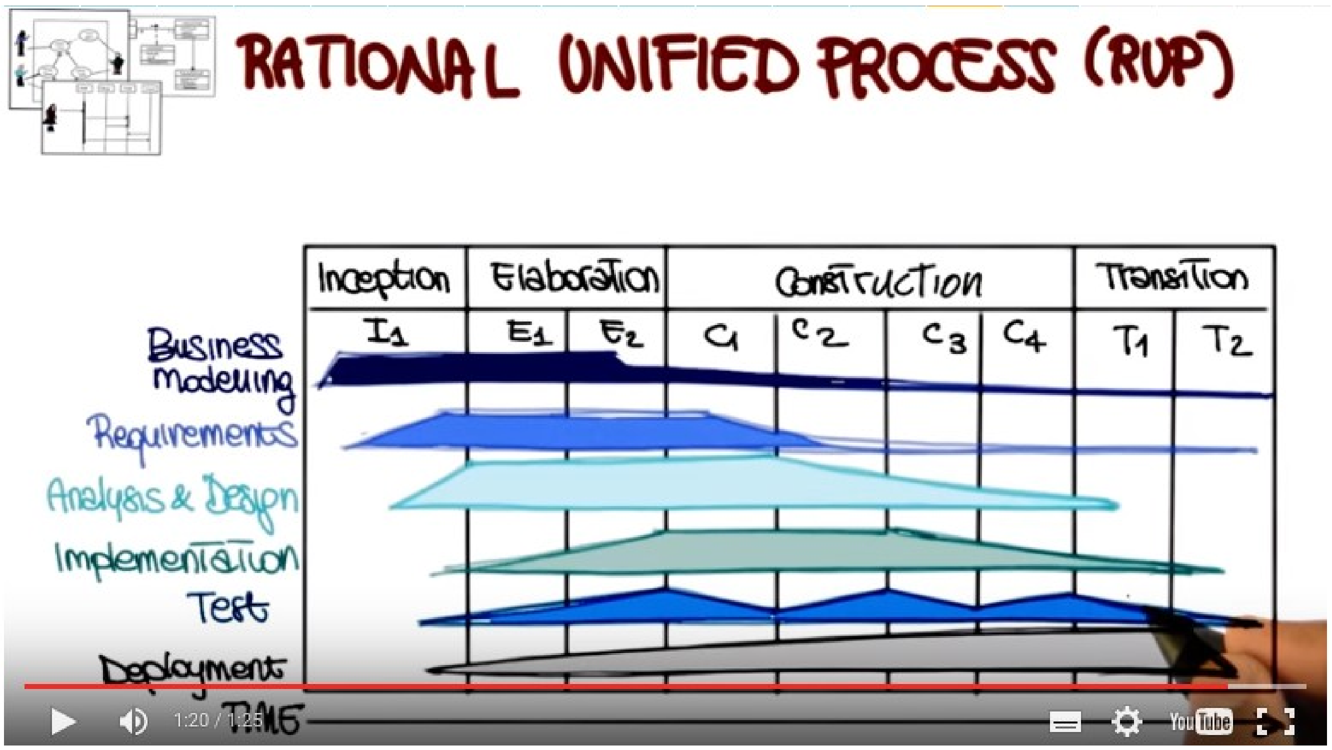
尼瑪, 不小心從最後一節開始聽的, 還截了圖, 還想畫重點, 結果在開頭找不到文字, 才發現是聽最後一節.

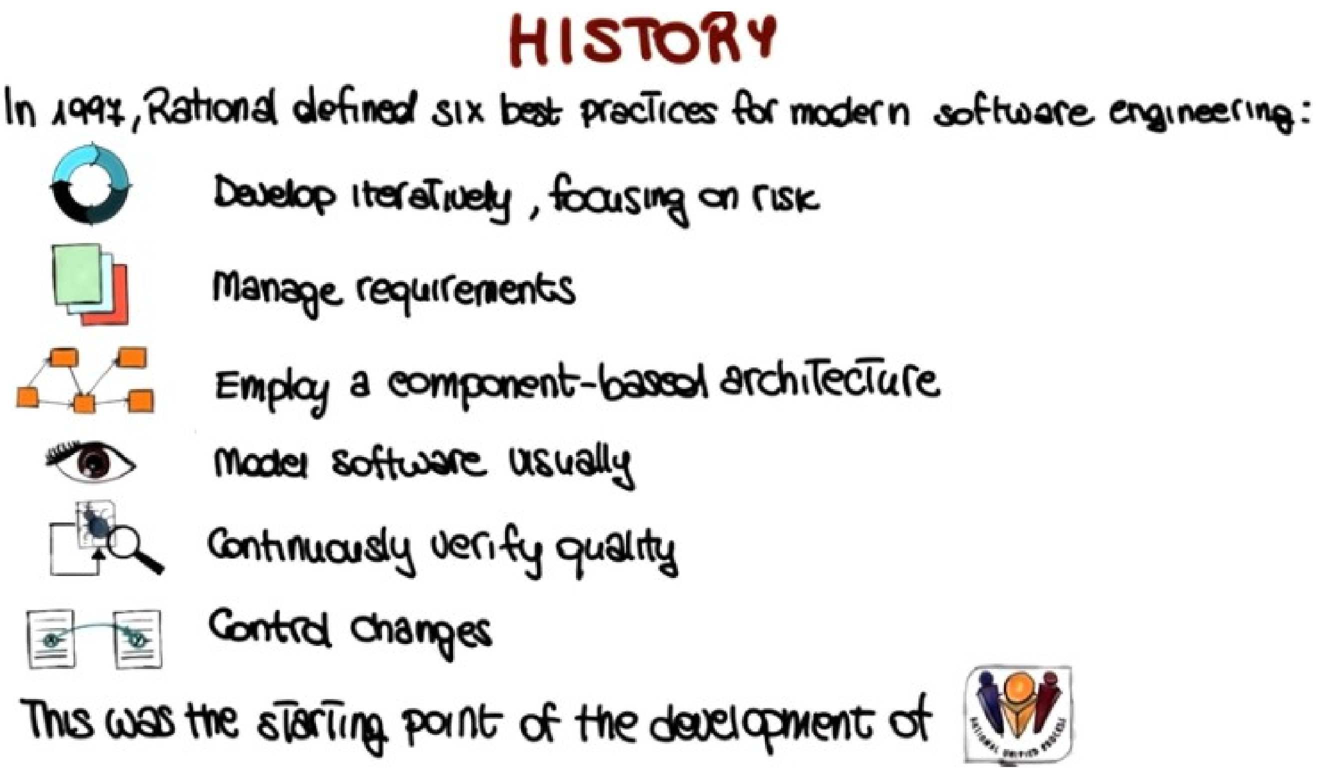
USP 就是跟 Waterfall 一樣的一個 Software process model (已驗證), 之前講過, 但不用回去看, 看以下的圖就明白了：



