Software Engineering

Prof. Umesh Bellur
Department of Computer Science & Engineering
Indian Institute of Technology, Bombay

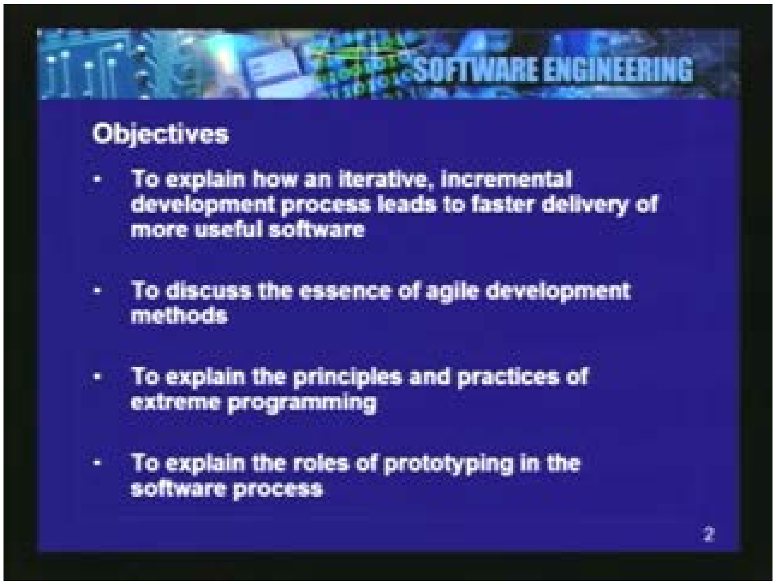
Lecture - 26

Agile Development, XP, Prototyping

That software engineering processes, methods and methodologies which allow you to built software systems in a very streamlined manner but most of these have focused on the building of fairly a large software system and which there is a significant amount of design to be done, there is a significant requirement body of requirement that has been established, a specification has been written and so on and there is a typical process that you end up following to deliver systems of that kind. But there may also be situations in which businesses need the turnaround time of software development to be very very small. In other words, I need the product of the requirement at the end of specifying very very quickly because they are responding to a certain business need in turn and this necessitates a different model to be followed in the production of software and that is what agile development is all about.

It is about trying to turnaround requirements very very quickly may be in an incremental mode; in fact most often in incremental mode so that there is some output at the end of a very short period and the rest of the output can follow behind that and we will take a look today at some of the processes and methodologies that we can use to turn software around very very fast. So the objectives of this part of this course is to basically explain how an iterative incremental process is very very useful to deliver software quickly and what is the what are the aspects of such a process you know what are the considerations to be followed in such a process to explain the principles of a particular programming methodology called extreme programming and we will go into a little bit of detail about extreme programming which is one of the latest and one of the newest practices in software development and to explain the roles of what is called prototyping in this software development process.

(Refer Slide Time: 02:21 min)



SOFTWARE ENGINEERING

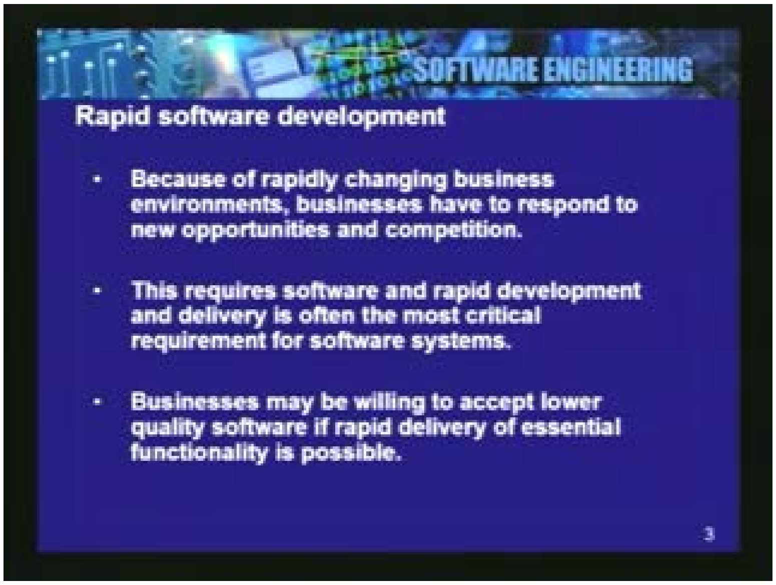
Objectives

- To explain how an iterative, incremental development process leads to faster delivery of more useful software
- To discuss the essence of agile development methods
- To explain the principles and practices of extreme programming
- To explain the roles of prototyping in the software process

2

So why is software development rapid software development required and why is there a need to turn something around very very quickly whereas traditionally software was constructed the way that other things are constructed.

(Refer Slide Time: 03:19 min)



SOFTWARE ENGINEERING

Rapid software development

- Because of rapidly changing business environments, businesses have to respond to new opportunities and competition.
- This requires software and rapid development and delivery is often the most critical requirement for software systems.
- Businesses may be willing to accept lower quality software if rapid delivery of essential functionality is possible.

3

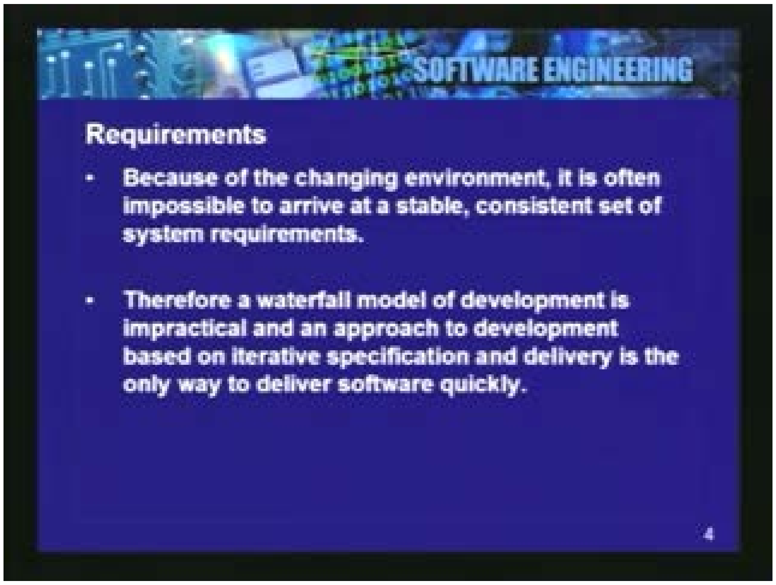
For example, example if you are building a dam or if you are building a bridge across a river there is no short cut method to be followed, you know, the bridge or the dam has to be designed, measurements have to be taken or the design has to be very carefully validated to make sure that it is going to withstand the weight, the speed of the vehicles that are moving over it and so on before construction has begun and there is no piecemeal construction of the structure either the structure is there and usable or it is not at all it is just a binary kind of a decision there.

But in the case of software it need not be the case by the very nature of the name because there could be very rapidly changing business conditions that lead the business to make certain decisions so they have to respond so maybe there is a new opportunity round the corner that they have to respond to maybe it is a new service that they wish to introduce in say the telecommunications domain or there may competitors who are coming up with certain products that the business has to respond to in which case it may have to make changes to its own products.

So all of these requires a software development where the delivery is very very critical and the delivery has to happen very quickly and so that requirement comes about and it is obviously often true that there has to be some compromise that is going to be made between the time that it takes to deliver a product and the quality of the product at the end of the day as long as there is some software available to be rolled out that shows the progress that shows the progress has been made and the rest of it can come along in a sequential manner then it is okay so there may be businesses that accept a slightly lower standard of quality in a shortest time cycle and quality can be improved as things go along so may be non-functional requirements may not be perfect, it may not be able to carry the same amount of load for example, the software but as time goes along we can improve the performance, we can improve the reliability, we can improve the availability of the software as well.

So, because of the changing nature of the environment and because of the rapid changes that are taking place in the business environment it may not be possible always to arrive at a system requirements phase which can be called complete in any sense of the term before you move on to system specification, before you move on to functional specification design, development and testing and so on.

(Refer Slide Time: 05:32 min)



SOFTWARE ENGINEERING

Requirements

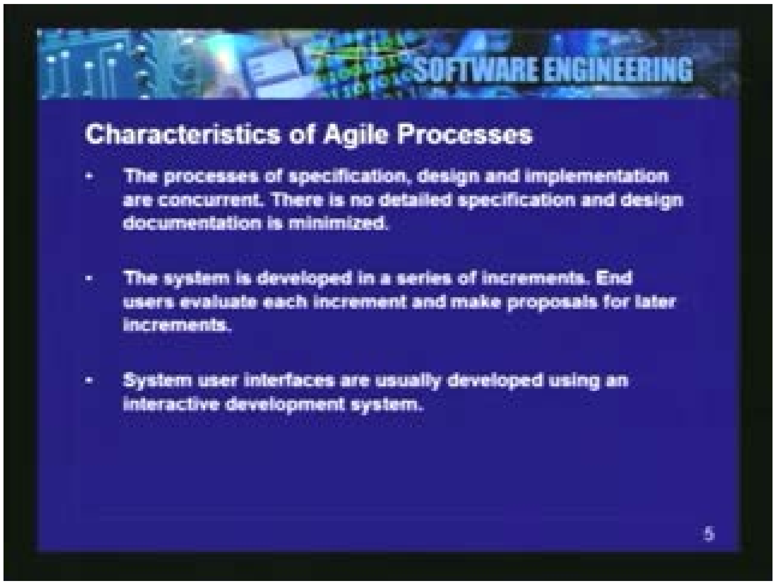
- Because of the changing environment, it is often impossible to arrive at a stable, consistent set of system requirements.
- Therefore a waterfall model of development is impractical and an approach to development based on iterative specification and delivery is the only way to deliver software quickly.

4

So if the requirement themselves keep on changing and therefore the traditional waterfall models of development that we have seen so far is not very easy to follow that because there there is a presumption that the requirements phase is complete before you move on to the design phase or the specification phase of the software and that may not be possible here because requirements are coming in incrementally. So the details of the new service may all be known, the new server is going to be introduced and here the essential characteristics of the service but the details of the characteristics may not be known and that may come in as the project progresses and therefore iterative delivery of software is the only way to handle changing requirements like this.

So what are the characteristics and what are the fundamental of the essential nature of the agile process of development; agile process basically refers to delivering software quickly.

(Refer Slide Time: 6:31)



SOFTWARE ENGINEERING

Characteristics of Agile Processes

- The processes of specification, design and implementation are concurrent. There is no detailed specification and design documentation is minimized.
- The system is developed in a series of increments. End users evaluate each increment and make proposals for later increments.
- System user interfaces are usually developed using an interactive development system.

5

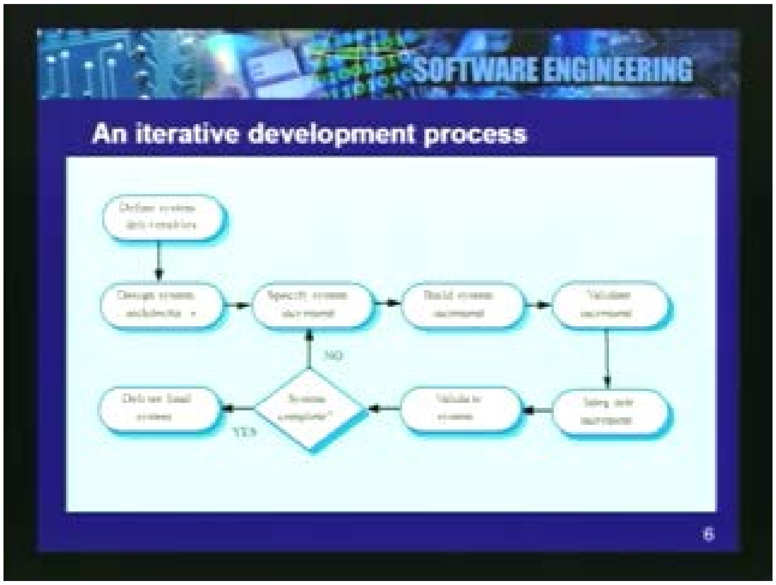
So one of the characteristics is that we just noticed that the requirements phase cannot be complete before we move on to the other phases which means implicitly that these phases are concurrent in nature so that the requirement specification phase, the design phase, the development phase and the test phase often may be concurrent with one another so also there is no detailed specification typically written out for this. The design documentation can be very minimal because you are moving very very quickly in order to get the software out. So the characteristics are that because of the incomplete nature of the requirement, because of the change in requirement and because of the concurrency of the different phases the system is usually delivered in increment; it is delivered in very very small increments and every increment is completely enough and itself has to be integrated back into the main system as the project progresses.

