What is meant by software Process Model?

Background:

> AVIANA is a software company. It has a team of 25 programmers. A famous neuro surgeon of Karachi has recently established his small hospital in Karachi and asked AVIANA Tech to develop a management system for his hospital. The request was to develop a system to electronically manage patient records and perform administrative functions similar to 'what is being used in other such hospitals but within a budget of Rs. 70,000 and delivered within 3 months.

If AVIANA adopts an ad hoc approach to software development and has not worked on large scale projects before. What problems do you anticipate in this project?

Problems:

- **Difficult to distinguish between tasks**
- **Important tasks may be ignored**
- **Inconsistent schedules, budgets and functionality**
- **Erroneous or Poor product quality**
- **Delayed problem discovery**
- **Errors are costly to fix**

What is a Software Process Model?

An abstract representation of a process. It presents a description of a process from some particular perspective.

Software Process Models provide guidelines to organize how software process activities should be performed and in what order.

A process Models defines the following things:

1. The set of tasks that need to be performed
2. The input and output from each task
3. The pre-conditions and post-conditions from each task.
4. The sequence and flow of each task

A Simple Process Model:

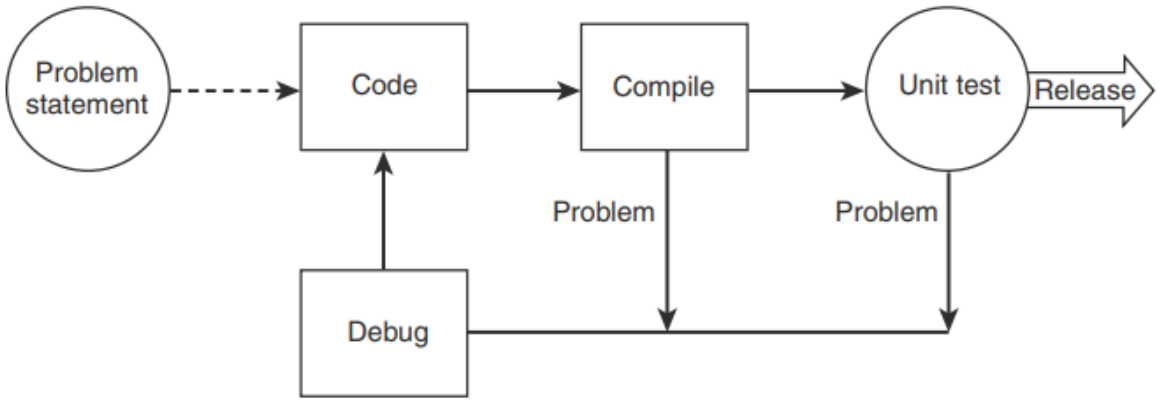


Figure: A Simple Code & Fix Approach

Problems:

- There is no way to estimate time-scales or budgets.
- There is no assessment of possible risks and design flaws: you may come close to a finished product only to find an insurmountable technical problem which sets the whole project back.

We only mention the code-and-fix approach in the context of life cycle models since it is a base-line model which we should avoid. From a software engineer's point of view, code-and-fix is a worst case scenario.

PRESCRIPTIVE SOFTWARE PROCESS MODELS:

Prescriptive process model were originally proposed to bring order to the chaos of software development. Prescriptive process model define a prescribed set of process elements and a predictable process work flow. "prescriptive" because they prescribe a set of process elements framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project.

1. The Waterfall Model

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- In this model requirements must be specified in the first step.
- This model is used for the small projects.
- The output from each step is fed into the next step as input
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