1. In the previous lessons of this mini-course, we discussed high level design, or architecture, low level design, and design patterns. Now, we're going to see how we can put this and also others software engineering activities together in the context of a UML-based process model, the unified software process, or USP. We will discuss how USP was defined, its main characteristics, its phases, and how we can apply it in practice.

2. As Seth just said, today we're going to talk about the Rational Unified Process. And you know that I like to provide the historical perspective of the topics that we cover in class and this lesson is no exception. So let's see a little bit of history of RUP. To do that we have to go back to 1997 when rational defined six best practices for modern software engineering. So let's look at what these practices were. The first practice involved developing in an iterative way with risk as the primary iteration driver. The second practice had to do with managing requirements, including updating them and keeping trace ability information between requirements. And other software artifacts. Practice number three was to employ a component-based architecture. What that means is to have a high level design that focuses on cooperating components that are nevertheless very cohesive and highly decoupled. Modeling software visually is another key aspect of the rational unified process. And the key concept here is to use visual diagrams, and in particular UML visual diagrams, in a very extensive way so as to make artifacts easier to understand and agree upon among stakeholders. And the fact that the process is defined in an iterative way, allows for performing quality assurance activities in a continuous way. So it allows for continuously verifying quality throughout the development process. Finally, change management and control were also at the center of the rational approach These six practices, that I just mentioned were the starting point for the development of the Rational Unified Process, which is what we're going to

discuss next.



HISTORY

In 1997, Rational defined six best practices for modern software engineering:

Develop iteratively, focusing on risk

Manage requirements

Employ a component-based architecture

Model software visually

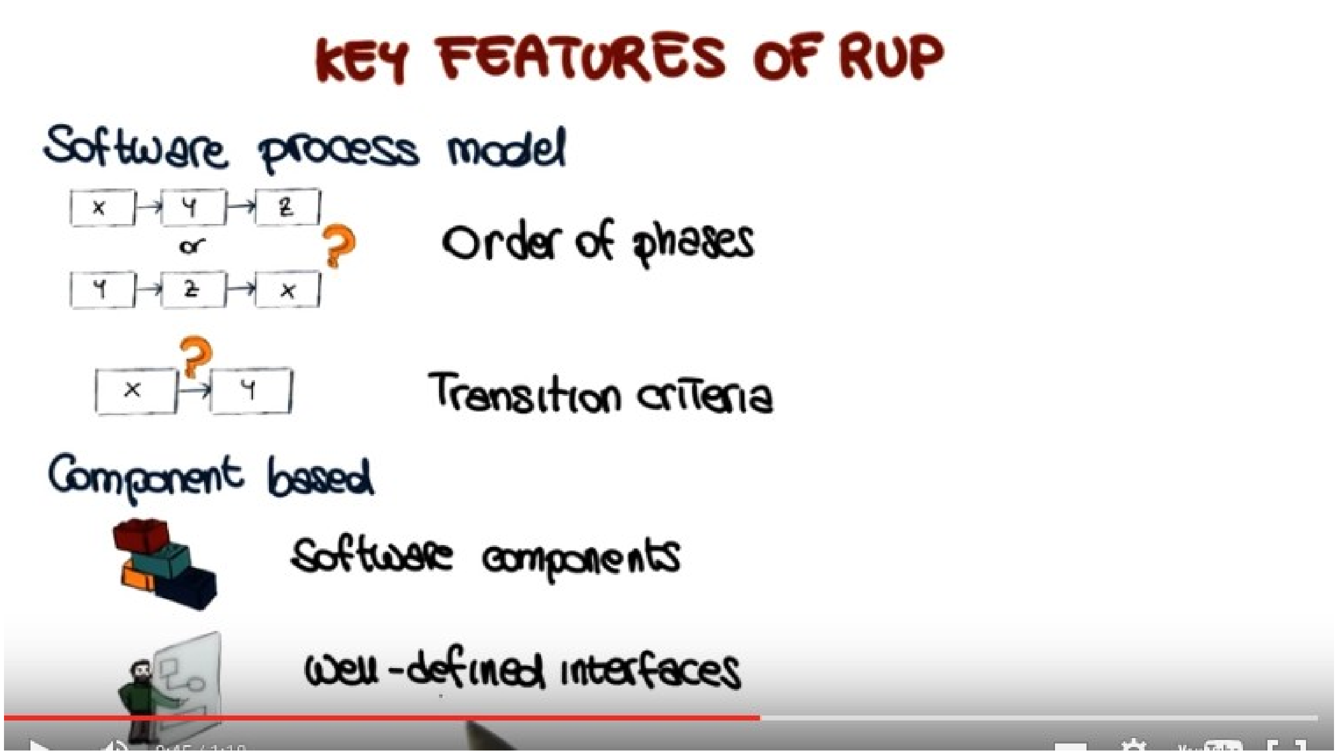
Continuously verify quality

Control changes

This was the starting point of the development of

RATIONAL UNIFIED PROCESS

3. So let's start by seeing how these activities and principles are reflected in the key features of the Rational Unified Process. First of all, the Rational Unified Process is a software process model. So if you recall our introductory lessons, that means two main things. The first one is that it defines an order of phases that have to be followed in the software process. And the second thing is that it also prescribes transition criteria, so when to go from one phase to the next. The second key feature of RUP is that it is component based. And also in this case, this implies to main things. The first one is that a software system is defined and built as a set of software components. So software components are the building blocks of our software system. And the second one is that there must be well-defined interfaces between these components, interfaces through which these components communicate. In addition, the Rational Unified Process is tightly related to UML. And in particular, it relies extensively on UML for its notation, and with respect to its basic principles. Finally, the three man distinguishing aspects of the Rational Unified Process are that it is use-case driven, architecture-centric and iterative and incremental. So let's now look in more detail at these three distinguishing aspects, and we're going to look at each one of them individually.