And another very key feature is that because it is delivered in increment that is the key word here the users get to see sample and you know touch and feel the software very early on the lifecycle so that they can give feedback, they can give suggestions on the improvement in the existing software, they can also tune the further requirements that they are going to give in a way that is going to make it easy for you to construct the rest of the software that is one thing.

The third and the last characteristic here is that user interfaces are typically developed very early on here to get user feedback and they can be developed using an interactive paradigm. So, for example there could be something which will allow you to change the positions of the buttons, which are allow you to change the positions of the fonts, the text features that you have to enter, the kinds of inputs that you have to give and all of these in fact be changed on the fly as the user is experiencing using the software and these changes can be made permanent going forward.

So an agile process is really focusing on what is called a rapid application development environment and the things that go along with that. So what is an iterative development process? Let us take a look at what the process itself looks like.

(Refer Slide Time: 8:43)



SOFTWARE ENGINEERING

An iterative development process

Define system deliverables

Design system architecture

Specify system increment

Build system increment

Validate increment

Integrate increment

Validate system

System complete?

NO

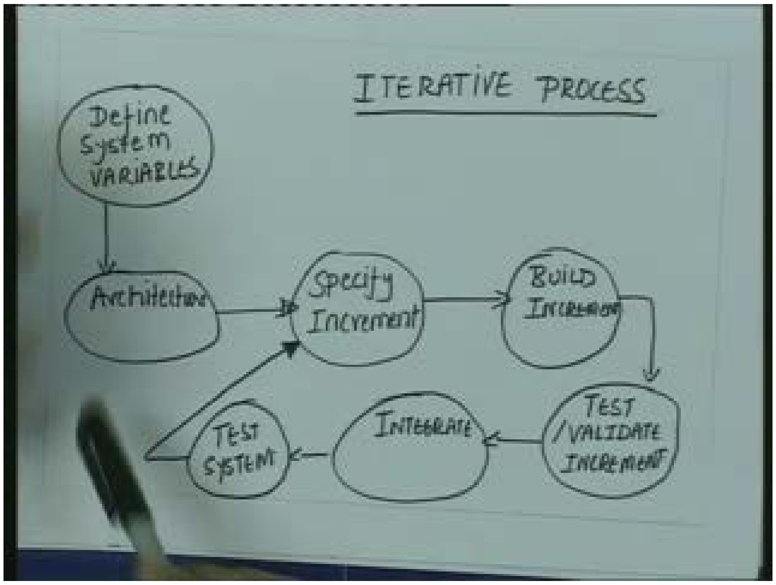
YES

Deliver final system

6

We have drawn out the diagram here where you start out by defining the different system variables that can exist.

(Refer Slide Time: 8:50)

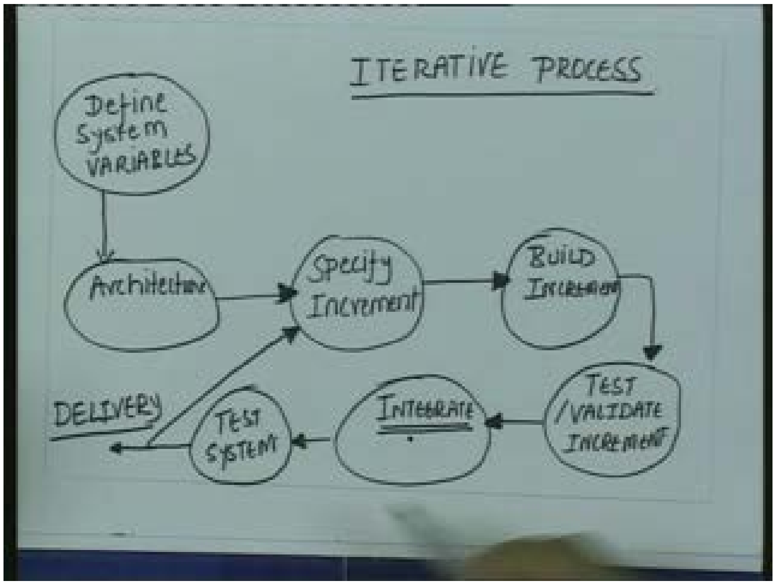


Now, once the architecture, the system variables lead to a decision on what kind of architecture is to be employed, so, for example, a pipes and filters architecture can be employed, a three tier architecture can be employed, a rule based architecture can be employed and so on. Once the architecture decision is made then the requirement that exist after this point in time is specified in a series of increment. So, for that increment that you are going to take a development on you specify what it is going to contain and once that increment has been specified you go ahead and build the increment just like you do in a normal software development process.

Now what happens typically is once you finish development of the software you test it so that step is exactly the same except that you are doing it for a very limited subsection of the requirement which is what exist within the increment and once the test and validation phase is complete the phase that exist here (Refer Slide Time: 9:52) which is quite important and which is happening very often is an integration phase that may exist only once during the lifecycle of a typical software development process which is non-agile in nature. Many of these can exist because this entire thing all the way from specifying the increment, all the way to testing the system can be a loop because you specify many increments.

Thus, once the integration phase is done we then have to test the entire system. what we tested back here in the increment testing phase is your testing the code that was written during this particular increment; you are testing the features that were supposed to be built during this increment but then once you integrate with the rest of the software that exists you have to test the system as a whole and once the system as a whole is tested then there is a question to be asked; now is the system complete or is the new increment to be specified. If the system is complete then you move out of here to delivery of the system (Refer Slide Time: 10:52); if the system is not complete then you go back to specify the next increment within the process.

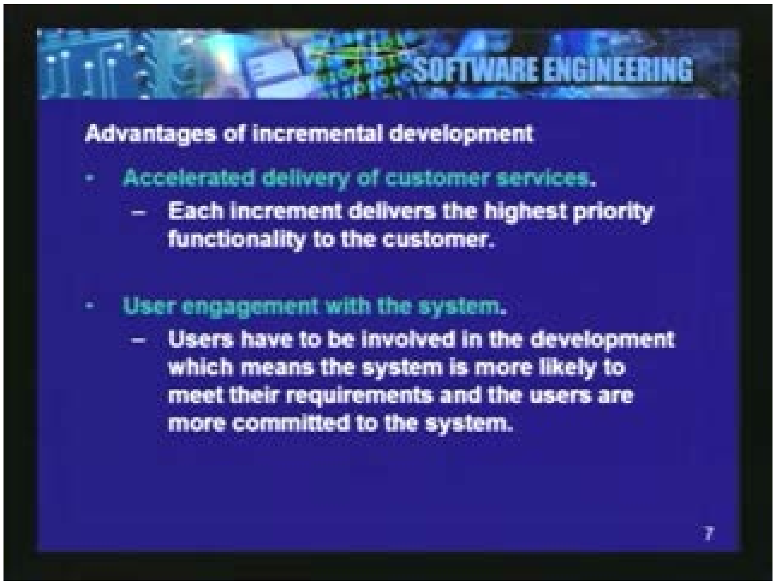
(Refer Slide Time: 11:02 min)



So an iterative process is basically very similar to the spiral model of development that you have seen earlier during lifecycle models and even there the focus is on trying to iterate over small amounts of requirement and go to the complete software development lifecycle with that small sort of requirements, come back, get user feedback and then start work on the additional set of requirements that may exist and the iterative process is very very similar except that the frequency with which this cycle occurs or repeats will be quite high compared to the spiral development models. So there are many more increments; in fact an increment would be something as small as a single single small set of requirements which is called a story which we will see as we go down to the rest of this presentation.

So what it is that is so good about the iterative development process and that is something that we would like to look at next. What are the advantages of such a process and what are the disadvantages of such a process.

(Refer Slide Time: 12:08 min)



SOFTWARE ENGINEERING

Advantages of incremental development

- Accelerated delivery of customer services.
  - Each increment delivers the highest priority functionality to the customer.
- User engagement with the system.
  - Users have to be involved in the development which means the system is more likely to meet their requirements and the users are more committed to the system.

7

The obvious advantage is that there is an accelerated delivery to the customer. So the customer is going to get something in their hands pretty quickly; it may not be the whole system but it is something that they can use to give feedback to the software development team and they can also see for themselves if this is the system that they want to develop; it is progressing in a way that they believe is right for their needs. So every increment essentially delivers the feature set that is the highest priority of the customer. So the customer evaluates that and gives immediate feedback and that itself is another advantage; the user is always engaged with the system so it is not possible.

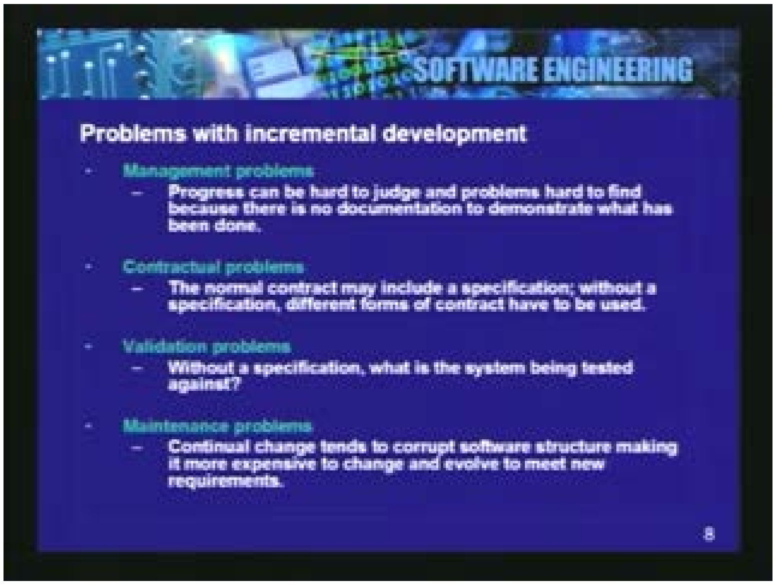
For example, in this process for you to go off into a corner, do development of a large system completely then come back to the user and then the user figures out oops! this is not what I wanted at all, this is entirely wrong; I think I will have to restart specifying the system; that cannot happen in this process because they are going to see something very quickly; in fact they are not just going to see sketches of user interfaces but they will see an entire system which they can actually work with, which they can actually experience with, which they can play with and then they give feedback to the users; I mean feedback back to the software development team. So the users are very committed in this entire process and that is a very very big advantage of the process.

Also, it is very likely that as the user sees the system as it is being delivered to them they might change the requirement so that eventually the end product that is delivered to the customer exactly meets the needs that they do have. On the other hand, the system also presents certain disadvantages.

Disadvantages are there can be serious management problems with this process, progress can be very very hard to judge because of the levels of increment. So, suppose there are medium size systems, it has twenty to twenty five increments that have to be done sometimes increments are as short as one to two weeks in nature so a system that originally took may be a year or two years to build would have like close to hundred increments and to keep track of all these increments and whether progress is eventually being made towards the overall goal of delivering the software product to the customer is a hard thing to do because the customer is allowed to change the requirements on a per increment basis that is one thing.