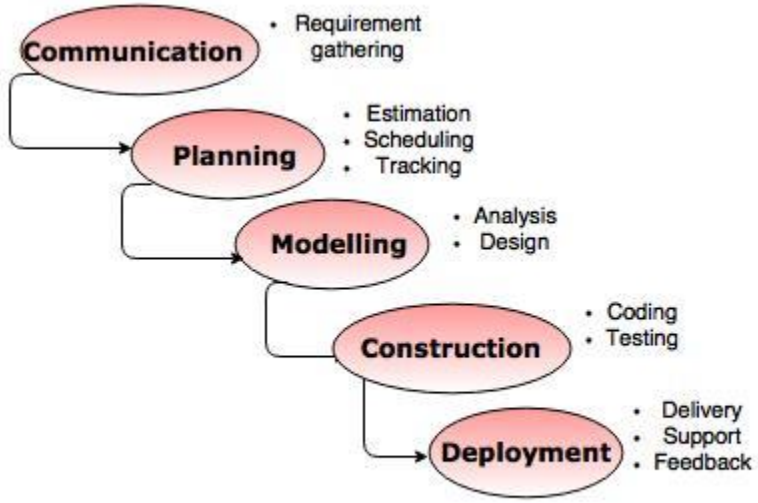


Fig. - The Waterfall model

An alternative design for 'linear sequential model' is as follows:

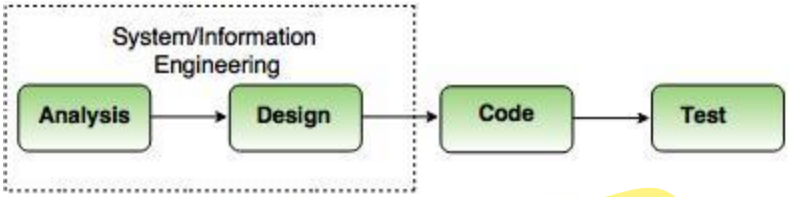


Fig. - The linear sequential model

Advantages of waterfall model

- The waterfall model is simple and easy to understand, implement, and use.
- The stages consist of well-defined tasks which promotes good scheduling and cost estimation (if all stages occur in the expected sequence once only). • The deliverables provide targets or milestones to see how far a team has reached in the development process.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

2. Incremental Process model

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.
  For example, the word-processing software is developed using the incremental model.

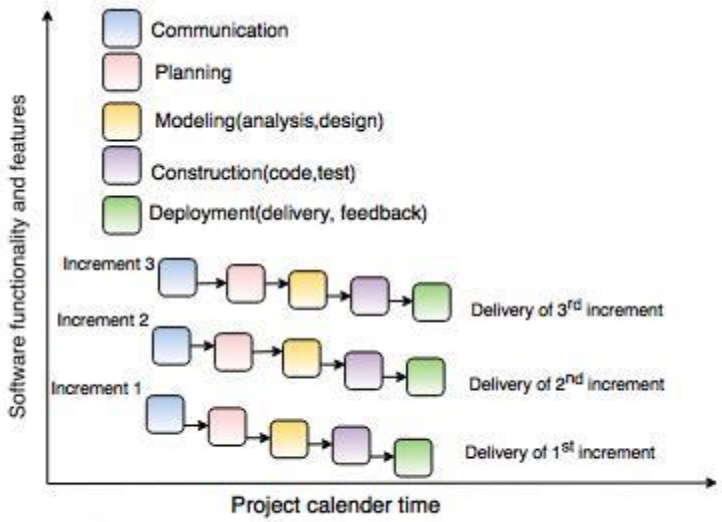


Fig. - Incremental Process Model

Advantages of incremental model

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The process is more responsive to changing user requirements than a waterfall approach — later subsystems can be re-specified. Also a modular approach can mean maintenance changes are simpler and less expensive.
- There is an opportunity for incremental delivery to users, so the users can benefit from parts of the system development without having to wait for the entire life cycle to run its course.
- Incremental delivery means that users have a portion of the software to examine in order to see how well the software meets their needs, and whether the software requirements have to be modified.
- Complete project failure is less likely, since users will have some working sub-systems even if time and money run out before the complete system is delivered.
- The project can begin with fewer workers, as only a subset of the final product is being worked on.

- The risk associated with the development of the software can be better managed.
- The time taken to develop previous iterations can be used as an estimate for the time needed to develop the remaining iterations, and hence improve project planning.
- The customers can respond to its functionalities after every increment.
- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks.