KEY FEATURES OF RUP

Software process model

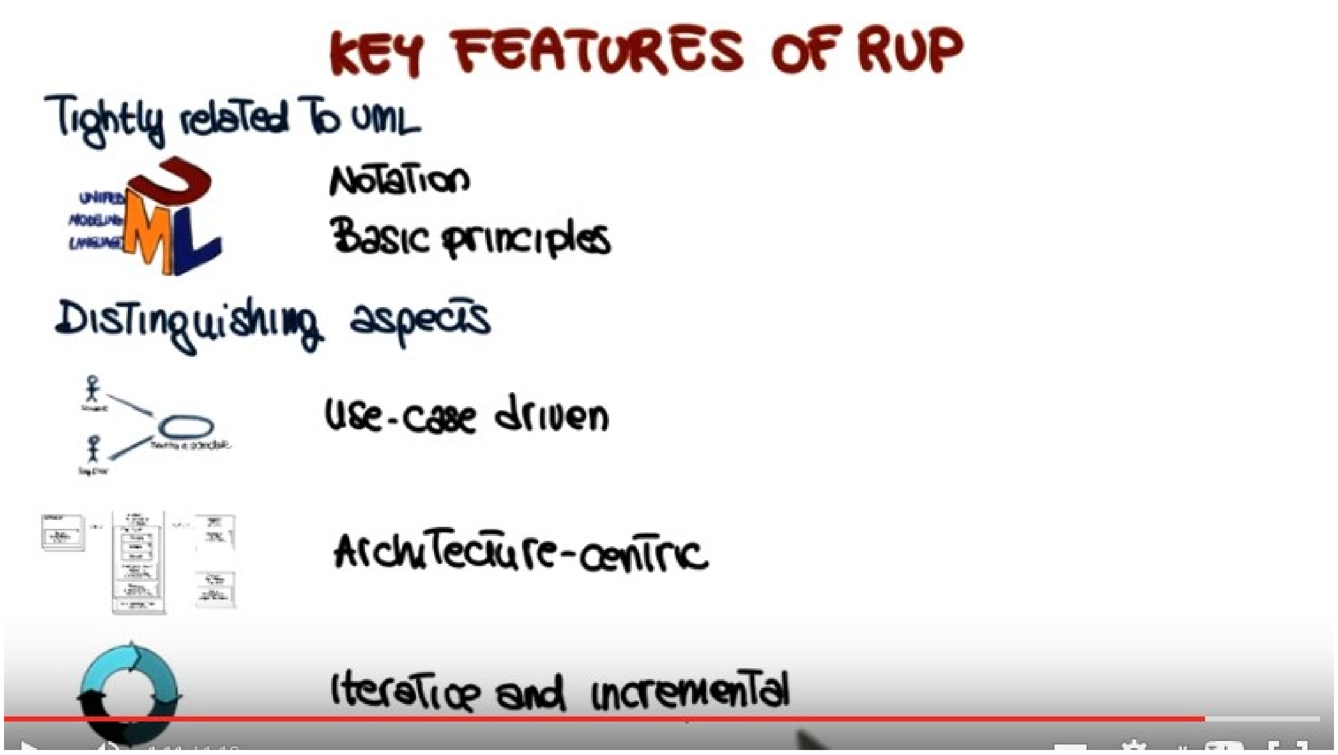
Order of phases

Transition criteria

Component based

Software components

Well-defined interfaces



KEY FEATURES OF RUP

Tightly related To UML

Notation

Basic principles

Distinguishing aspects

Use-case driven

Architecture-centric

Iterative and incremental

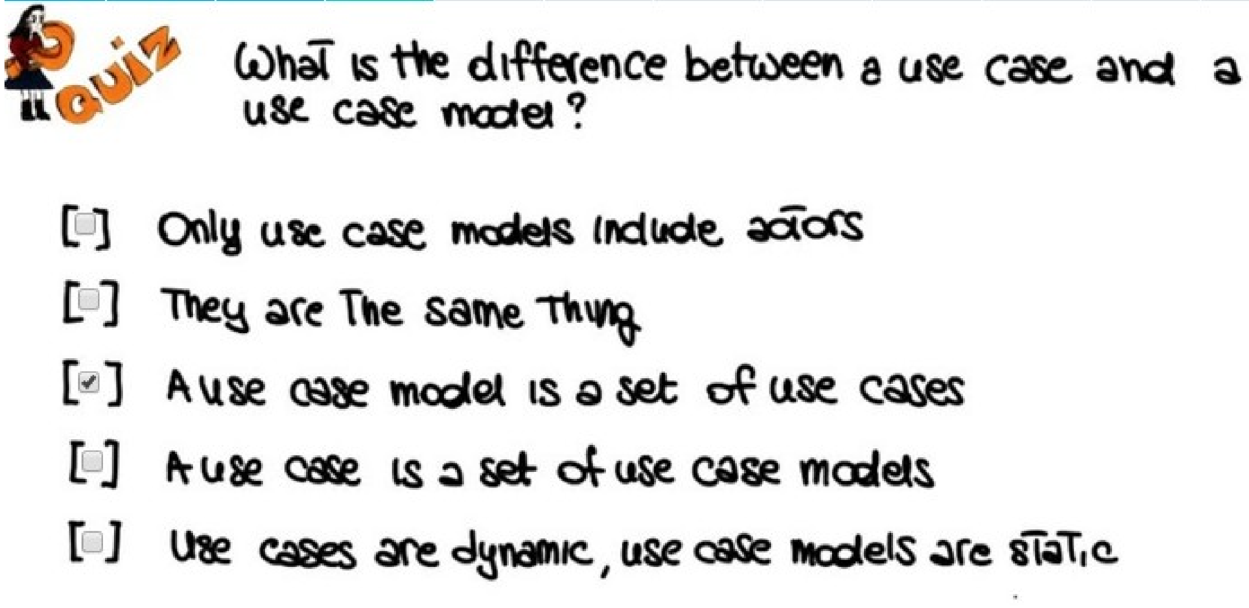
4. Before doing that though, let's have a quiz to check whether you remember the basics of UML. Since we're going to talk about use cases, I want to ask you, what is the difference between a use case and a use case model. So here are the possible answers, and you can mark more than one. First one is that only use case models include actors. The second is that they are the same thing. The third one is that they use case model is a set of use cases. The fourth one says that a use case is a set of use case models.