Also, the focus is not on documentation and as long as there are no design documents it may start getting hard to judge exactly what has been done up to date and what is the amount of work that is remaining to be done. There can be contractual problems that lead out of this process the same thing and different forms of contracts may have to be used because a typical contract would say here is the specification of a system to be built and this is what it is going to cost in terms of time and money to build these many numbers and people who are going to be use the resources that are going to be required and so on. But that process is not as clean in the case of agile development. Here it says you know for one increment it is pretty clear what you are going to do; it is a set of well-specified requirements as far as one increment is concerned. But it is hard to keep signing contracts on per increment basis; the contract has to be for the period of development of the entire software project and it is hard to do that here.

(Refer Slide Time: 15:36 min)


SOFTWARE ENGINEERING

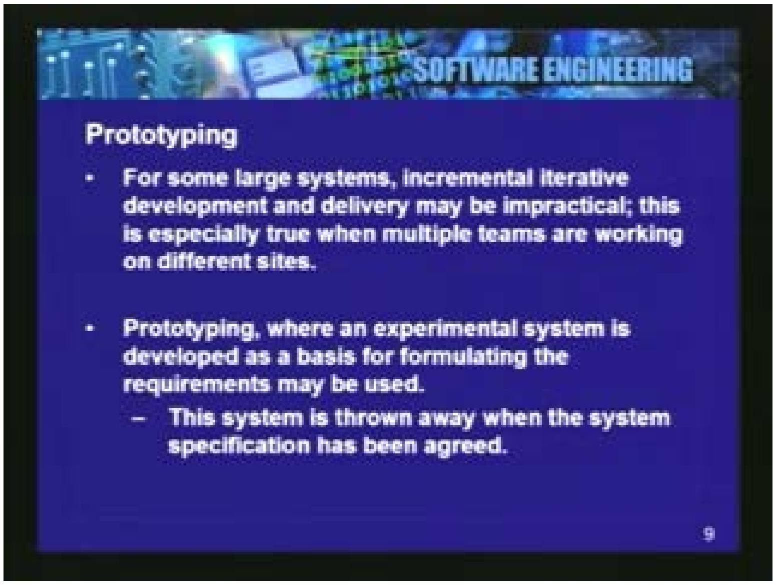
Problems with incremental development

- Management problems
  - Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.
- Contractual problems
  - The normal contract may include a specification; without a specification, different forms of contract have to be used.
- Validation problems
  - Without a specification, what is the system being tested against?
- Maintenance problems
  - Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.

8

Again, validation problems can also exist. Since there is no real specification the specification keeps changing. there is no single body of specification in which validation can be done as a result of which validating whether you are on the right track can be a little hard. This is mainly done informally because of the user feedback and that is actually a plus that works very well but there is no formal method of validation, verification that can be applied in this process and certainly the complexity of the software tends to grow over a period of time because of all the changes that are being made to it constantly; during every iteration you may end up adding to the code, you may end up modifying existing code because of the additional requirement that you are taking and so on as the result of which designs may not be very clean and the complexity of the software may grow to a point where maintainability becomes a real issue and that is something that you have to watch out for in this case. Now very similar to the lines of what is called iterative development is another process called prototyping.

(Refer Slide Time: 16:41 min)



SOFTWARE ENGINEERING

Prototyping

- For some large systems, incremental iterative development and delivery may be impractical; this is especially true when multiple teams are working on different sites.
- Prototyping, where an experimental system is developed as a basis for formulating the requirements may be used.
  - This system is thrown away when the system specification has been agreed.

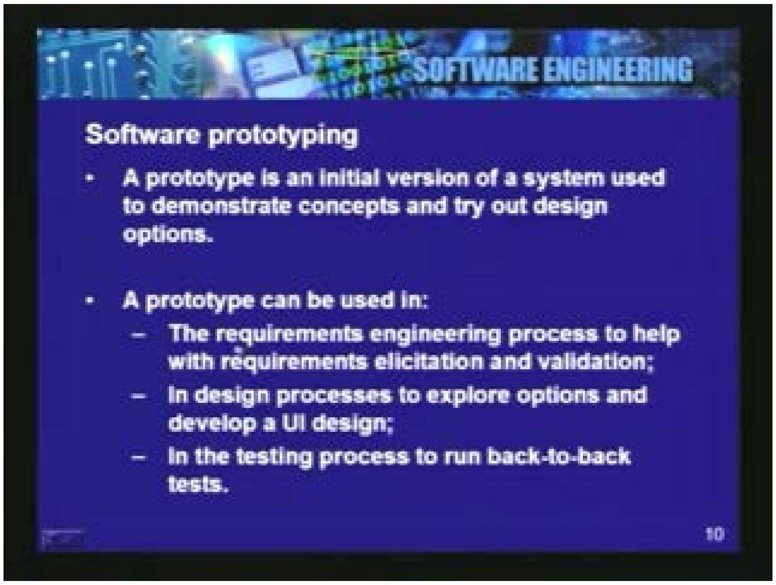
9

Software prototyping: you probably heard of this during the earlier versions in this course and prototyping is very similar to incremental development it does not really focus on delivery to a customer. What it essentially is is a technique for the software development team to satisfy themselves that they have made some progress in the right direction here. It can also be used to get feedback from the user but its prototype is not an entity that is eventually going to be given to the user for them to keep.

Also, a prototype could be something that is purely front end sketch of the system for example, it may just contain dummy user interfaces that do not actually end up doing anything in terms of business logic for example, may not be coded in the case of prototype. So, from very large systems iterative development can prove to be a hindrance as we have seen because of some other disadvantages and because of the impracticality of using iterative development we might need to resort to some other means and so prototyping is the best way of doing that.

Prototyping is basically a process where experimental systems are developed; it may or may not be used; typically prototypes are thrown away before real system development starts and they may be used just validate certain proof of concepts, certain design ideas that are going to be used in eventual software. So a prototype is just an initial version of a system by definition and it is used to demonstrate concepts as we just said and it can be used in several different situations; it can be used, for example, in the requirements engineering process to help out in the requirements elicitation so what it does here is that the user is trying to specify let us say a user interface need that he has so the system needs to give me a form to enter some data or let us say it is a banking system or a financial system and it needs to give me a form to enter some data; I do not know what the form is going to look like so I cannot really say how much of data I want to collect off for that form. Whereas UI prototype or a user interface prototype can quickly be developed, the form can be shown to the user and the user can make some adjustments based on that. They can also realize whether they are going down in the wrong path and they can completely change the requirement. So, very early on prototypes can be very very useful in trying to ensure that the system requirements are done correctly.

(Refer Slide Time: 18:20 min)



SOFTWARE ENGINEERING

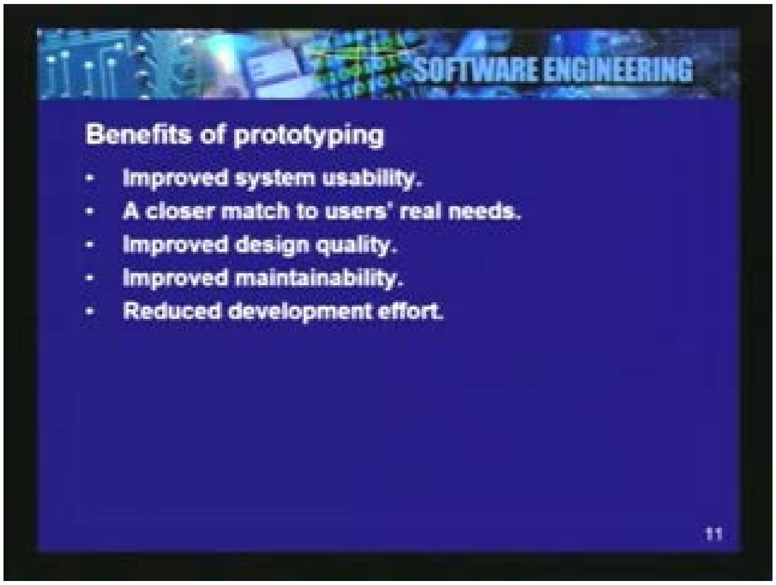
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore options and develop a UI design;
  - In the testing process to run back-to-back tests.

10

Certainly it is also useful in the design process because there could be a proof of concept you are trying to validate. For example, you are trying to use triple desk security in a situation which may or may not require that. Also, triple dusk security may require lot of processing power; the system is supposed to run on a CPU of 1.8 GHz only because that is the requirement from the user so here is a design decision that you have to make as to which security protocol are you going to end up using or you are going to use triple desk security or you are going to use RSA etc etc and this maybe can only be validated with the help of a prototype so you actually put that in there, you do a quick performance study to see whether RSA processing is going to take too much time or not and if not you plug in a different security algorithm into that situation and pick the right one that will fit your need. So, that is an example of using prototyping in the design process and certainly in the testing process, well, to run back to back tests so you can essentially validate some ideas through the prototype you can also run the same test on the real system and then do a comparison to see how these two behave and is the prototype doing something that is acted according to the user and the real systems feeling those steps so there must be some problem.

(Refer Slide Time: 20:37 min)



SOFTWARE ENGINEERING

Benefits of prototyping

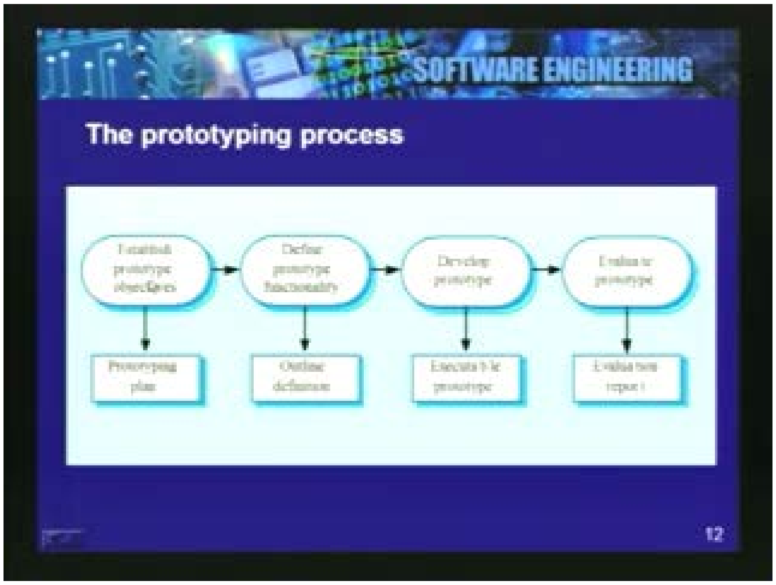
- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

11

So the benefits of prototyping then are improved system usability because of the feedback that is got in from the user very early on. also the requirements are going to be much clearer as a result of which the output or the end product of this entire process matches the user's needs very well; the design quality is highly improved because you have actually had the time to take a look at several choices not just take a look at them on paper but try out the several choices and then pick the one that is right for you. Also, the maintainability is good because this is not code that is reused; all the testing that you have kind of during the design phase of the software engineering lifecycle so you wanted to plug in different algorithms and see which one is the best one and you might incrementally change some of that. now all of this can be thrown away because the prototype can be thrown away and when you start development for the real system you know exactly what is to be done so the code is much better return and therefore much more maintainable.

And overall in fact can reduce in in fact it can actually reduce the development effort or the development work that is to be done, it may even result in the system being delivered faster because the actual coding lifecycle and the testing part of the lifecycle which is very very important which can take very long can get reduced because the quality of the code is much better so those are some of the benefits of prototyping.

