

Digital Notes



**AGILE SOFTWARE DEVELOPMENT**

**UNIT – III**

**Agile Modelling**

**AND XP**



**Extreme Programming**

* Introduction
* The fit
* Common practices
* Modelling specific practices
* XP objections to agile modelling
* Agile modelling and planning XP projects
* XP implementation phase.

**Introduction**

* Agile modeling is a collection of practices and principles that can be used to model software, and Extreme Programming (XP) is a software development methodology that's based on Agile principles
* Both are from the agile concept and both are motivated by the desire to produce better software faster.
* What is the relationship between Agile Modelling and XP? Are they complementary or contradictory?

**The FIT**

* Many of the Agile Modelling practices fit straight into XP - modelling equivalent of the XP programming
* Agile Modelling’s emphasis is modelling and XP’s is programing.
* so there are differences and there are practices that are only relevant to one of the methods or the other.
* Do the Agile Modelling practises add any value to the XP practises.
* Whether Agile Modelling actually fits in with the philosophy of XP
* Main conflict that can come across is –XP do not need to concentrate more on modelling where as Agile modelling mainly concentrates on modelling methodologies.
* Except that, **Agile Modelling is more of a philosophy than a hard and fast approach**.
* Indeed, Agile Modelling needs to be applied with some other development methodology to be of any real benefit. As such, it brings a suite of agile techniques to the process of modelling.
* XP does do modelling. However, it does not do “big up-front modelling.” Instead, modelling is performed as and when required during the lifetime of an XP project. It is at these points that Agile Modelling practices can become not only relevant but also extremely useful.
* **Common practices**
* So both Agile Modelling and XP are part of the agile movement so what practices do they have in common. Actually, they have quite a few practices in common, although the names may differ from one approach to the other.
* To illustrate this, consider the below Table. This table pairs up various Agile Modelling practices with their equivalent or near-equivalent XP practices.





* Note that as the emphasis on the two approaches differs, there is bound to be a slight variation in emphasis within common practices.
* **“Model with Others” is matched to “Pair Programming”** because the intention in both cases is that everything produced is examined and thought about by two or more people. Some differences are more obvious; in Agile Modelling we use modelling standards while in eXtreme Programming we use coding standards but again the intention is the same.
* Other Agile Modelling practices have potentially less obvious parallels within XP practises. But again if they are examined, it can be seen that they represent the same or similar intention but from the modelling perspective. These are illustrated in Table above
* For example, in Table when performing Agile Modelling you should consider how what you are modelling might be tested? How you can model to facilitate testing? In turn, the practice of displaying models publicly, on a modelling wall is akin to the open communication promoted by XP.
* Agile Modelling also encourages modelers to work in “small increments” that is similar to XP’s practice of defining a single test, implementing what is needed for that test, before continuing with the next test.
* Finally, one way of validating models created in Agile Modelling is to prove those models by implementing them. This is akin to requiring code to pass an individual test before continuing. It also fits in very well with XP, in that you may model a little bit, and then implement it before continuing with modelling which is a very XP-like approach.
* **Modelling Specific Practices**
* There are some Agile Modelling practices that may, at first at least, appear to have no place within XP at all. These practices include:
  + Model with a purpose,
  + Using multiple models and
  + Know your models.
* The only thing that tends to be discussed is the use of stand-up modelling meetings during which existing code is analyzed or new code explored through the use of diagrams. The resulting XP “models” are often thrown away as soon as they are no longer needed as they may only exist for a short time on white boards, note pads or index cards and are never translated into a design tool.
* **Model with a Purpose**
  + As many XP practitioners may see no place for modelling at all, this point may, to them, seem self-defeating. That is, there is no purpose to modelling within an XP project! However, this can be very wrong. There are a number of purposes to modelling which can be very relevant to XP
  + **“Model to Understand”** and **“Model to Communicate**”. “These two practices are considered in more detail below.
  + ***Model to understand the software***. This is relevant because, before you refactor or extend your code you must understand it. Individual methods or classes may be understandable merely by considering them in isolation.

However, due to the nature of object-oriented systems, such as those implemented in languages such as Java, it is often necessary to consider a group of cooperating classes together. This may involve reviewing how the instances of the classes are created, interacted and are discarded.

For example, when looking at part of the implementation of the ClearSpell spell checker developed by a colleague, I need to become familiar with the structure of the main spell-checking engine. To do this, I have decided to use the Omondo plug-in for Eclipse to generate a class diagram for myself. This is done within Eclipse by creating a new UML diagram.

Once this is done, Omondo finds out which classes and interfaces are in the selected package. It then presents a selection dialog to the user, which allows them to select which classes and interfaces they actually want to include in the diagram and what type of relationships to show.

* + ***Model to communicate****.* Source code is the end result that we all are trying to produce -it is not a model which will be delivered as an executable after all.

However, it is not the best way to explain your ideas to one or more other people. Within XP, if a new piece of code is to be implemented, a programmer may call a quick “stand-up” meeting to run through their ideas with their pair-programming partner.