Finally, the last one says that use cases are a dynamic representation, whereas use case models are a static representation of a system. So mark all the answers that you think are correct.

5. And the correct answer, in this case it's only one, is that a use case model is a set of use cases. So a use case model is simply a collection of use cases that represent different pieces of functionality for the system.

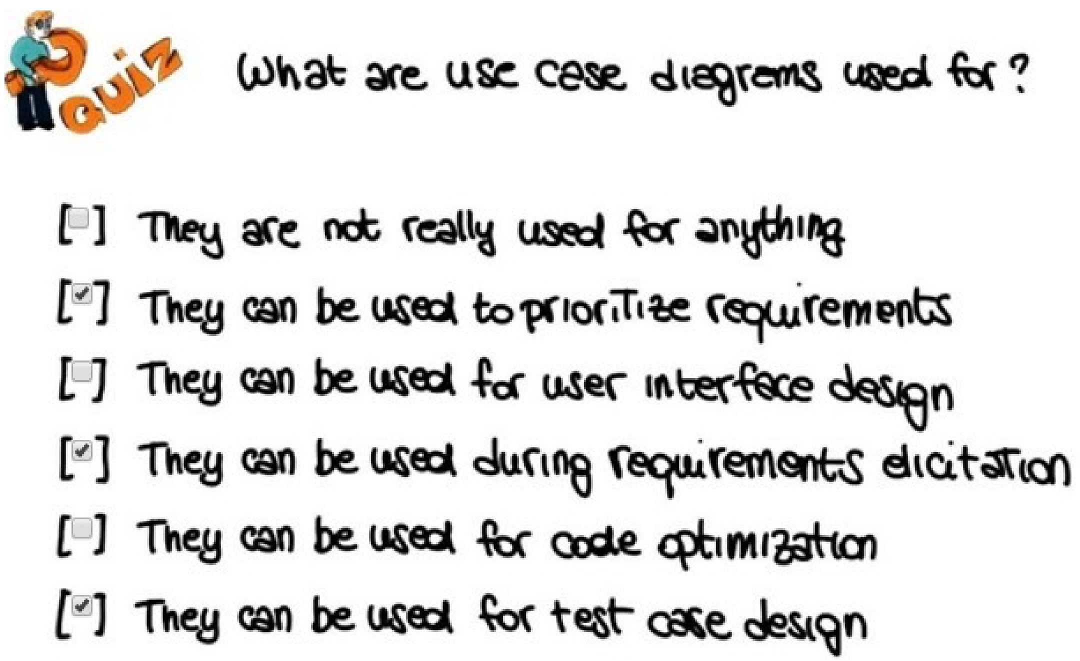
6. So, since we are talking about use case diagrams, I also want to ask you, what are use case diagrams used for? So, why are they good for? Why are they useful within a software process? Also in this case, I'm providing several possible answers. They are not really used for anything. Or, they can be used to prioritize requirements. Or, maybe they can be used for user interface design. They can be used during requirements elicitation. They can be used for code optimization. And finally, they can be used for test case design. Also, in this case, I would like you to mark all the answers that you think are correct.

7. In this case there are multiple, correct answers. So you should have marked several of those. So let's go through the list. Well this is definitely not true. They are used for something. The second answer, is a correct one, because you can order, the used cases that you planned to realize, according to your prioritization criteria. So basically what you're doing you're prioritizing either in terms of functionality. So you, you can decide which piece of functionality you want to realize first in your system. Or you can also prioritize based on the actors involved. Maybe there are some actors, maybe there are some user roles that you want to support before others, and we'll see some examples of that. The next correct question is that they can be used for requirements elicitation. Why? Well because used cases express what the system is supposed to do for each user. They're an ideal way to collect, represent, and check functional requirements. And we'll also get back to this. And finally, used cases can definitely be used for test case design. So why is that? Because each used case represents a scenario of interaction between users and the system. So testers can very naturally construct these cases based on used cases. And in addition, and most importantly, they can do that even in the absence of code that realizes a used case. So they can do it as soon as they have the requirements. They don't have to wait until the code is ready. So this is not very a important point. So you can have your test cases ready even before writing the code. And now for completeness. Even though this is not listed in the quiz. I also want to mention two additional uses for used cases. The first one is that used cases can be used to estimate effort as we will discuss in more detail in mini course four, when we talk about a giant software development. And they can also be used by customers to assess requirements. Which is another fundamentally important role of the used cases. They provide a common language between the customers and the developers which makes it easier to collect the right requirements.



What is the difference between a use case and a use case model?