(Refer Slide Time: 21:50 min)



SOFTWARE ENGINEERING

The prototyping process

Establish prototype objectives

Define prototype functionality

Develop prototype

Evaluate prototype

Prototyping plan

Outline definition

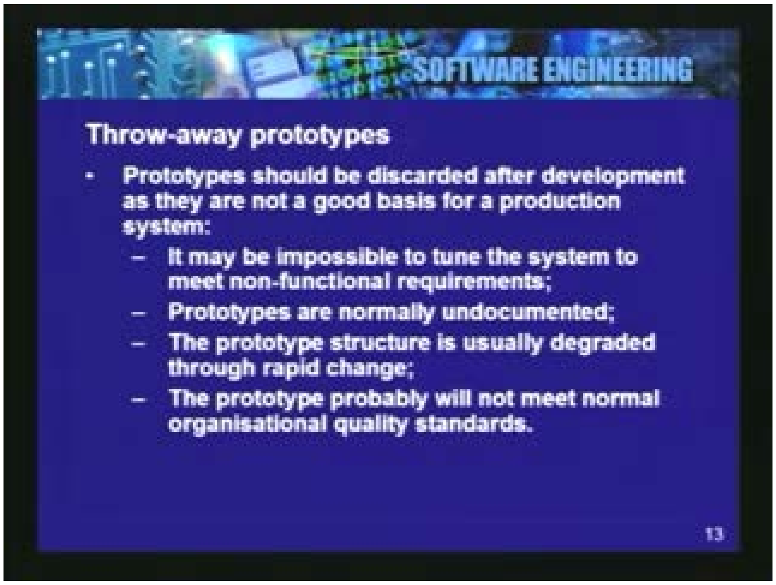
Executable prototype

Evaluation report

12

The process of prototyping is fairly straightforward. So you basically establish the objectives of the prototype as shown here in the diagram and you come up with a prototyping plan because of that, then you define the functionality of the prototype so it is kind of like developing a small module of software very very similar to that and then you develop the prototype, evaluate the prototype and so on. Eventually a prototype may in fact result in a better specification of the system and as we shall see it may not result in any kind of code that can be reused at all.

(Refer Slide Time: 22:36 min)



SOFTWARE ENGINEERING

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organisational quality standards.

13

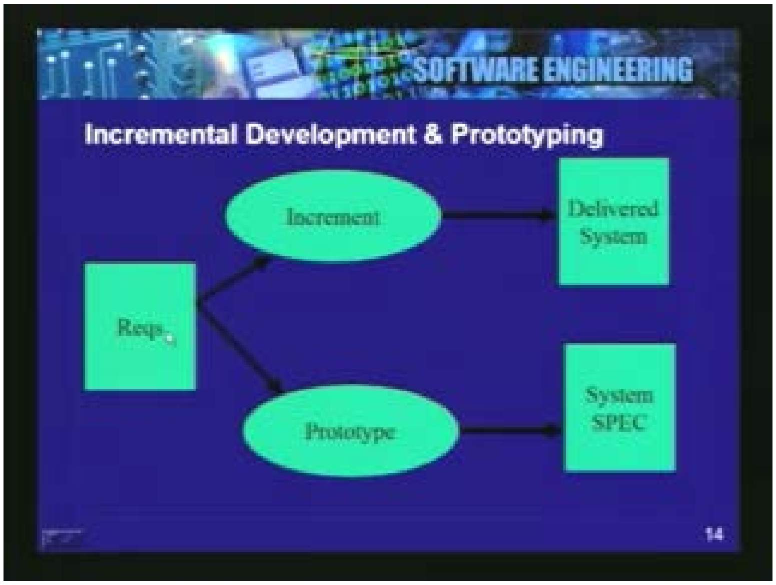
Prototypes are typically throw away in nature because of the fact that you have put something in the prototype to test functionality but that may not necessarily meet in non-functional needs of the system. Performance is a very good example which is most commonly encountered here. So the algorithm for example that you pick up maybe something that meets the functional needs of the specification, the functional needs of the user but it may be impossible to meet some of the performance needs of the user which is a non-functional need which is equally important at the end of the day. So the prototype will validate say the functional need and say you can certainly design the system using this idea, this design pattern and so on but it has to be developed from scratch now using different algorithms to enhance performance and so on, that is one thing.

Prototypes typically are very rarely documented so you do not want to keep at around it is done basically from a designer's perspective, it is done from the perspective of the person who is designing the system initially and so they do not focus on documentation, if that is kept around then you will be left with no documentation on the design of the system at the end of the day and that can prove very hard in maintenance.

The prototype structure usually gets very rapidly degraded. because of the amount of changes that is being made to the prototype you are touching the code very often and you are making small or lot of changes typically small changes very often because you are trying to tune the system at that point in time; you are trying to tune your design, you are trying to find out which is the best idea that are going to work within the context to these requirements so the structure is usually quite degraded after a period of time and maintainability drops as we have already seen in certain earlier lectures and it will most often not meet the quality standards of the organization so that there will be some coding standard for example that most software organizations end up tending to follow both with respect to documentation as well as with respect to code and most prototype do not meet that because that is not what they are meant to do, they are not meant to be real code in the first place they are meant to stress out certain ideas so these are what are called throw away prototypes.

More often they are not that prototype should be thrown away it is only in very very rare situations that they are retained and extended and augmented, enhanced etc so also meet the other requirements of the system.

(Refer Slide Time: 25:03 min)

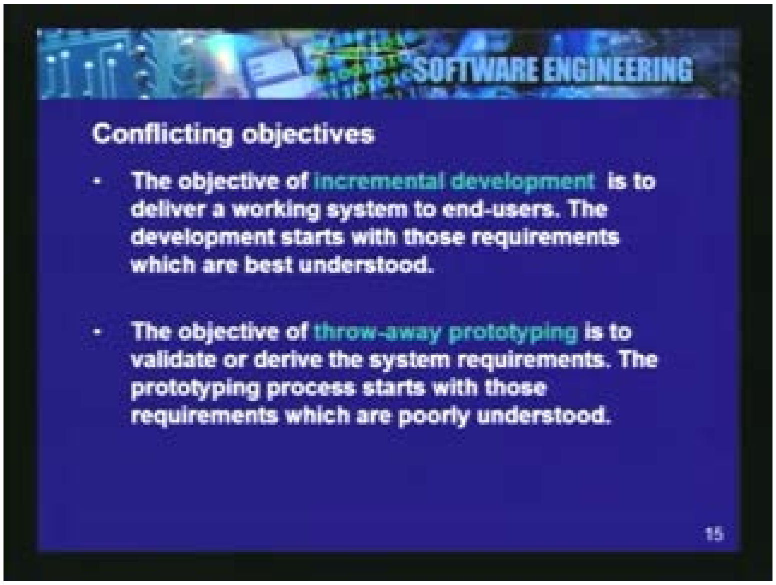


SOFTWARE ENGINEERING

Incremental Development & Prototyping

Therefore, the main difference stands between incremental development and prototyping is that both start with requirement and specification and the increment is basically resulting in a delivered system as we see here. So an increment is something that is actually used by the user eventually; it is put into production, it is actually used and then the user gives feedback on that so it is not a toy system it is not a test system in other words whereas if we go down the prototyping path that is become down this path then the prototype is constructed it is not incremented but it is constructed and the prototype essentially leads to a system specification; it could lead to a system design which is also a design specification of sort but essentially it leads to a better way of doing this; the best way of doing this is usually found out so the best set of requirements of the system is found out, the best specifications for the system, the best design for the system etc so those alternatives can be validated using prototypes.

(Refer Slide Time: 26:03min)



SOFTWARE ENGINEERING

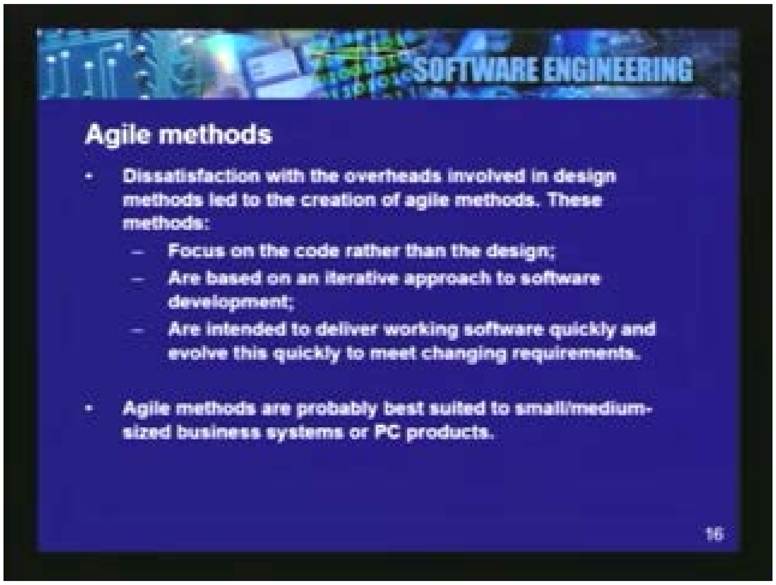
Conflicting objectives

- The objective of incremental development is to deliver a working system to end-users. The development starts with those requirements which are best understood.
- The objective of throw-away prototyping is to validate or derive the system requirements. The prototyping process starts with those requirements which are poorly understood.

15

So the conflicting objectives of these two different ways of handling agile development is that the objective of incremental development is always to deliver a working system to the user as quickly as possible but having very small amount of functionality. So, usually start with requirements which are best understood requirements, which are unambiguous, which are clear and so on whereas the objective of the throw away prototype in fact is to solidify those requirements that are very ambiguous so you typically start with a requirement which are very ambiguous, which are not well understood and try to prototype those ideas then show it to the customer the customer never ends up using the system in any way they only end up using it in a test mode just to see whether what they have asked to be constructed is indeed what they want so the prototyping process always starts with requirements that are very very poorly understood and this leads to a process of refining these requirements so that they are understood much better by the system designers going forward.

(Refer Slide Time: 27:09 min)



SOFTWARE ENGINEERING

Agile methods

- Dissatisfaction with the overheads involved in design methods led to the creation of agile methods. These methods:
  - Focus on the code rather than the design;
  - Are based on an iterative approach to software development;
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- Agile methods are probably best suited to small/medium-sized business systems or PC products.

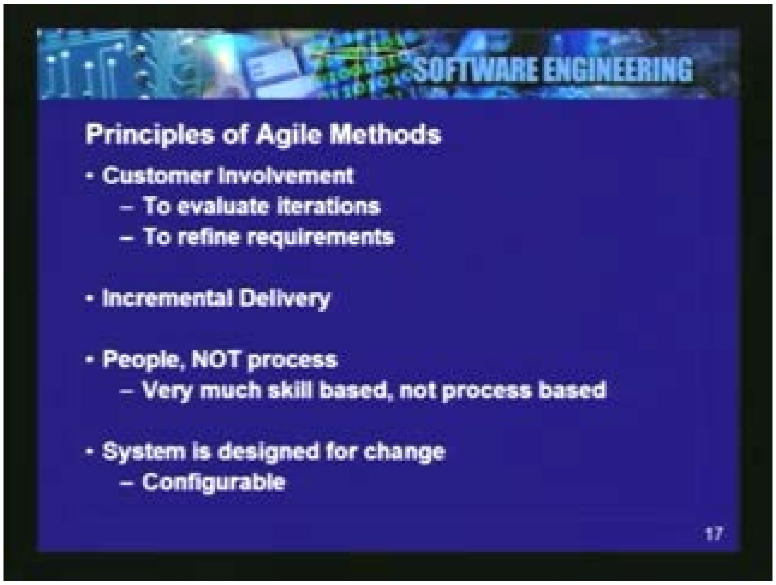
16

So agile methods are typically as we have seen best suited to small business small business size systems, small businesses, rapidly changing environments in which it is not very clear, so for example you do not tend to use agile methods in the development of software that runs the space shuttle
it is not something that is done that way; there formal methods would need to be used, the specifications would have to be understood perfectly before you can construct any piece of software and see it is not likely that you are going to use the software in incremental mode on the space shuttle but only when everything is perfect will the shuttle be launched. So, in those kinds of systems you do not use agile development.