Disadvantages of the incremental model

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- This development model relies on close interaction with the users — if they are not easily available or slow in evaluating each iteration, the whole process can slow down.
- The reliance on user involvement can worsen the already difficult task of estimating the amount of time and budget required.
- High user involvement means that resources are drawn away from the customer's normal operation during system development.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

3. RAD model

- RAD is a Rapid Application Development model.
- Development cycles are rapid, typically between 60 to 90 days
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The RAD model consist of following phases:

1. Business Modeling

- Business modeling consist of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

2. Data modeling

- The information in the business modeling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

3. Process modeling

- The data objects defined in the data modeling phase are changed to fulfil the information flow to implement the business model.
- The process description is created for adding, modifying, deleting or retrieving a data object.

4. Application generation

- In the application generation phase, the actual system is built.
- To construct the software the automated tools are used.

5. Testing and turnover

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
- The data flow and the interfaces between all the components are are fully tested. Hence, most of the programming components are already tested.

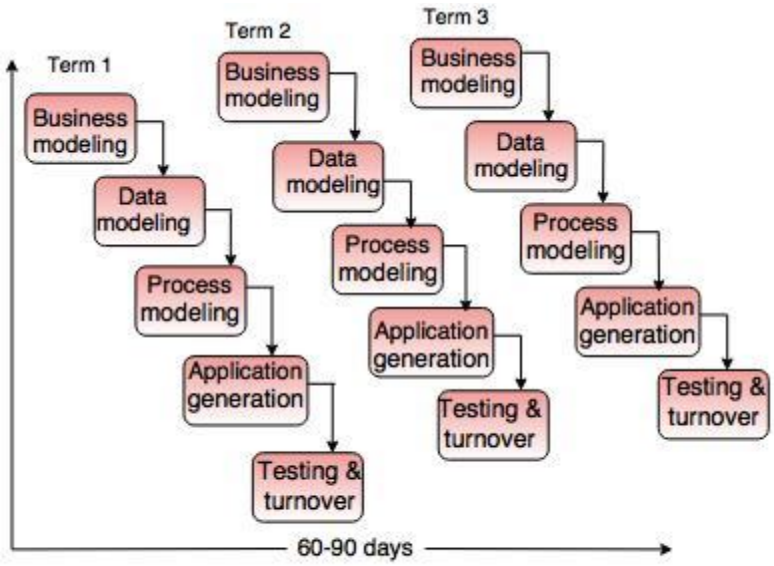


Fig. - RAD Model

Disadvantages of the RAD model

- For large projects, RAD may require a large number of people to split the project into a sufficient number of teams.
- The developers and the customers must be committed to the necessary activities in order for the process to succeed.
- The project must be suitably modularised in order for RAD to be successful.
- RAD my not be appropriate where high-performance is necessary.
- RAD may also not be appropriate when technical risks are high.

Evolutionary process models

Product requirements may change with time, even while the software is under development. Worse, the initial specifications may not be detailed, and tight deadlines may result in a need to have software quickly ready.

All of this points to a product that evolves over time, and evolutionary process models are designed to satisfy the engineering requirements of these products. Evolutionary process models are, as we shall see, iterative; they allow for the software engineer to deliver a product, and then iteratively move towards a final product as the understanding of the product improves.

1) Prototyping life cycle model:

A prototype system is a smaller version of part(s) of the final system that gives the user a sense of the finished system's functionality.

It has some of the core features of the final system and, where features and functions are omitted, it pretends to behave like the final system.

Prototypes are typically developed quickly, may lack unnecessary features, may be buggy, and have poor usability.

However, prototypes can fill an important role in understanding software which does not have clear requirements.

Where the system to be developed is a truly new system, there may be no clear requirements defining the software's behaviour. By building a prototype, both the developers and users have some real, visible working system model on which to focus their ideas. An analysis of this prototype forms the basis for the requirements specification, and perhaps even some of the design.

Advantages of prototyping include:

• Users get an early idea of the final system features.

• The prototype provides an opportunity to identify problems early and to change the requirements appropriately.

• The prototype is a model that all users and customers should be able to understand and provide feedback on, thus the prototype can be an important tool to improve communication between users and developers.