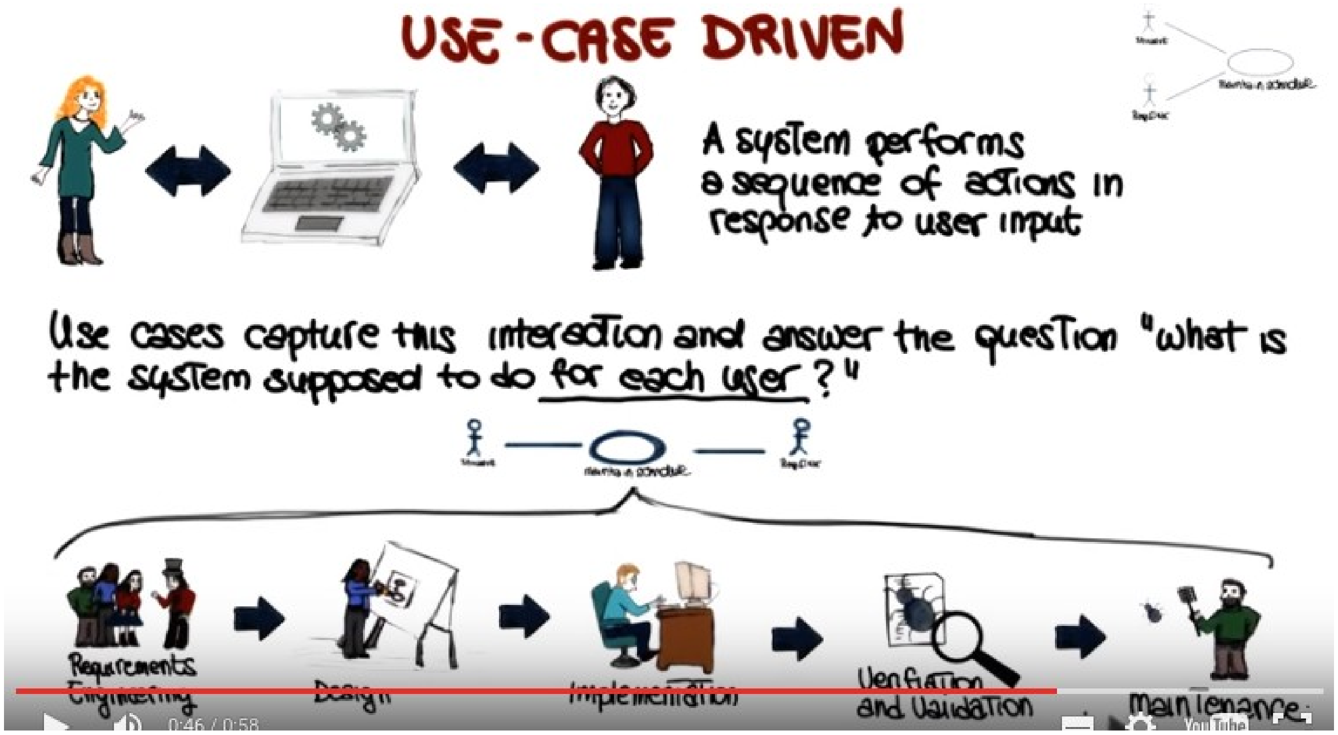
- [ ] Only use case models include actors
- [ ] They are the same thing
- [✓] A use case model is a set of use cases
- [ ] A use case is a set of use case models
- [ ] Use cases are dynamic, use case models are static



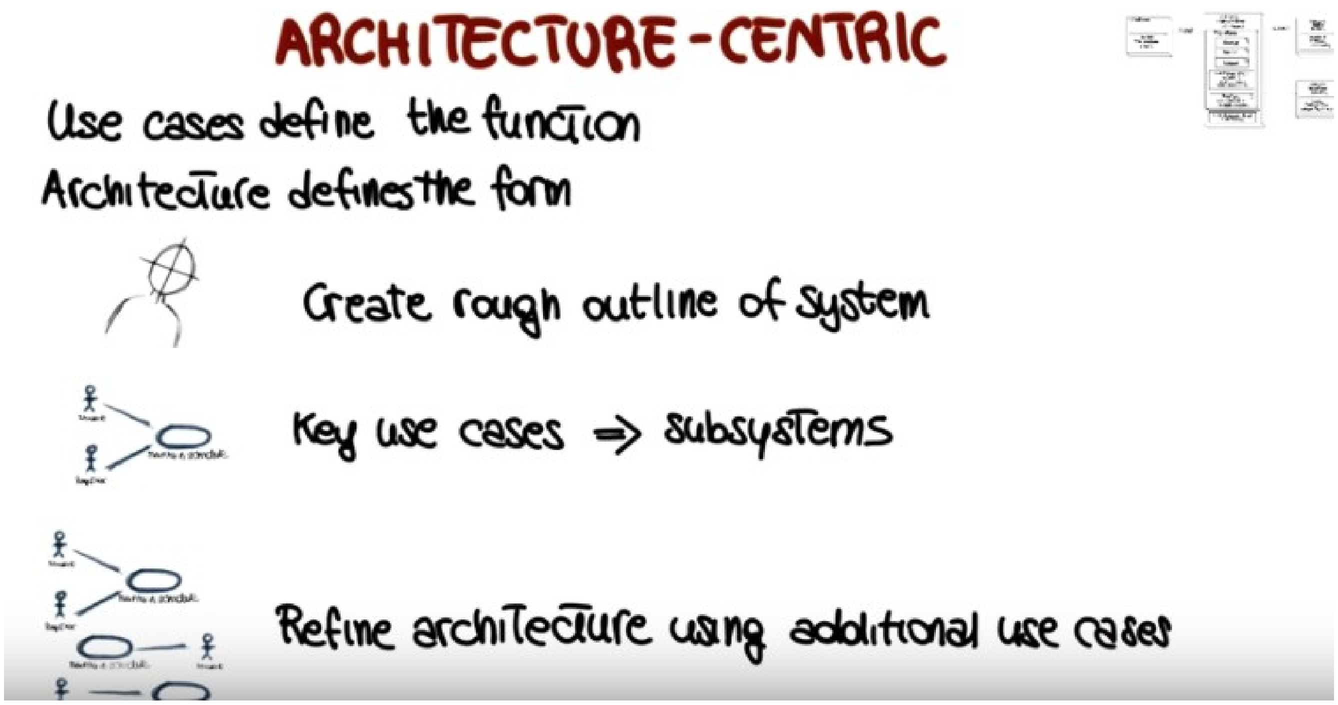
What are use case diagrams used for?