Systems, on the other hand, that very commonly employ these kinds of methods are e-commerce systems in which features can be added on an incremental basis for the basic e-commerce site that is got to be up and running for a business to be functional very very quickly so they need to be able to place an order, they need to be able to validate a credit card and they need to be able to just take and process the order in the back end. So the placement of the order, the validation of the payment method may be something that is put out there very quickly, put out there first so that the business can get up and running and processing the order, for example, may not be automated for a long time in fact it may be completely manual behind the scene up until the point where the next increment delivers the order processing part of the system and so on.

Hence, typically the need for this is felt by companies where there is dissatisfaction at the high level of overhead that is introduced by the software development process; people need the software very quickly and the focus here is more on code rather than the design the design may not even live very long because of what is called re-factoring and re-factoring is the way of changing the design or changing the code to best meet today’s requirements, meet the requirements of now and not what the requirements are likely to be in future whereas the objective of a good design is always to meet its future proof, it is a future proof to the system to make sure that any new requirements may come in down the line and can also be accommodated in the system very easily. Agile methods are again in summary are based very much on iterative methods of the software development and intended to deliver working software very very quickly.

(Refer Slide Time: 29:48 min)



SOFTWARE ENGINEERING

Principles of Agile Methods

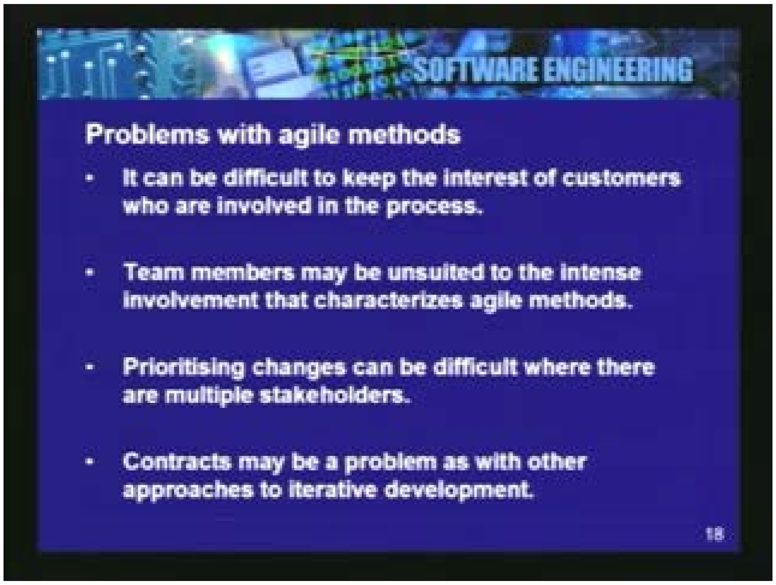
- Customer Involvement
  - To evaluate iterations
  - To refine requirements
- Incremental Delivery
- People, NOT process
  - Very much skill based, not process based
- System is designed for change
  - Configurable

17

So what is the process? For example, what are some of the principles behind agile methods or rapid application development?
The first principle clearly is that of customer involvement. We have seen that the user is continually involved; the user has to be involved in specifying the system, in quickly evaluating the increment that is coming out or prototype that is coming out, giving feedback and so on and so incremental delivery obviously is another principle of this process and this is a process that very much depends, the methods here, the methodology very much depends on the skill level of the people rather than on the process itself. So it is not process base it depends on how good the people are at working very rapidly to turn around software without appropriate designs, without formal specifications and so on so these would have to be people who can understand the customer's needs by looking at a initial set of requirements quickly working with the customer to deliver some software and the system is inherently designed for change.

For example, what does that mean when it comes down to reality is that everything that can be made configurable has to be made configurable? A good example of this might be that the system can work with any database. today we are working with a free database that is available and free software and this may be MySequel for example or (31:08…..sql) whereas a one steep system reaches a certain level of maturity we might have to switch over to Oracle, we might have to switch over Sybase and some other kind of relational database that is commercially accepted. So in such a case what we want to do is completely isolate the dependence of the system on the database itself by introducing some kind of a standard layer such as ODBC, JDBC and so on and then the exact database that you end up using is typically simply a configuration parameter that is placed in some kind of a configuration file and on changing the configuration parameters the system picks up the right driver to the database runs with it and is able to function on any database platform. So the system is designed for change; essentially there is not a lot of coupling between different modules of the system; this is something that is a standard software engineering metric that you want to reduce the coupling between modules and you want to increase the cohesion within a module and that has been followed in the extreme when it comes down to agile method because of the amount of change that is going on all the time within the software.

(Refer Slide Time: 32:17 min)


SOFTWARE ENGINEERING

Problems with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritising changes can be difficult where there are multiple stakeholders.
- Contracts may be a problem as with other approaches to iterative development.

18

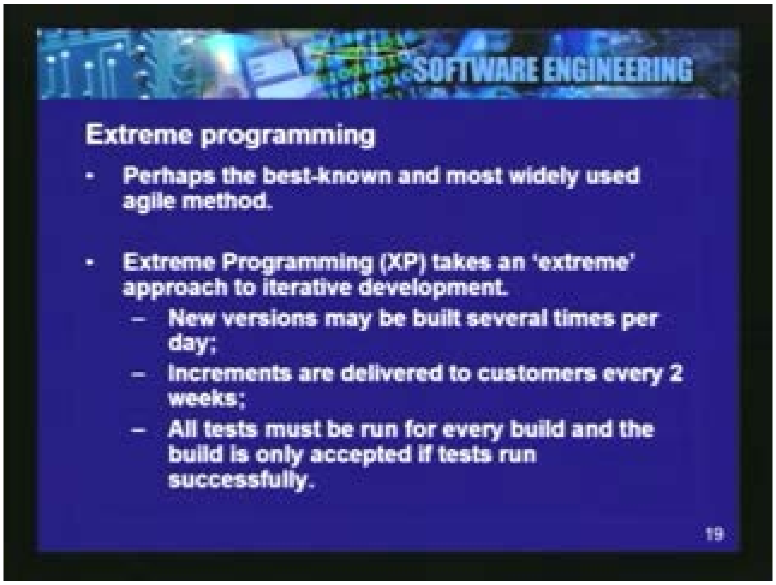
So the problems with agile methods are it can be difficult to keep the interest of the customers alive because they have to do so much more work in the development of the software here although they are not exactly sitting down and coding or designing they have to be constantly involved and to keep that involvement at a high level all the time may be very very difficult. It may be hard to assemble and maintain a team which get used to the very intense kind of environment that characterizes these methods because the software turnaround time could be small as two to three weeks and for a piece of software to get developed in two to three weeks implies that there is going to be some fairly intense activities in those two to three week period so not all people are capable of working at that pace and you have to pick the people who are right for that kind of environment who thrive on that kind of environment.

Also, prioritizing changes can become very difficult. So which are the requirements that need to go out first; the customer typically has the tendency to start saying I want all of them because all of them are important to me; and it is very hard for him to prioritize the set of changes that are going to go into the next increment. Also, there can be multiple stake holders using the system. So, if a system is used by accounting, by finance it is used by operation, it is used by legal and so on; each person will have a different perspective of what is important for them and to put out an increment which serves one group of stake holders but not another is going to end up alienating the rest of stake holders in the process and that would become a real problem. And as we have already seen earlier contracts and the process of writing contracts would be very very hard in this process. So, having seen the general principles involved in agile software development let us take a look at a very specific process that has become very very popular for the past five years or so; it is called extreme programming.

Extreme programming is probably the best known and the most widely used agile method today especially in open source development where there are lot of people who use but certainly in lot of companies as well and there are various flavors in extreme programming it is not a practice that lays down like a ISO: 9001 process that you know this, this and this have to be done it is a set of principles guiding principles that have been put together to say try to do software development this way and here is the way by which you can do rapid application development which impossibly works for you.

As the name kind of implies it takes a very extreme view or an approach towards iterative development. For example, there can be several builds happening per day; by build mean I am compiling all your code, integrating everything and running the tests increments delivery to the customer every two weeks that is pretty much fixed and it is a very short cycle time to be delivering software; if you work on the software project you will understand what I mean.

(Refer Slide Time: 35:07 min)



Extreme programming

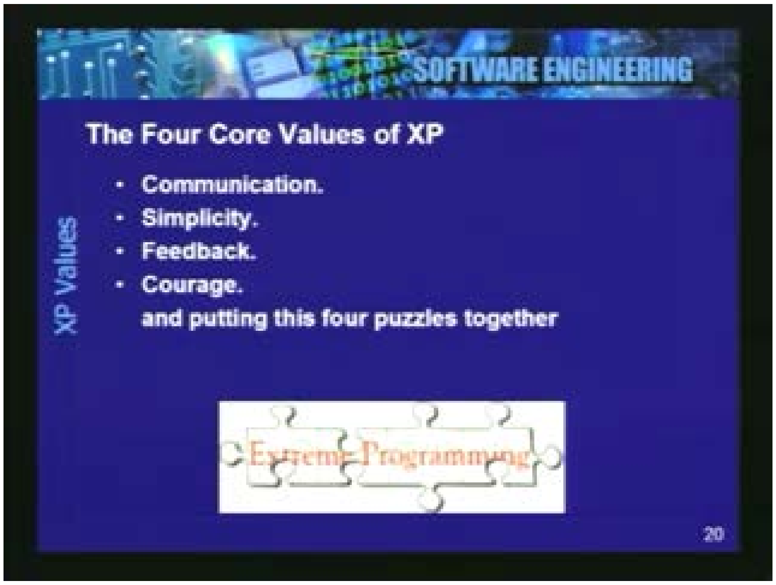
- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

19

And every test must run for every build and the tests typically have to be written even before the code is written that the tests are going to test. So it is a strange, initially it sounds strange it is an extreme form of agile development like I said. Let us take a look at what some of the elements of extreme programming are.

Over the period of the last several years Kent Beck was one of the founders of extreme programming. There has been several books written on this and this whole thing is solidified into a fairly well understood process today. the four core values as we call them of extreme programming are the first thing is the ability to communicate; so the communication has to be very very strong and tight in the team, the second thing is simplicity, the third is feedback and the fourth is courage and all these things have to come together in order for an XP project to be successful. Although each one of these sounds a little vague there is actually a very definite process behind each one of these how we can achieve good communication and so on.

(Refer Slide Time: 35:59 min)


SOFTWARE ENGINEERING
XP Values
Extreme Programming
20

The Four Core Values of XP

- Communication.
- Simplicity.
- Feedback.
- Courage.

and putting this four puzzles together

So let us first look at communication which is the first XP value. Communication is basically enforced through a series of practices that XP has woven into its process which cannot be carried out without communication and one of the examples of this is pair programming.

Pair programming is the notion that two developers sit down together to write the same piece of code. In fact they do need a single workstation so they are not working independently and then comparing. as one developer is working and writing the code the other developer kind of watches over his shoulders, kind of helps him avoid the mistakes that he would otherwise end up making which is going to be caught later in the cycle. So instead of catching mistakes later, so, for example, you write back code and it is going to fail a test later if your other pair developer is going to catch that error right as you are coding it then you can save a lot of time and effort so that is one example of communication.

