this phase is the implemented product that includes the software as well as the design and associated models. The product may not be without defects as some further work has yet to be completed in the transition phase.
  + Transition. The transition phase is the last phase of a cycle. The major milestone met by this phase is the final production quality release of the system



**FDD and RUP**

* Agile Modelling is not the only agile methodology that can be usefully applied to a software project that is employing the Unified Process.
* Feature-Driven Development is another method with things to offer an RUP project.
* One area in which we have found it beneficial to amend the Unified Process is to make it also Feature-Centric.
* Feature-centric means that each iteration centers on the identification and realization of system features.
* A feature is a schedulable requirement associated with the activity used to realize it.
* These requirements may be user-related requirements, application behavior requirements or internal requirements.
* The features can then be grouped into work packages that can act as the basis of the planning required to monitor and manage the software development process.
* Another aspect that can be applied to the Unified Process from Feature-Driven Development is the application of fixed timescale iterations.
* Clients like to know when they will get the next release. This means that they can plan their own acceptance testing, deployment and training schedules.
* Associated with the concept of fixed timescale iterations, we have found that clients are very willing to consider the priorities of features and to order them. This means that if we find that an iteration will be unable to implement all the features originally planned for that iteration, we are able to work with them to determine which features will move to the next iteration or be returned to the list of features that will one day be required.
* In fact, what we have tended to do is to a wrap up a Feature-Driven Development project within the outer wrappings of a Unified Process project. This means that the Unified Process gives support for early project initiation activities, highlevel management support and ongoing software support operations. In turn, the Feature-Driven Development aspects allow us to focus on the primary task of the project, providing software that adds value to the clients within the timescales and budgets available
* The Feature-Driven Development process essentially handles the steps mentioned in RUP. This is illustrated in Figure below where the design, implementation and test disciplines have been subsumed by the Feature-Driven Development process. Thus, we do not have a big up-front design, followed by the implementation and then the testing



Integrating Feature-Driven Development with the Unified Process



**Agile methods and prince2**

* The PRINCE2 method describes how a project is divided into manageable stages enabling efficient control of resources and regular progress monitoring throughout the project.
* The various roles and responsibilities for managing a project are fully described and are adaptable to suit the size and complexity of the project, and the skills of the organization.
* Project planning using PRINCE2 is product-based, which means the project plans are focused on delivering results and are not simply about planning when the various activities on the project will be done.
* Because an iterative or incremental approach may at first sight appear less controlled than an approach such as PRINCE2, some have perceived that this means that PRINCE2 and such approaches are inconsistent.
* However, this is not the case. Indeed, if you consider the emphasis of the Feature-Driven approach, then similarities with some aspects of PRINCE2 can immediately be seen. For example, product (or feature)-based planning, the involved partnership of users and developers and the strong emphasis on the underlying business need (or case)
* Those who have used PRINCE2 to control their Feature-Driven, timeboxed project have found that an unyielding approach – applying the method straight from the manual – can lead to duplication, overlap and conflict.
* Table below makes it clear that the iterative approach includes some project management content while in turn PRINCE2 maintains a view of the whole project.
* In addition, the iterative approach fixes the length of an iteration in terms of time and then attempts to achieve a prioritized list of features.
* The general philosophy to combining a PRINCE2 approach with an iterative one is that where there is no overlap between the methods, you must refer to the appropriate approach. In general, this means that project management related issues will be handled by PRINCE2 and development related issues by the iterative approach.

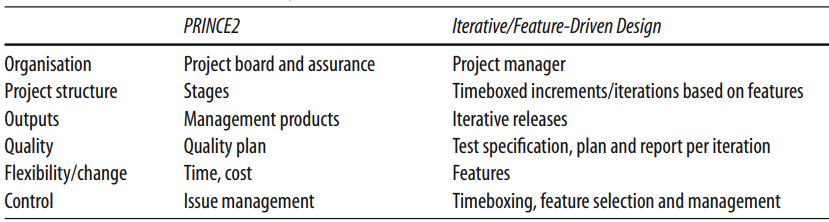


Table -PRINCE2 and FDD relationships.