- [ ] They are not really used for anything
- [✓] They can be used to prioritize requirements
- [ ] They can be used for user interface design
- [✓] They can be used during requirements elicitation
- [ ] They can be used for code optimization
- [✓] They can be used for test case design

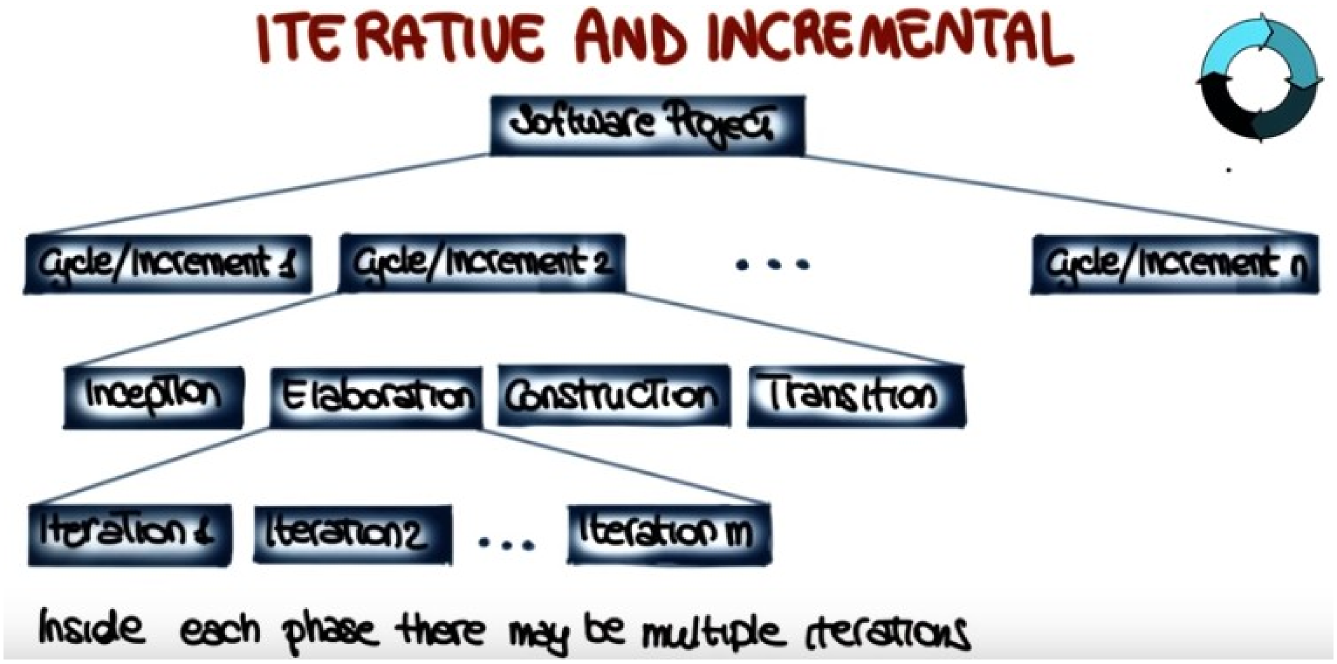
8. Now let's go back to the distinguishing aspects of RUP, starting from the first one. That is, that the rational unified process is use-case driven. So let's see what that means. Generally speaking, we can see a system as something that performs a sequence of actions in response to user inputs. So the user submits some requests, or requests some functionality, and the system responds to those requests. Use cases, as we just said, capture exactly this interaction and answer the question, what is the system supposed to do for each user? So, this is a very important point. So they can represent what a system can do for each of the different types of users of the system. For this reason, and as we will see in more detail, in the rational unified process, use cases are a central element of the whole development life cycle. From requirements engineering, all the way through the process until testing and maintenance. So, once more, use cases are used to support and help each one of these phases in the rational unified process.


USE-CASE DRIVEN
A system performs a sequence of actions in response to user input
Use cases capture this interaction and answer the question "what is the system supposed to do for each user?"
Requirements Engineering
Design
Implementation
Verification and Validation
Maintenance
0:46 / 0:58