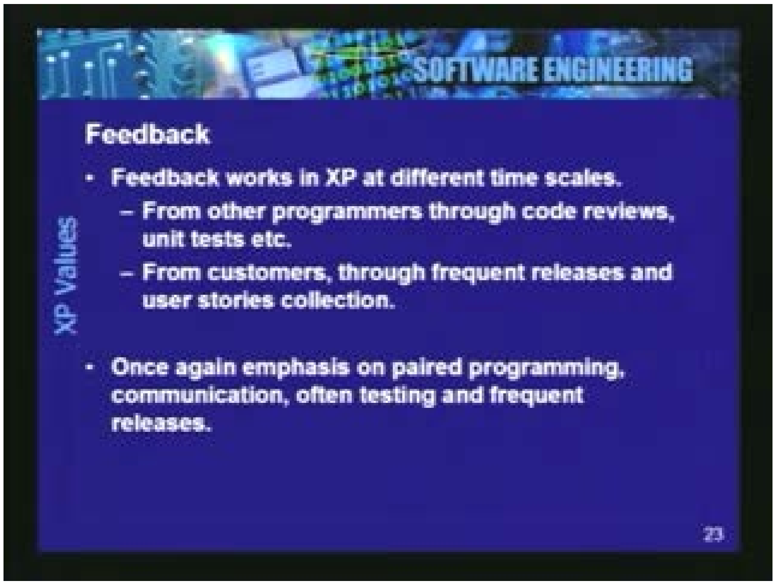
The other practice is that of frequent integration because of the fact that your code is integrating with somebody else’s change may be several times a day you are kind of forced to make sure that you stay in touch with everybody and explain where things went wrong, how you can fix them and how quickly you can fix them and so on.

In fact XP or extreme programming XP as it is called for short also advocates the job of a coach; a role that is taken on by one of the team members whose only job is to make sure that people are talking to each other all the time and what they see is that this is not happening he may end up reintroducing his people and fixing these problems so that is a very important role in the XP world.

So the second code principle or value of XP was that of simplicity and there are two ways of going about building software, actually building anything but building software as well. One way is to kind of design for the future. You put in certain features that you think that somebody might want to ask you down the road, now why build it later if we can build it into the software right now when I am building the module that is one philosophy.

The other philosophy is to keep things really simple up until the point that you need the feature and then you build it in. So you are not afraid of changing the code, you are not afraid of adding to the code but you can do that at the point in time if it is required so why complicate this system the features that are never going to be used up until some later point in time if you can build a simpler system. So simplicity and communication obviously support each other so you should only built minimalist systems, minimalist modules, minimalist units which are put together and these can be changed as the software development process goes on.

(Refer Slide Time: 39:24 min)



SOFTWARE ENGINEERING

XP Values

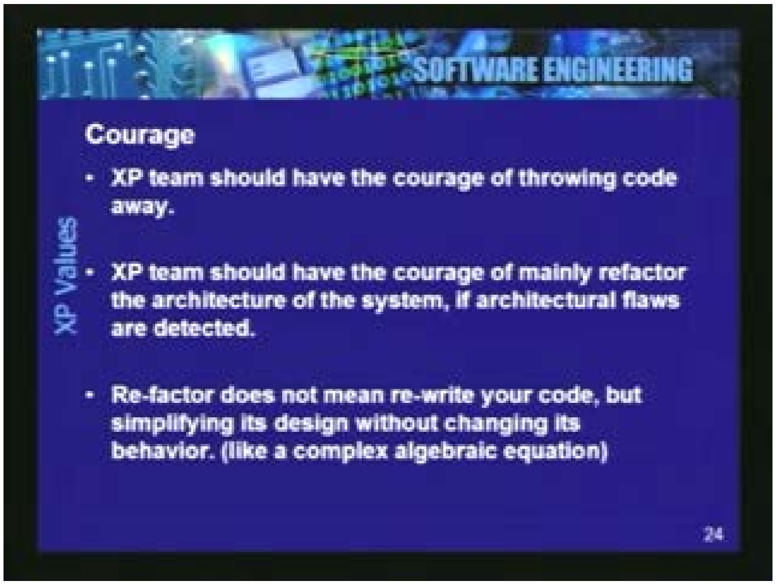
Feedback

- Feedback works in XP at different time scales.
  - From other programmers through code reviews, unit tests etc.
  - From customers, through frequent releases and user stories collection.
- Once again emphasis on paired programming, communication, often testing and frequent releases.

23

The third value is that of feedback and feedback in XP works at various levels. One is certainly the user feedback because of the increments that have been delivered to the user through different story collections that happen from the user; that is, the requirements have been changed because story is with an XP and it also happens amongst the different developers themselves so when pair programming is taking place one developer is giving the other one feedback, there are frequent reviews happening within this process where one programmer presents the idea of what is it that they have done and the other designers or the programmers in the team sit down and give them feedback on what could have been done better and what are the things that are working really well and so on. So the emphasis is on various values that encourage a very very tight feedback cycle within the process.

(Refer Slide Time: 40:07 min)



SOFTWARE ENGINEERING
XP Values
24

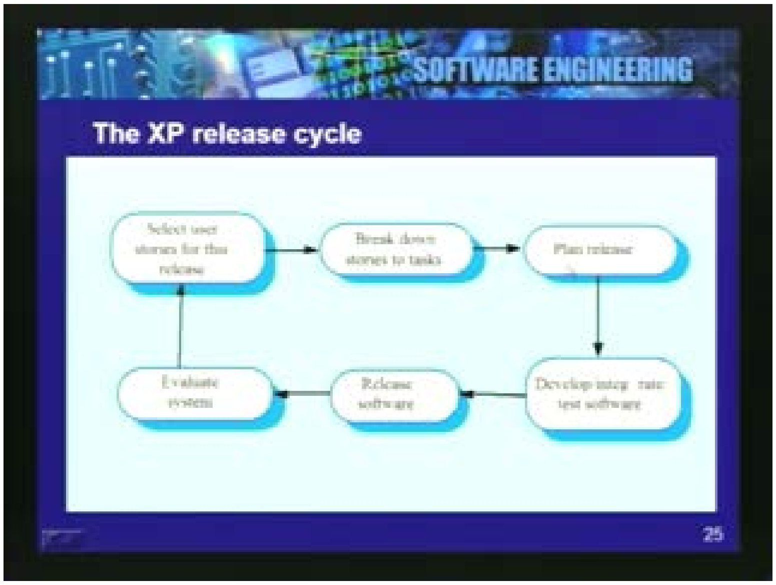
Courage

- XP team should have the courage of throwing code away.
- XP team should have the courage of mainly refactor the architecture of the system, if architectural flaws are detected.
- Re-factor does not mean re-write your code, but simplifying its design without changing its behavior. (like a complex algebraic equation)

And the fourth value is that of courage and by courage I mean that XP might ask you to re-factor which means you could end up to throwing your old code away much like the prototyping case; although people are typically very very hesitant to throw away work but they already done even understanding that the work is not good that it could be harmful in the future and here you should absolutely fight against that tendency because of the fact that you may only have written a very small of the system it may not be large enough for you to want to keep it all the time so you can actually throw it away and develop this whole thing anew and that is something that XP encourages quite a bit.

What is the release cycle for extreme programming look like?

(Refer Slide Time: 40:53)



SOFTWARE ENGINEERING

The XP release cycle

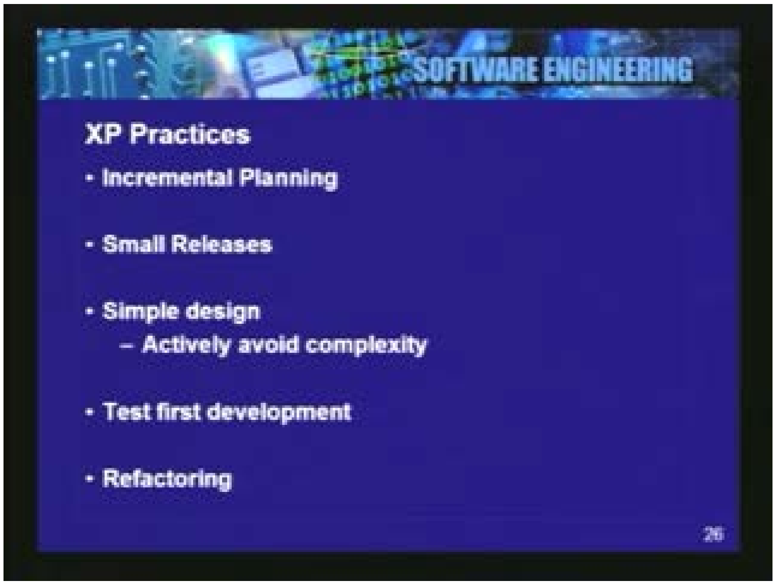
Select user stories for this release
Break down stories to tasks
Plan release
Develop integrate test software
Release software
Evaluate system
25

there are Remember these subset of requirements that go into an increment. This is nothing but an agile development process. The subset of requirement is called story; a story concentrates on a very small requirement or a subset requirement that are all related to each other and we will go through an example story as part of this lesson. So you break that story, you take that particular story that is going to be part of this increment and only typically one story is part of the increment; you break it down into a set of tasks as this diagram shows you and then you plan the release based on the set of tasks that have to be done. at this point in time you kind of know what the effect level required of that increment is going to be does it fit into the two week cycle is the question you have to ask yourself; if it does not then it is a wrong story that you have picked and you might have go back and change it.

Therefore, you develop you develop the software, you integrate, you test the software, release the software; this is a formal release unlike something that is done in prototyping. So this is something that is delivered to the customers, accepted by the customer and then you move on and then you end up evaluating the system, you select the stories for the next release. So this very much looks like the process that we saw earlier of iterative development, very very similar. In fact this is nothing but a process of iterative development specifically applied to this scenario.

So, what are some of the XP practices is something that we would like to go through next.

(Refer Slide Time: 42:32 min)


SOFTWARE ENGINEERING
26

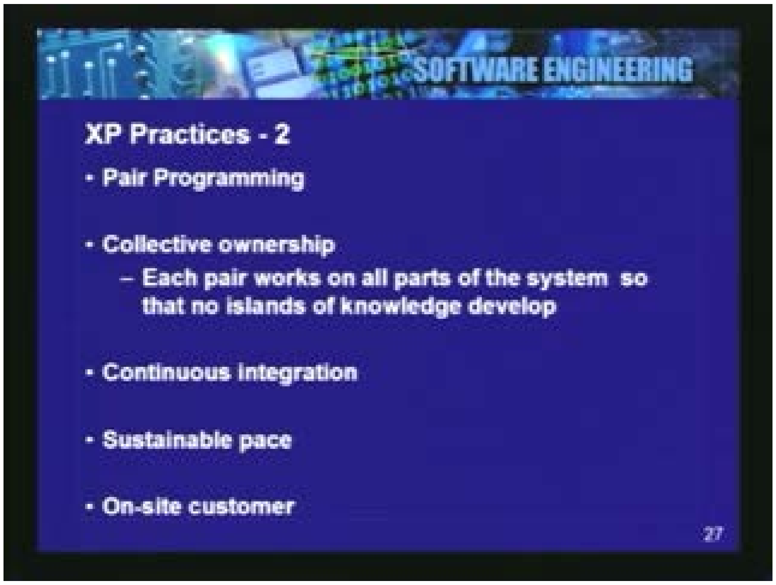
XP Practices

- Incremental Planning
- Small Releases
- Simple design
  - Actively avoid complexity
- Test first development
- Refactoring

Incremental planning as we have seen before what characterizes agile development. Incremental planning is something that is absolutely followed very regressively here. Small releases: two week cycle is pretty much mandated in extreme programming so you have to release some piece of functionality into every two weeks and that is the idea here. simple design: actively avoiding complexity so you want to make an active effort to just put in the set of features that are minimally required of that iteration and not of something that is going to appear three iterations that you feel that is going to appear three iterations later as always a tendency of people to want to make things robust or want to make things larger than what it should be for the current moment and even may end up jeopardizing by making the user paying for a feature that he never uses because it might affect performance, it would affect the availability and so on.

The other important XP practice is what is called test first development and there you develop the test a unit test even before you write the code and you will see what this is. Re-factoring is the next one. Re-factoring is the ability not necessarily to just rewrite the code but re-factoring is the ability to maybe change your design by without changing the behavior. So, for example, the way different modules are interconnected with each other could be one such example so you may end up changing the design without actually having to rewrite a large amount of code and that is what re-factoring is.