* Managing and controlling an iterative project using PRINCE2 is fundamentally the same as for any other PRINCE2 project. The purpose is to enable each level of the project management team to:
  + Demonstrate to the next level up that the project is on track to a successful outcome (that the project will deliver products that are fit for business purpose on time and within budget).
  + Identify early anything that may prevent this.
* To do this, there are mechanisms for controlling and tracking both the PRINCE2 aspects and the products of an iteration. In PRINCE2, management and control are done at each project stage and everything depends on how the project is broken up into stages.



**Tools to help with agile development - Eclipse: An agile IDE**

* **What Tools Do You Need?**
* So, what tools should you use in support of an agile project? To answer this question, let us first consider what tool requirements Agile Software Development imposes on us. Some of these requirements are presented below:

1. We should be able to refactor software simply and easily. For example, in Java if we move a class from one package to another, we would want package statement of classes to be updated and all references to that class to be modified, etc.

2. It should be possible to modify existing software securely in the knowledge that we can role back to an earlier version if it all goes horribly wrong.

3. We should be able to track changes in the system.

4. We should be able to run and re-run test suites simply and to review the results easily and immediately.

5. If we are undertaking to perform Agile Modelling, then we should be able to reverse engineer code into models simply and with a minimum of fuss. We should be able to modify code or models and keep both in sync. We should be able to update models without the need to heavy weight tools.

6. Ideally, we also want something that will tell us when we need to create a new build. For example, something that notices that new code has been released into the central repository and initiates the build process automatically.

* The tools listed below are those that I have found to be particularly useful in the projects I have worked on. They represent the most common areas that can benefit from tool support within an agile Java development project:

1. An IDE that can be integrated with the other tools proposed, which supports refactoring and iterative development.

2. A lightweight modelling tool to help with Agile Modelling.

3. A build tool to allow simple and rapid rebuilds of the system as and when necessary.

4. A version control system to handle the frequent and rapid changes introduced into the software and to allow the software to roll back when necessary.

5. A test framework to handle the unit tests so important to agile software development.

* **Eclipse: An Agile IDE**
* Eclipse is an extremely powerful open-source IDE that can be used for developing projects in a variety of languages. It is primarily known as a Java IDE and in this guise, it offers facilities on a par with commercial tools within an open-source framework. It can also be used for developing C, C++ and C# applications – thus, offering a common IDE for a variety of development languages.
* Eclipse was originally developed by IBM but was moved to an open-source model to widen its appeal and the base of developers working on it. Not only is it an open-source system but it is also an open platform that allows additional tools to be plugged into it to extend its basic functionality



* As an IDE, Eclipse offers the standard suite of features we expect, including:
  + Syntax analysis of code as it is entered
  + Integrated context-sensitive help system
  + Auto-complete
  + Pop-up function/procedure prototypes
  + View source of supplied components
  + Ability to run applications from within the tool
  + Integrated debugger
  + Variety of wizards for creating different types of Java element
  + Integration with version control systems (for example, CVS)
  + Sophisticated, context-sensitive, search
  + Various perspectives (including code, class, inheritance, etc.)
  + Integration with Java development tools such as ANT, Junit, etc.
* Eclipse is also fast, lightweight and relatively small. It is written in Java and is thus cross platform.
* From the point of view of an agile development approach, one of the best features of Eclipse is its support for various refactoring operations. It is possible to refactor the location of packages, classes and interfaces, to refactor class and interface names, methods, variables and constants.