• It may be possible to use aspects of the prototype specification and design in the final system specification and design, thus some of the prototype development resources can be recouped. A major problem with developing “disposable” prototypes is that the customer may believe it to be the final product. Customers may not understand

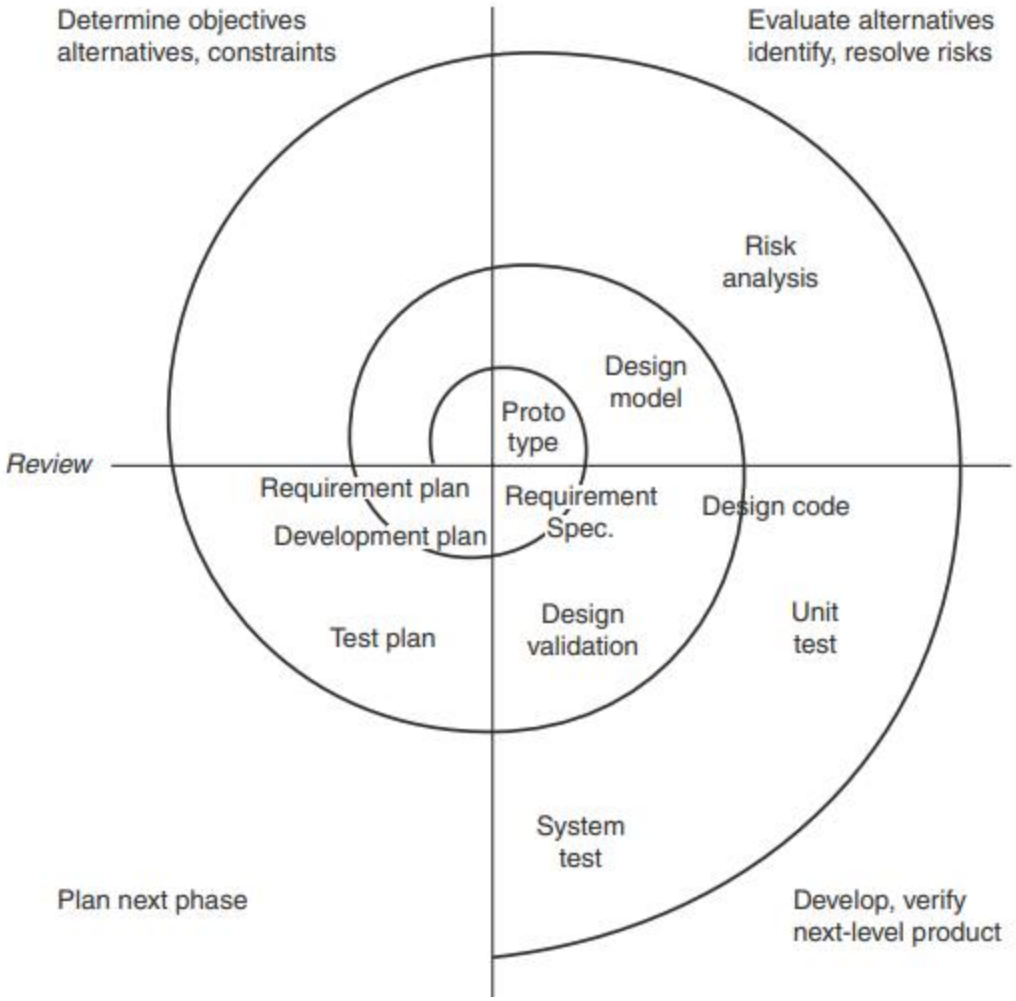
the need to re-engineer the software and restart development, and may ask that the prototype be “cleaned up” and released to them.

Boehm's spiral model

The spiral model was published by Barry Boehm in 1986.
It provides an iterative, evolutionary approach to software development combined with the step-by-step aspects of the waterfall process model and the requirements analysis abilities of prototyping.
It is intended for development of large, complicated software projects.

This process model provides for the rapid development of progressively more complete versions of the software. Each iteration of the evolutionary development will have a release, which may merely be a paper model of the software, a prototype, or an early iteration of the software.