(Refer Slide Time: 44:03 min)



SOFTWARE ENGINEERING

XP Practices - 2

- Pair Programming
- Collective ownership
  - Each pair works on all parts of the system so that no islands of knowledge develop
- Continuous integration
- Sustainable pace
- On-site customer

27

Pair programming: Something we already talked about where two or more people typically two people sit down and write a part of code together.

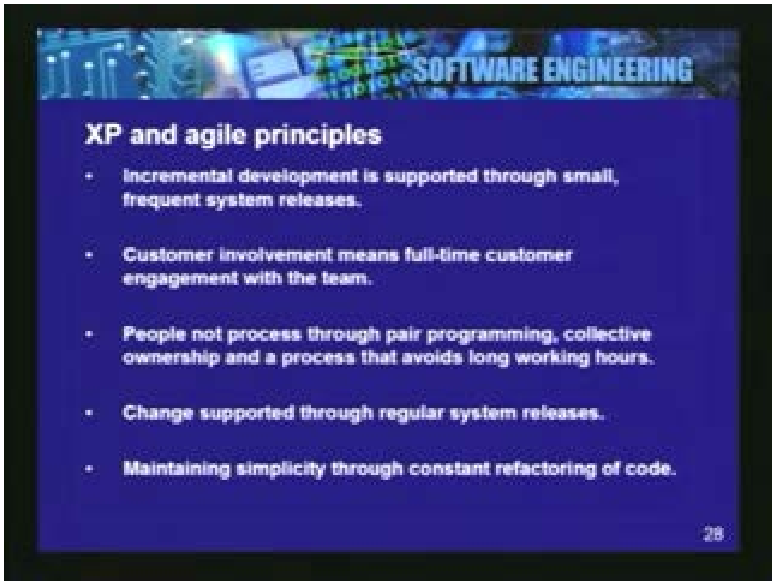
Collective ownership: where it often what happens in software development processes is that a group of developers get wedded to one or two modules some x modules of the system and they know those modules very well but they do not want to understand the rest of the system at all and that proves very fatal because if those group of programmers leave or for some reasons they have to be moved out of the project then there is nobody else who understands that part of the system very well.

So what XP does is try to introduce the practice where people move around different modules they are not wedded to one module of the system as a result of which they typically tend to get a good idea of all parts of the system and there is no islands of knowledge that is going to end up developing.

The next practice is continuous integration. Continuous integration is one where multiple people changes code changes are brought together frequently. So, typically twice a day is how you end up tending to integrate and for every integration what is more important is to make sure that you test the integration; it is not just that you stored the code together and then you move on you have to measure that it all complies together so you might have interfaces that are being used by other people and it may not compile because you made a change in the interface that is being used by somebody else; it also has to pass all the tests; it just not has to compile but remember that since the tests have been written even before the code is being written it has to pass all the tests.

You have to make sure that you keep a sustainable pace of development; it is not something that…… XP in fact actively discourages people from working very very long hours so the standard notion of overtime does not exist in this kind of a model so you have to keep a sustainable pace of development and not something that goes very high for a period of time and then drops drastically after that. It also demands an onsite customer involvement so there needs to be somebody who pretty must stays with the development team on a continual basis; it is not something that can be thrown out to the ward of the customer say you take a look at it and come back; the customer is often involve d in development as well to some extent.

(Refer Slide Time: 46:24 min)


SOFTWARE ENGINEERING

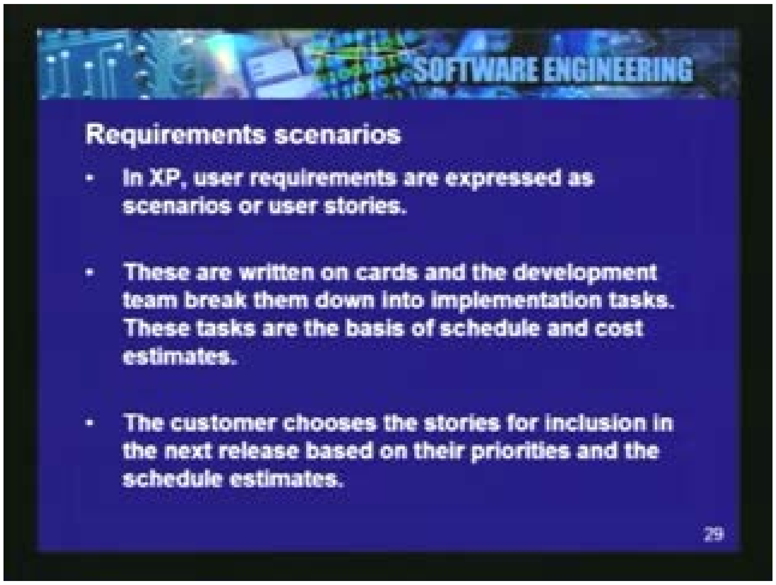
XP and agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

28

So the agile principles are pretty obvious. XP is clearly a way of doing agile development as well and we have seen that the focus is on people here and not processes, change is supported because of the regular system releases that are already built in and so on and customer involvement is pretty continuous.

(Refer Slide Time: 47:05 min)



SOFTWARE ENGINEERING

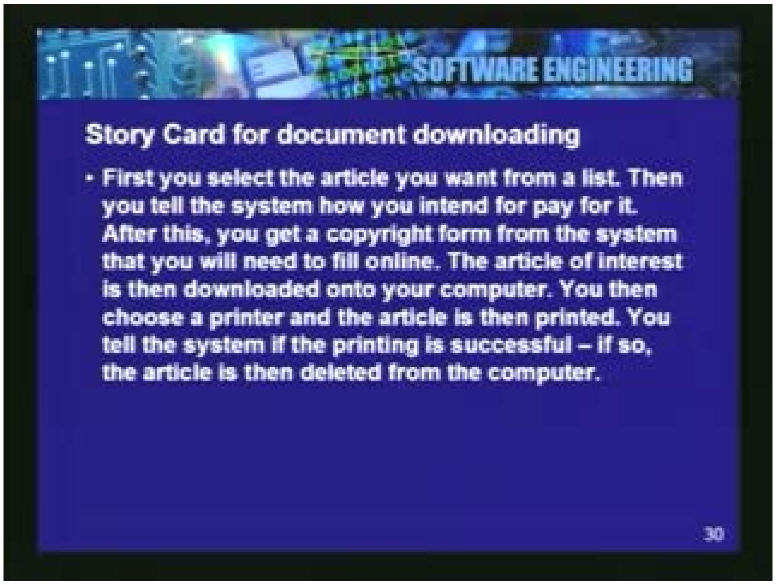
Requirements scenarios

- In XP, user requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

29

So, requirement scenarios in XP are basically called stories. Let us take a brief look at how XP functions internally and these stories are basically written on cards and the development team breaks each story that they pick for that increment down into a set of implementation tasks. There may not be an overall design; certainly there may be some architecture that the team adheres to. So, for example, this is a service oriented architecture that you are going to use, this is the pipes and filter architecture that we are going to use and that decision may be made very early in the life cycle of the entire software project or may be even before the first increment is done but going beyond there there is no emphasis on designing individual modules by themselves so the tasks are basically implementation tasks and these are the basis of creating the schedule and the cost estimate. So only after you have broken them into tasks and since you are doing it on a very small scale you can then say this iteration is going to cost x, there is no cost for the entire project it is very hard to determine them because you have not planned out all the iterations ahead of time and these are done as you go long. So the customers are the ones that end up choosing the stories for inclusion of a particular increment and that was then to drive the cost for increment and so on.

(Refer Slide Time: 48:26 min)



SOFTWARE ENGINEERING

Story Card for document downloading

- First you select the article you want from a list. Then you tell the system how you intend for pay for it. After this, you get a copyright form from the system that you will need to fill online. The article of interest is then downloaded onto your computer. You then choose a printer and the article is then printed. You tell the system if the printing is successful – if so, the article is then deleted from the computer.

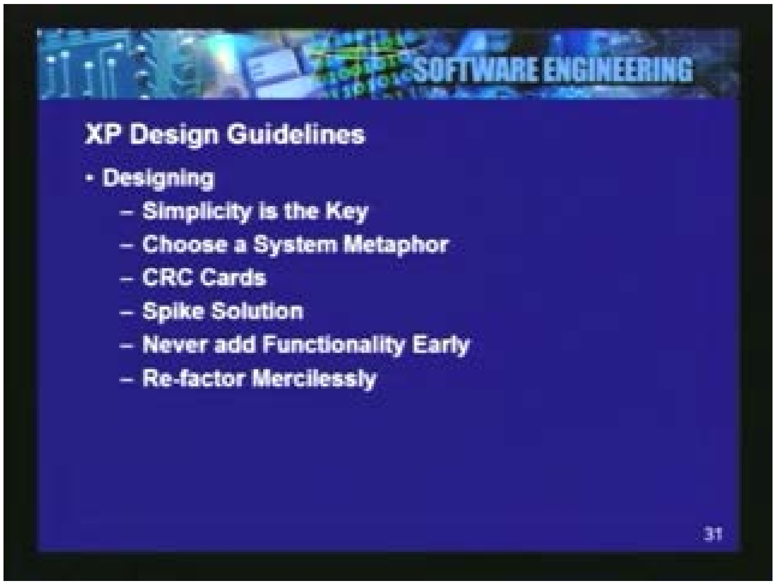
30

So here is an example of a story card that we have written out. The story card is with reference to the library system that we talked about in earlier lectures in this course and here is the story card for downloading a document. So the assumption is that people can download documents and print them for a certain fee. So the story goes thus:

First you select the article you want from the list and once you selected that article you tell the system how you intend to pay for it then you get a copyright from the system, copyright form from the system that you have to sign and submit back to the system; you can fill this online and once you have submitted it the system downloads the document onto your computer, it asks you to select a printer onto which this document is going to be printed, prints the document and then deletes the document from your computer because you are not allowed to use it any further; that one printed copy that you have is pretty much all you are entitled to of that payment because the copyright are restrictions on a document.

This is the story card. This is the story; it kind of details an entire user's scenario if you will. So, if you take if you take an analogy to uml used cases this may be one used case in the system or it may be one or two used cases which are related to each other. So the design guidelines then for XP is that simplicity obviously is the very key and is the heart of the entire design process so you choose some kind of a system metaphor and you can write out CRC cards as in the object oriented design process you must have heard of CRC card notion then you create a spike solution never add functionality too early in the process and you can always start off with very very small stop but then you re-factor as much as you want to add things as you go along as opposed to committing very early on in the process to requirements that may not be needed.