At such a meeting, models may well be the best form of communication. Of course, this does not necessarily imply the use of a modelling tool. It may involve white boards, post-it notes, index cards, etc. But then, that is exactly what Agile Modelling encourages you to use

* **Multiple Models**

If you wish to fully understand your application, then you may well need to consider a class diagram (for the system structure), a sequence diagram (for its behavior), a data diagram (for the database representation), etc.

This is still as true for XP as it is for Agile Modelling! You should then switch between the different models as and when required

* **Know Your Models**

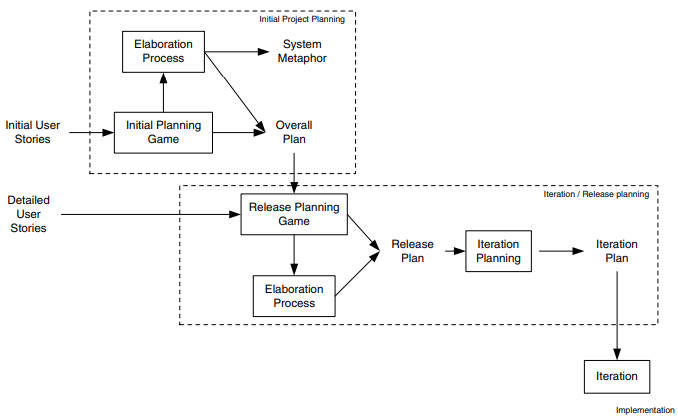
Whether you are an Agile modeler or an XP developer you need to know the tools available to you.

Even for those working on an XP project, modelling is relevant (as has already been said). Thus, XP developers need to understand the models available to them just as much as an agile modeler should.

That is, they should understand the strengths and weaknesses of different types of models.

This will help them to keep models as simple as possible, as well as helps to apply appropriate models which will help them to understand the systems under consideration.

* **XP objections to agile modelling**
* Arguments raised by XP practitioners against Agile Modelling include
* **Modelling is all about big up-front design** Agile Modelling clearly does not promote this. This is illustrated by the Agile Modelling practices such as “Model in Small Increments” and “Prove it with Code.”
* **All models are permanent documents** that must be updated when any changes are made. This is clearly not what Agile Modelling says. For example, the practices “Discard temporary models,” “Use the simplest tools” and "Update only when it hurts” contradict this view.
* **You need to use a complex modelling tool**, such as Rational Rose to carry out any modelling activity. However, as Agile Modelling explicitly debunks that myth, stating you should use whatever modelling medium is appropriate, which may include modelling tools such as Rational Rose, but also white boards, index cards, post-it notes, etc.
* **You need to know, and use, UML to create models**. Agile Modelling does say that you should know how to apply whatever representation you are using in your models and that UML is one example of this. But it does not mandate any particular type of representation and indeed Agile modelers know that something like UML does not cover all the modelling situations you might want. In addition, an Agile modeler will not worry about creating a precise and complete UML diagram. Instead, they will focus on the audience of what they are creating and make sure that it is comprehensible to that audience.
* **XP does not encourage modelling**. Actually this is wrong. XP does promote the creation and use of models. The use of index cards for user stories and classes is a form of modelling. XP practitioners will also often draw diagrams on white boards while trying to consider how to address a problem of refactor code, etc. These are again forms of models.
* **XP does not create any documentation** and models are a form of unnecessary documentation. XP promotes code as the core form of documentation for a system as only code is in sync with code. However, documentation needs to be appropriate for the reader of that documentation. Source code may be a good source of reference for programmers, but it is unlikely to be appropriate for end users, non-programmers, support personnel, etc. In some cases, models may be a very useful form of documentation for some of these audiences. For example, a UML deployment diagram may be very useful for understanding how the system will be installed over a network.
* **Agile modelling and planning XP projects**
* Here we will consider how and where Agile Modelling fits into the project-planning aspects of an XP project
* An XP project is planned at a number of levels and at various points during an XP projects lifetime
* This means that Agile Modelling practices may be more or less relevant at different stages during this process.
* In the following, we will consider the planning process and where Agile Modelling can be exploited to the benefit of XP





* **Initial Project Planning**
* There are two primary steps within the initial project-planning phase; these are the initial planning game and the elaboration process.
* *Initial planning game*. During this process, business and development may resort to modelling to help them clarify the user stories. By applying Agile Modelling practises, this modelling can be controlled and focused. An example of where they might do this is when a User Interface mock up might be created, with some simple flowcharts to prototype system behaviour as a way of elaborating a user story. For example, Figure illustrates a possible user interface design for a membership web site. This is a diagram drawn on a white board to consider what fields are needed and what will happen when a user selects the submit option. The flow diagram presented in Figure expresses what happens when the submit button is pressed. Again, this is a diagram drawn on a white board.
* *Elaboration process*. During the elaboration process, various models maybe created to help the developers understand what will be required of the system. This will help to produce better estimates, etc. Again Agile Modelling practises can be of great help here.
* **Iteration/Release Planning**
* During the iteration/release planning stages, modelling is again important.
  + ***Release planning game***. As with the initial planning game, Agile Modelling practices can help focus the modelling activities used to clarify user requirements.
  + ***Elaboration process***. Although this is typically a shorter process than for the initial project planning phase, some modelling often still takes place and Agile Modelling can be applied to ensure that modelling does not become a burden.
  + ***Iteration planning***. In order to break down user stories into tasks, it may be necessary to model how the user stories might be implemented. This might involve initial class structures, behavior, etc. This can allow tasks to be identified, clarified or split up. Note that this is not large up-front design, as the models may be discarded and may only be intended to help elaborate the tasks.
* **XP implementation phase**