Each iteration of the spiral model contains all of the activities from the generic process framework outlined above: communication, planning, modelling, construction and deployment. One can consider an iteration to be an arc in a spiral: each arc contains the same breakdown of how the development is approached, but each arc will focus on something new.



Each iteration also requires a certain amount of risk assessment, in order to lay out the plans and determine how the project should proceed.

The advantages of this model are:

- The spiral model considers the entire software life-cycle.
- Because of its iterative approach, it is adaptable, and appropriate for large-scale projects.

However, the model does have disadvantages:

- It requires expertise at assessing and managing risk.
- It may be difficult to convince customers that such an evolutionary approach is necessary

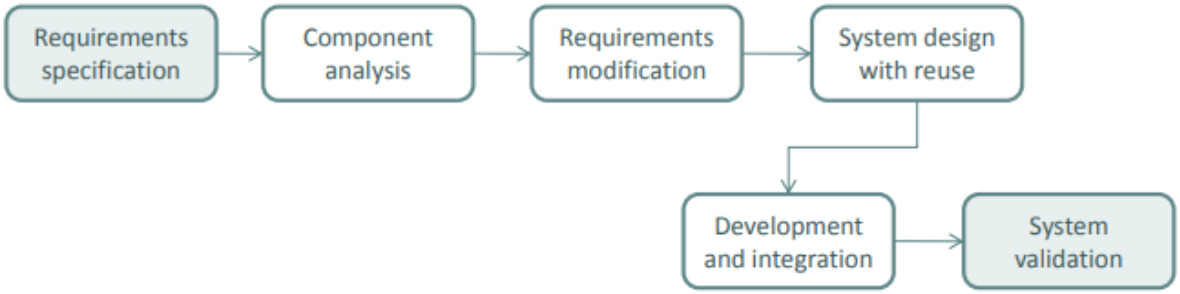
Component-based development

In this process model, software is developed by integrating pre-developed software components and packages. This may be commercial, off-the-shelf components, or they may be components previously developed by the software engineers themselves.

Each component needs to present a well-defined interface to allow for easy integration.

The component-based model proceeds through the following steps:

- Determine what components are available and evaluate them for their suitability. Consider how the component will be integrated with the software.
- Design the software architecture so that the components may be easily employed.
- Integrate the components into the architecture.
- Test the software to ensure that all of the components are functioning appropriately together.



This approach may lead to a strong culture of component reuse. It has been shown that this model also leads to a 70% reduction in development time, an 84% reduction in project cost, and increased developer productivity.

This model is similar to RAD, which we discussed earlier. Note that RAD differs in that it is focused on rapid development, rather than specifically on component reuse.

THE FORMAL METHODS MODEL

The formal methods model focuses producing formal, mathematical specifications of the software product. When the software is built to the given specification its behaviour will already have been

verified to strictly meet the software's specific requirements.

The importance of this model comes from its ability to discover ambiguity, incompleteness, and inconsistency in the software requirements. This stems from the formal, rigorous, mathematical approach employed for software specification.

Formal methods are important to the development of safety-critical software, such as that used in aircraft avionics and medical devices. Formal methods have also been employed in business-critical software, where, for instance, severe economic problems may occur if the software contains errors. Even though this model can produce extremely reliable software, it has many disadvantages:

- The development of the software's formal model is both time consuming and expensive.
- Very few developers have any training in formal methods, and so require extensive training.
- Formal methods cannot easily be used as a means of communicating with the customer and with non-technical team members.

Remember that expert training in formal methods is needed to employ this process model. Formal methods also do not replace traditional methods: they should be used in conjunction with each other. Importantly, employing formal methods does not mean that the software developer need not adequately test the software.

THE UNIFIED PROCESS

- This unified process is also known as the Rational Unified Process (RUP).
- The unified process is a unification of the various early object-oriented analysis and design models proposed in the 80s and 90s. It is an attempt to combine the best features of these various models which initially resulted in the unified modelling language (UML). The UML has become the standard diagrammatic language for modelling object-oriented software.
- The unified process is an incremental software process that is architecture driven, focuses on mitigating risk, and drives development through using use cases. Being architecture-driven, early iterations focus on building the portions of the software that will define the software's overall architecture.