(Refer Slide Time: 50:06 min)



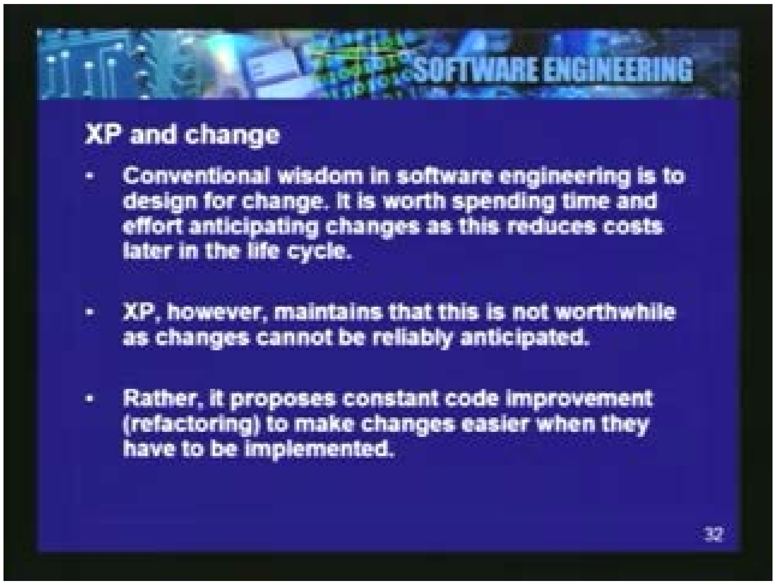
SOFTWARE ENGINEERING

XP Design Guidelines

- Designing
  - Simplicity is the Key
  - Choose a System Metaphor
  - CRC Cards
  - Spike Solution
  - Never add Functionality Early
  - Re-factor Mercilessly

31

(Refer Slide Time: 50:27 min)



SOFTWARE ENGINEERING

XP and change

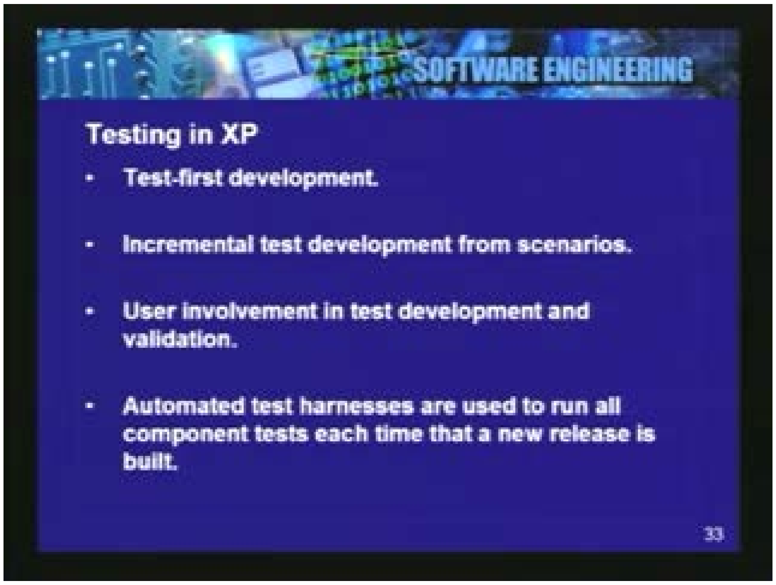
- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

32

So, in terms of change basically XP takes the opposite view of the conventional wisdom of change and in that it basically anticipates tht change is going to happen all the time and the conventional views that it is worth anticipating changes and therefore let us build it into the software ahead of time and therefore we have future proof in the software.

XP however is completely opposite; it says we will only build what is required now and as we go along when changes need to take place we do them and there is no problem for this.

(Refer Slide Time: 51:08 min)



SOFTWARE ENGINEERING

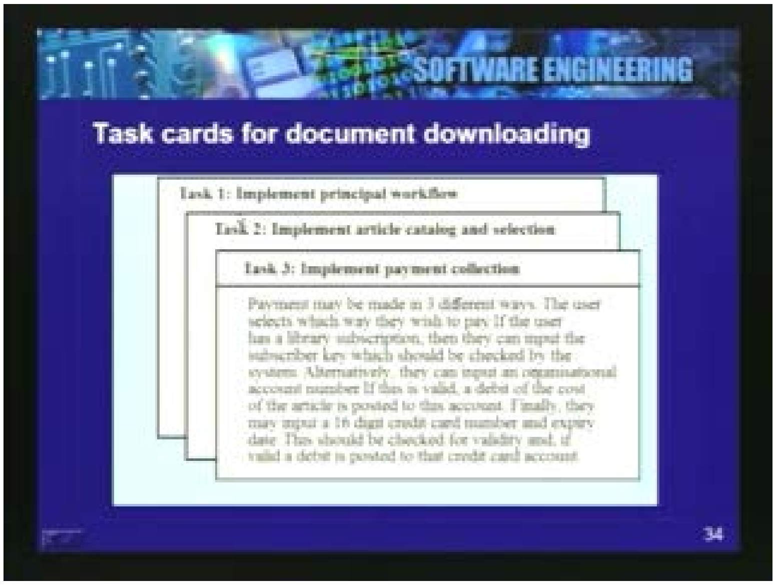
Testing in XP

- Test-first development.
- Incremental test development from scenarios.
- User involvement in test development and validation.
- Automated test harnesses are used to run all component tests each time that a new release is built.

33

Testing: you always do test first, so you have to do the unit test first, integration test and all of that tests have to be built out for the scenario or the user story that has been selected for that increment. So, only after you have written the unit test you go ahead and code the module. So the user development automated test harnesses have to be set up very early on in the life cycle and this can be done using several methods and tools that are available out there and every time something is released it is completely tested so every integration is unit tested, is integration tested before we move on to the next iteration.

(Refer Slide Time: 51:55 min)



SOFTWARE ENGINEERING

Task cards for document downloading

Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

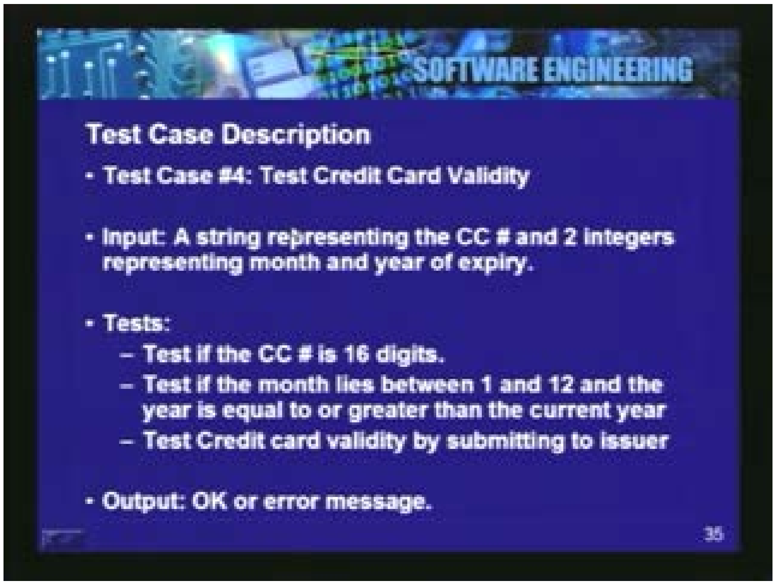
Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

34

The task cards: These are the examples for the task cards. There can be three different tasks so that story that we just laid out the story was that of document downloading. The three tasks here is that you implement the principle workflow the workflow is that you will select the document, you would select the payment method, you would fill in the payment details, it would validate the payment details, it would give you a copyright form etc etc so that is the workflow. Then you implement the article, catalog and the selection so this may be a drop down menu, this may be a set of articles that come onto your html screen etc etc and the third thing is you implement payment collection; payment collection may be collecting a credit card information, validating the credit card information with the credit card provider and so on and so forth. So, that particular story was broken down into three different tasks and when you have tasks at this granularity it is easy to determine what the cost of this is going to be.

(Refer Slide Time: 52:39 min)



SOFTWARE ENGINEERING

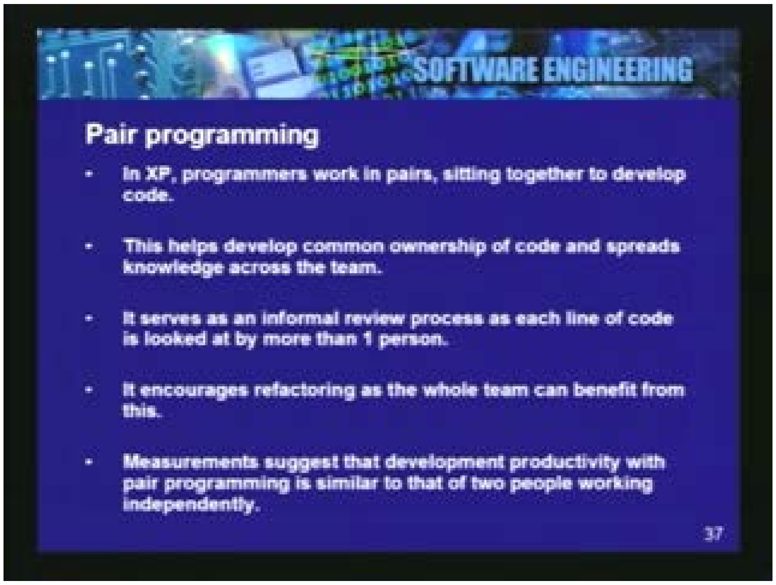
Test Case Description

- Test Case #4: Test Credit Card Validity
- Input: A string representing the CC # and 2 integers representing month and year of expiry.
- Tests:
  - Test if the CC # is 16 digits.
  - Test if the month lies between 1 and 12 and the year is equal to or greater than the current year
  - Test Credit card validity by submitting to issuer
- Output: OK or error message.

35

Here is an example of a test case description so you are testing credit card validity in this case. So the input to this test process is two sets of strings; one string is the credit card number, the second string is two integers which will represent month and year of expiry; I mean the second input is two integers that will represent the month and year of expiry then you have to run all the tests, you have to make certain validation before you go and check whether the credit card itself is valid by submitting it to the issuer of the card and the output of this test is okay or error message; an example of the test that is written out even before development starts that is what is key and that is what test-first development is all about.

(Refer Slide Time: 53:28 min)


SOFTWARE ENGINEERING

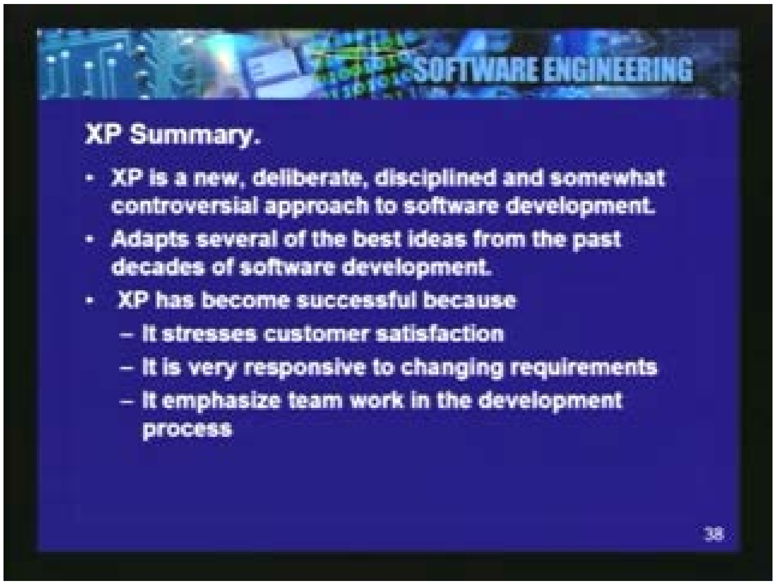
Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

37

Pair programming we also discussed and this is something where programmers work in pair sitting down together to develop code, it also helps develop common ownership of the code as well as encourages the communication value that we talked about earlier, it also serves as an in-built review process so constantly re-factoring then and there because there is somebody giving you feedback the moment that you are writing the code and not something that happens far later.

(Refer Slide Time: 53:54 min)


SOFTWARE ENGINEERING

XP Summary.

- XP is a new, deliberate, disciplined and somewhat controversial approach to software development.
- Adapts several of the best ideas from the past decades of software development.
- XP has become successful because
  - It stresses customer satisfaction
  - It is very responsive to changing requirements
  - It emphasize team work in the development process

38

So, in summary, XP is a new approach, it is very deliberate, disciplined and designed and somewhat controversial because not everybody accepts it because some of the changes to the way of thinking that have to be done and does adapt several of the best ideas from past decades of software development that are taking place; it has become very very successful because it stresses customer satisfaction because of the fact that the customers are involved all the time right from the first two week increment that is delivered the customer is in the loop and he is giving you continuous feedback on what they like about it and what they do not like about the way development is progressing and it emphasizes team work in the development process.

So this is the summary of extreme programming which is an agile development process and there are some other processes as well but XP is one of the most popular ones to watch out for.