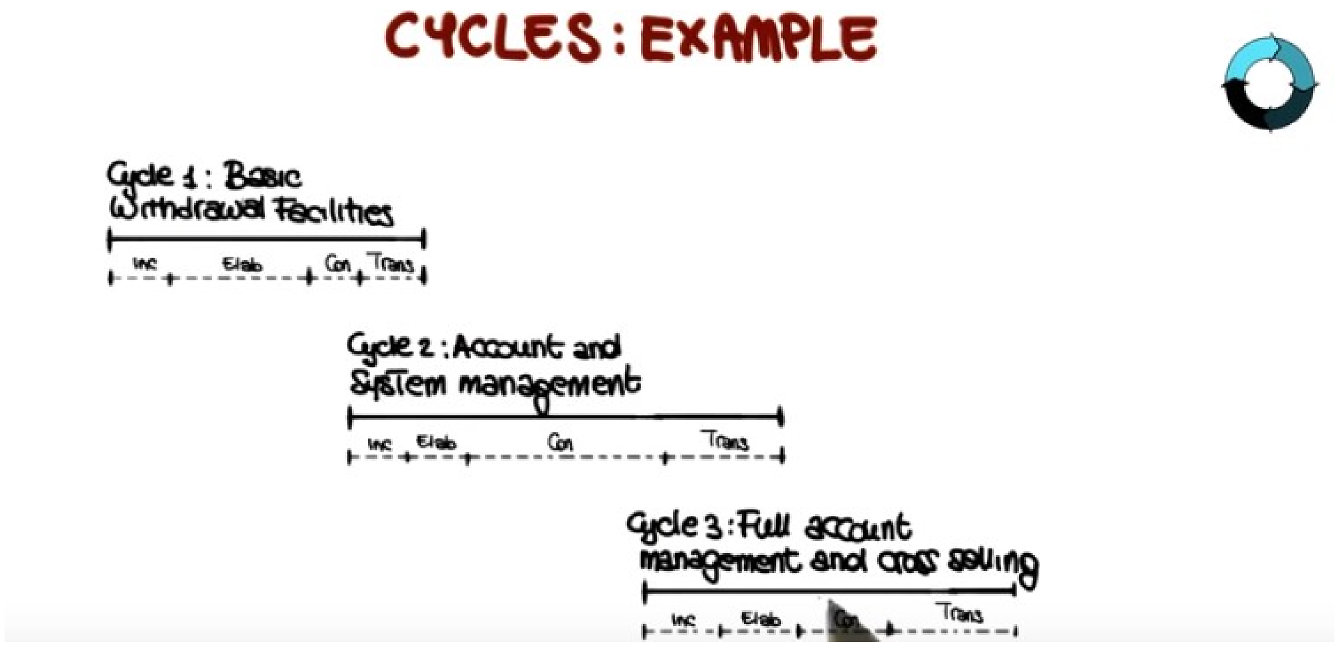
9. The second distinguishing aspect of RUP is that it is architecture-centric. As we saw in the first lesson of this mini-course, a soft architecture is a view of the entire system that represents all high level principal design decisions. Another way to see this is by saying that use cases define the function of the system, where as architecture defines the form of the system. Use cases give the functionality, architecture tells you how the system should be structured to provide the functionality. So how do we define a soft architecture in the ration of a unified process. Also in this case this happens through a sort of a iterative process. We start by creating a rough outline of the system. And in this case we do it independently from the functionality. So this is just the general structure of the system. For example, we model aspects such as the platform on which the system will run, the overall style. For example, whether it's a client server or a peer to peer system and so on. We then use the key use cases in our use case diagram to define the main subsystems of my architecture. For example, in the case of a banking IT system, one of these subsystems might be the withdrawal system. So what will happen in that case is that we will have some use case that refers to the withdrawal activity, and by analyzing that use case, we'll realize that we need a subsystem that implements that piece of functionality. So again, we use the key use cases to identify and define the key subsystems for my architecture. So once we have that we keep refining the architecture by using additional use cases. So considering more and more pieces of functionality that will help us to refine the architecture of the system and also leveraging our increasing understand of the system that we're modeling. And this will continue until we are happy with the architecture that we define.


ARCHITECTURE - CENTRIC
Use cases define the function
Architecture defines the form
Create rough outline of system
Key use cases ⇒ subsystems
Refine architecture using additional use cases

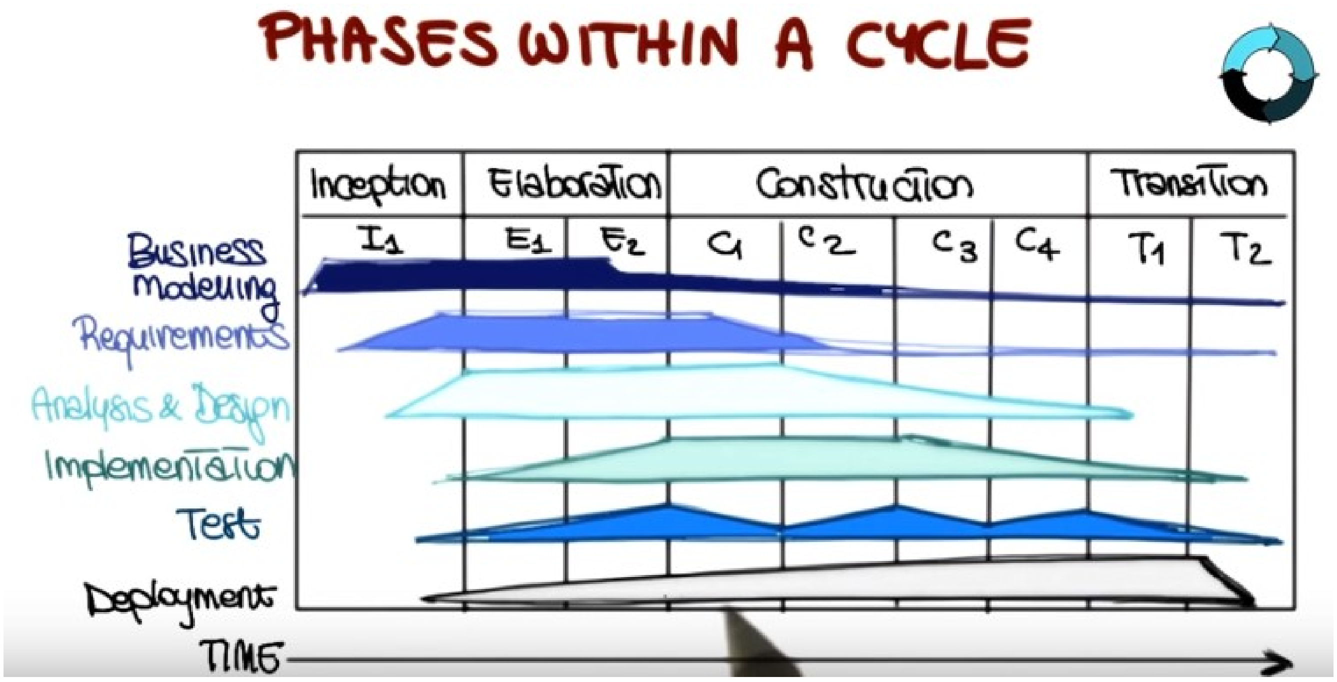
10. We just saw two of the three distinguishing aspects of the rational unified process. The fact that it is used case driven and the fact that it is architecture centric. The third and final distinguished aspect of R.U.P. is that it is iterative and incremental. So let's see what that means by considering the lifetime of a software project. Basically, the lifetime of a rational unified process consists of a series of cycles, such as the ones that are represented here. Cycle one, cycle two, through cycle n. And as you can see, these cycles can also be called increments. And each one of the cycles involves all of the main phases of software development. In addition, each cycle results in a product release which can be internal or external. More precisely, each cycle terminates with a product release that includes a complete set of artifacts for the project. That means code, manuals, use cases, no functional specification, desk cases, and so on. So, I've just said, that each cycle involves all of the main phases of software development. Specifically, each cycle is further divided in four phases. Inception, elaboration, construction and transition. In a minute, we will look at each one of these phases in detail and see how they relate to the traditional activities of software development. Before that, I want to mention the last level of this iterations, which happens within these individual phases More precisely, inside each of these phases, there might be multiple iterations. So what are these iterations? Well, basically, each iteration corresponds to a group of use cases that are selected so as to deal with the most important risks first. So if you have a set of use cases that you're considering, which means that you have a set of features that you need to implement, you will select for each iteration the most risky ones that you still haven't realized, and realize them in that iteration. And then continue in the following iterations with less and less risky ones. So basically what happens in the end is that essentially each iteration extends the functionality beyond the previous iteration.