* This is where the code actually gets written within an XP project. There are therefore various points at which a model may be relevant and therefore Agile Modelling practises may be applied.
* For example, in helping to understand code in order to refactor it, etc. We will now look at how Agile Modelling can complement several of the implementation-oriented practises of XP.
* Implementation of oriented practises, mean practises such as “Test-first coding,” and “Refactoring, “rather than the more process-oriented ones such as, “The Planning Game” or the“40 hour week rule.”
* The practises to be looked at in this section are:
  + refactoring,
  + test-first coding,
  + simple design and
  + Pair programming.
* **Refactoring**
* Refactoring is primarily a code improvement technique, so is it compatible with a modelling activity? Is modelling and Agile Modelling in particular, relevant to refactoring? The answer of course is “yes,” as we have already indicated earlier in this chapter. In the last chapter, some of the issues to consider when refactoring were given as:
  + Make sure you know how to improve the code.
  + Make sure what you have done has improved the code.
* That is, “know what you are doing!” It was also stressed that you should not refactor: “When you haven’t got a clear plan of how you will improve the code.”
* Agile Modelling can be used to help with all three issues. By modelling various aspects of the system, you may gain a better understanding of what it does.
* By modifying the model, you can evaluate how you might refactor it and whether it appears to have benefited from the refactoring.
* This is a lot cheaper than actually coding the changes and then considering the results. It may also be a better medium through which to convey your ideas to your pair-programming partner or to others; thus allowing improved communication of ideas.
* By applying Agile Modelling practises, you can determine if they do actually need to be revised or not. For example, by considering the following two Agile Modelling practises:
  + Update only when it hurts not to do so.
  + Discard temporary models.
* You might decide not to update the existing models until absolutely necessary. Even then, I find it better to wait until someone shouts for the models and update them at that point (in Just-In-Time fashion) as an XP project may refactor the code several times before the point at which someone wants to reference the documentation. This would lead to unnecessary model revisions taking place.
* **Test-First Coding**
* At first sight, with respect to test-first coding, modelling may seem at best superfluous and at worst contradictory. This is because, in test-first coding, you essentially follow this cycle:

1. Write a test.

2. Write the code to be tested.

3. Run the test/get the code to work.

4. If the test has passed, then return to step 1 until finished.

* So where does Agile Modelling fit into this cycle? It may fit in at a couple of points. You may have decided to carry out a small amount of modelling at the start of the current task in order to understand what you need to do.
* If you have taken into account the “Design for Testability” Agile Modelling practise, then it may have helped to identify the tests to be implemented. Another point at which Agile Modelling may be relevant is once a test has been written and you need to consider how to implement the business code.
* A short “stand up” design session with your pair-programming partner can be invaluable. It can help to clarify any number of issues and help to make sure that they (and you) are engaged in what is being done and that no one is just along for the ride. This also fits with the Agile Modelling practise “Prove it with Code.”
* Thus, if you modify the test-first coding cycle to the following, then you are maximizing this XP practice, as well as, supporting the Agile Modelling principle of rapid feedback:

1. Write a test.

2. Model the solution.

3. Implement the solution.

4. Run the test/get the code to work.

5. Discard temporary models.

6. If the test has passed, then return to step 1 until finished.

* **4 rules of Thumb**
* Given all of what has been said above, what can we say about where XP and Agile Modelling should be applied? There are actually a number of rules of thumb that can be identified. Note that these are rules of thumb and are not hard and fast rules. For example, XP can be and has been applied successfully to larger projects. However, it is more difficult and there is a greater potential for failure, particularly for those inexperienced in agile approaches. The rules of thumb that have emerged include:

1. Smaller projects of typically less than 10 people. The larger the project, the harder it is to manage as a pure XP project. The XP approach, that is to a large extent self- organising, becomes difficult with 20 or 30 developers involved. Of course, the team can be broken down into smaller groups and treated as smaller XP projects, but then that’s what you have, smaller XP projects interacting.
2. Known domain/applications. For larger projects, XP projects work best where the domain and the type of application are well understood.
3. Well-established architectures. This point is related to rule 2 in that the reason that XP works well in well-understood applications is that there is a (possibly) implicit architecture. The developers know what they should do where, when and how. If this is not the case, then an architecture needs to be established within which the XP project can operate.
4. Scalability not an issue. If scalability is an issue, it must be considered early on in the project so that it does not become an issue later on. This is typically a problem in larger, longer-lived projects where it is difficult to see the scalability issues early on. Again, an architecture may help.