- Focusing on risk, early iterations also focus on developing the high-risk portions of the software. Software development iterations moves through five phases: inception, elaboration, construction, transition and production.

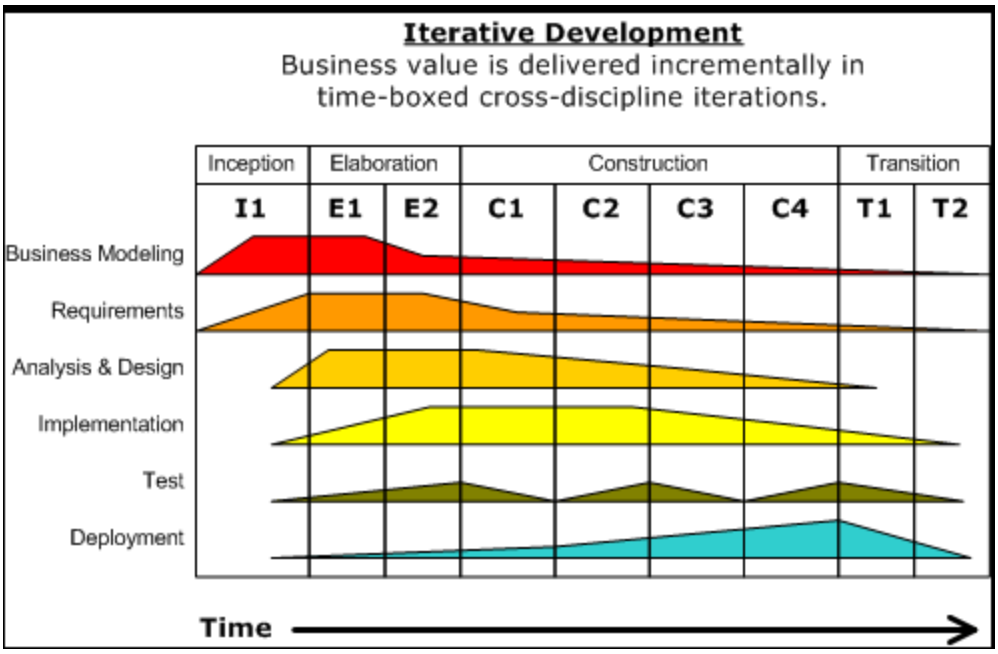
- These phases cannot be directly mapped on to the generic process framework activities: rather, each iteration contains some of the framework activities.

- **The inception phase** is concerned with project feasibility: what should the software do, in broad terms rather than specifics, and what are the high risk areas? Should the development go ahead? Inception is usually a short phase, often having no more than one iteration. Little development usually occurs during the inception phase, but the software requirements are discovered using use cases (communication), and a small subset of these requirements (those with high risk, and which focus on the software architecture) are fleshed out (communication and planning).

- Programming begins during the iterations of the **Elaboration phase**. Each iteration develops the requirements fleshed out in the previous iterations (modelling and construction), and chooses more requirements to flesh out (communication and planning) for development in the next iteration. The elaboration phase completes once all of the requirements have been fleshed out. However, this does not mean that communication and planning activities stop and do not occur in later phases: there is always constant communication with the customer and an understanding that requirements may change.

- Much of the construction activity occurs in the iterations of the **construction phase**. While the iterations of the elaboration phase each had at least one meeting in which some use cases are fleshed-out and selected for development in the next iteration, all the use cases have already been fleshed out when the construction phase begins.

- The **transition phase** contains the initial portions of the deployment activity: the software is given to the customer for evaluation. The customer's feedback will cause the software to be modified as required, and thus the transition phase includes some communication and construction activities. The transition phase involves beta testing.


Iterative Development
Business value is delivered incrementally in
time-boxed cross-discipline iterations.
Inception
Elaboration
Construction
Transition
I1
E1
E2
C1
C2
C3
C4
T1
T2
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Time