**Building Applications with ANT**

* ANT is the Java build tool; it is made for Java. It allows a developer to specify what components need to be built, when and into what format.
* Eclipse has high-quality ANT integration either built-in or available as a download. Indeed, it is getting to the point within the Java world where many Java developers expect to be able to use ANT as a basic part of their development environment.
* Indeed, in many cases, if you download an open-source project, you will find an ANT script has been provided to allow you to build that project.
* Using ANT, it is possible to:
  + Extract all the current source from a version control system such as CVS.
  + Compile all the Java code in a system.
  + Automatically generate a build number, provide a build date stamp, add a version number to a property file, etc.
  + Jar that code up into a single file.



* + Copy that jar to a deployment location.
  + Generate the Javadoc for the compile system.
  + Copy the Javadoc to an appropriate location.
  + Create Web Archives (WARs).
  + Create Enterprise Archives (EARs).
  + Deploy WARs and WARs to servers (such as Tomcat or JBoss).
  + Startup servers such as Tomcat and JBoss.
* ANT is written in Java and uses XML configuration files to control its build process. These build files, called “build.xml” by default, control what ANT does and how it does it. Each build file contains one project and at least one default target. A target describes an ANT activity such as the compilation of some Java code, or the creation of a Jar file.

**Obstacles to agile software development**

* we will consider some of the obstacles that can be encountered when trying to introduce an agile approach into an organization. We will also try to suggest some approaches to overcome these obstacles
* **Management Intransigence**
* The drive towards agile software development often comes from the developers themselves or from developer teams.
* In such situations, management can represent a significant obstacle to adoption of an agile approach. This can be for a host of reasons that include:

1. Lack of familiarity with Agile Software Development methods.

2. Mis-comprehension of what eXtreme Programming and Agile Modelling offers.

3. A feeling of losing control.

4. Remoteness from the actual coalface and thus remoteness from development issues.

5. The need to feel that they have the whole project planned out in advance.

6. Lack of suitability for their own review-and-assessment process. That is, if they are assessed on a more traditional waterfall model, then they may have the production of a detailed project plan as one of their aims and objectives to be completed.

7. Belief that adopting eXtreme Programming (and thus doubling developers up) will actually halve productivity. Although current research indicates that the opposite is actually true.

8. The parapet syndrome. That is, a manager may not want to risk a different approach to that normally adopted (i.e., putting their head above the parapet), as no one may be censured for doing things the standard way, but they may well be for doing things in a different way and failing!

9. The fear of the unknown!



* **The Failed Project Syndrome**
* Having a successful agile project in another part of your organization can really help you to convince management of the benefits of the new approach.
* However, the failed agile project can of course have the reverse effect – i.e. merely proving their belief in the futility of this agile thing!
* In a bid to retain credibility with the client in general, if we drill down and find out what the issues had been with the project. It turned out that

1. No one involved in the project had ever done an agile development before.

2. Half the developers were actually industrial engineering students taking time out between their second and third years, and had limited commercial software development experience anyway.

3. The belief appeared to be that you should never comment code (it was self-documenting).

4. They never wrote any documentation (believing it not to be the agile way).

5. They never did any design (as they considered design to be the antithesis of agility).

6. Refactoring appeared to have been an area they considered a waste of time!

* **Contractual Difficulties**
* Contracts traditionally have been based around a waterfall software development approach. Thus, the buyer of the software may state exactly what is required, and if all the specified functions are not provided, then the contract is not met.
* This situation is often exacerbated by the presence of fixed price projects. That is, a supplier must state exactly how much they will charge to provide the required functionality. All of this of course takes place in advance of the software development and may not even involve the actual developers.
* In general, we have found that working with clients so that they understand some of the benefits from their side is essential to the adoption of an agile development method. In particular, we have found emphasizing the following to be useful:

1.**The level of control/influence they can exert**. That is, the extent to which the clients can influence what is done when, can sometimes be a major surprise. Software buyers often used to hand over the requirements, and then feel that they lose control of the project, and that they are left out of any project decision made internally. In an agile approach, the client and his representatives should be on board all the time.

2. **The feedback they will receive**. Returning to the issue of the involvement of the client in the project, in many cases, clients have received very little feedback on the progress of the project. Often what feedback they have received may later prove to at best have been optimistic and at worst a downright lie. As the clients find themselves in the midst of the project, they will naturally receive regular feedback. They will also be able to judge the accuracy of the information they receive as and when releases are made at frequent intervals.

3. **The adaptability of the project to changing requirements**. One regular issue for bespoke software buyers is the problem of changing requirements. If the requirements document is the basis of the contractual agreement between supplier and customer and subsequently if a requirement changes, then the contract changes. In some cases, this can be a very large hurdle to cross and will require lawyers and protracted negotiations. However, as requirements may be drawn up well in advance of the software development, they may well have changed/may well change in the future. By adopting an agile approach, as requirements change, these changes can be naturally and simply passed through the project team.



4. **The ability to prioritize features**. A (possibly) surprising advantage of the agile approach to customers is that a low priority feature in a software system can be included but only implemented if time allows or its priority rises. This can be important whether for political reasons (some senior manager thinks it is important but no one else does, so it has to be in the requirements) or in green field software where the importance of some features may be unclear at the start. As feature’s priority is continually revised, then its inclusion (or not in a particular iteration) can be reviewed

* **Familiarity with Agility**
* One major obstacle to adopting an agile software development is the lack of knowledge of how to start and run such a project.
* However, until you have been involved in one or more agile software developments, a great many questions may stop you in your tracks. For example,
  + “How do we estimate the cost of the software to the clients at the start of a project?”
  + “How do we decide how many iterations will be there?”
  + “How do we know what will be in each iteration?” and “how do we decide how long an iteration should be?”
* There are no right or wrong answers to these questions – although there are answers that may be more right or more wrong than others. We will take up each point in turn and try to discuss why each should not stop you from starting an agile software development project.
* ***How do you estimate the cost of the software to the clients****?* The issue here may appear to be a question of how can we determine how much this software will cost for us to develop, given that we don’t yet know exactly what we will do, when and how long (particularly later) functionality will take to implement. The real issue here is that the focus of the question is wrong. Customers need a budget to work within and so do you.
* ***How do we decide how many iterations there will be?*** This may appear as an obstacle because the question being implied here is “how many iterations will it take to implement all the requirements?” Whereas in fact, the issue should be “How many iterations should there be, to enable the clients to receive an appropriate number of releases, and for them to provide feedback to the development team, and to consider the current open questions?” This question implicitly acknowledges that some features may remain un-implemented as time may not allow for them.
* ***How do we know what will be in each iteration?*** Again this question arises from the more traditional waterfall mind set – it is really saying, “I want to know exactly what will be done in alliterations of the project. ” Whereas, agile methods essentially say that we will roughly decide what features, functions, use cases or user stories (depending on your preferred terminology or approach) will be planned for each iteration, but that only the current iteration will be planned in detail. Then, at the end of the current iteration, we will review the overall plan for the iterations and features, and that the next iteration will then be planned in detail. Thus, we only plan for the next few weeks or months



* **Customer Opposition**
* It is probably true that you are unlikely to be a customer or a buyer of a software system. This is because the users of the software systems we build are not, in general, software developers – they are lawyers, accountants, architects, book editors, opticians, doctors, vets, etc. Thus, presenting a customer with an agile development approach can be daunting.
* It is likely that when you present a client with an agile approach, they may be less than ecstatic. This may be for a variety of reasons including:

1. Concern that they don’t know exactly what the software will do at the end of the day.

2. Related to this, the lack of a detailed plan that they can review.

3. The level of involvement and commitment required of them.

* One really useful thing to have is a “Customer Champion” – someone to push your cause within the customer organization. This champion may well require education as to what an agile approach is and what benefits it has for the customer
* Remember, you really do need the customer on board when you get to the development stage. The on-site customer (even if a virtual on-site customer) for me is one of the key indicators for success. If you have one, then you are more likely to succeed. Not having one does not necessarily guarantee failure