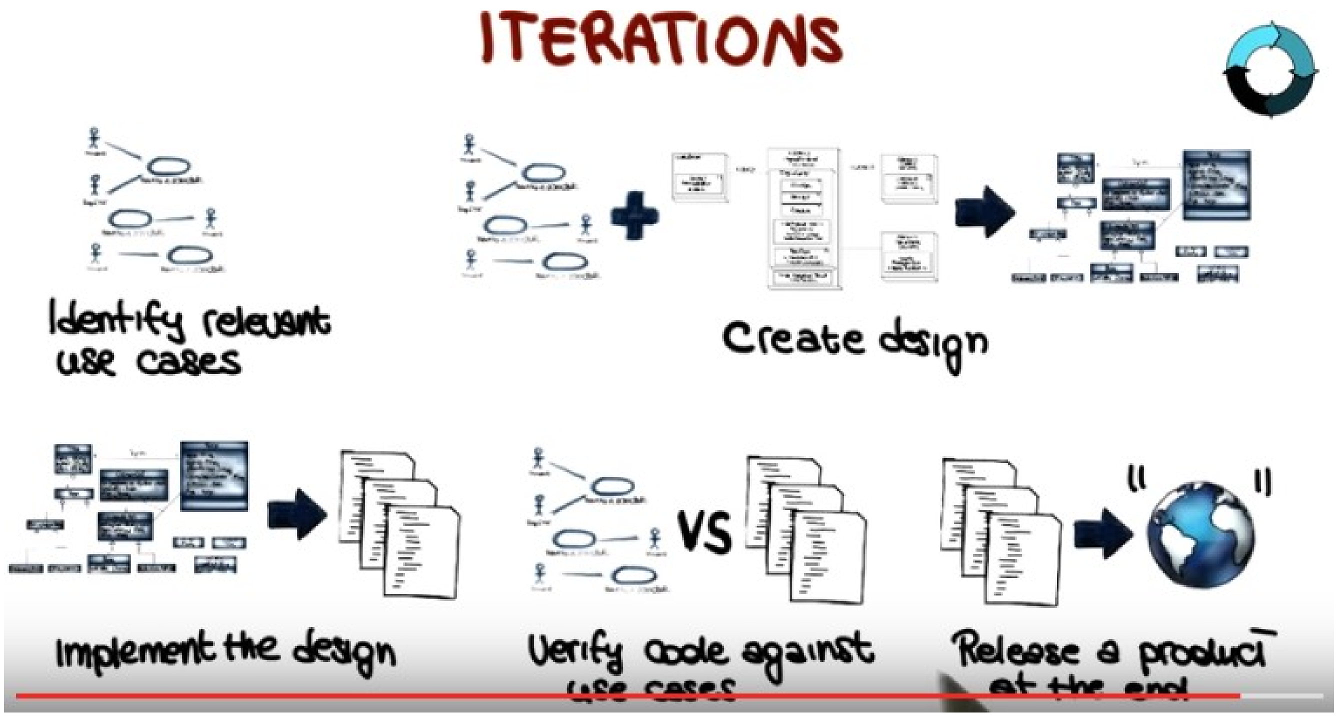
11. To make this a little more concrete, let's look at an example involving cycles, phases, and iterations. Let's assume that we have to develop a banking IT system. The first possible cycle for such a system could be one in which we implement the basic withdrawal facilities. What this means is that, at the end of this cycle, there will be the release of the product that implements this piece of functionality. But notice that this will not be the only product release because within the cycle, we will perform also the four phases that we mentioned before, inception, elaboration, construction, and transition. And within each of these phases, we might have multiple iterations. And at the end of each iteration, we will also have a product release. Which in this case, will be an internal one. As you can see, the iterative nature is really inherent in the unified rational process. So, now let's clean up here, and let's see what some other possible cycles could be for our banking IT system. Here, I'm showing two possible additional ones. The first one, cycle two, which will develop the account and system management. And the third one, cycle three, which will develop the full account management and cross selling. Similarly to cycle one, also these cycles will produce a product, both at the end of the cycle, and within the cycle in the different phases. And there's a few more things to note. So the first one, is that each cycle focuses on a different part of the system. So what you will do, when you use the rational unified process, you will select a subset of use cases that you want to realize within your cycle. And the final product for that cycle, will be a product that realizes those use cases. This is the first aspect. The second one, is that these cycles, as you can see, are slightly overlapping. So it is not the case that you finish a cycle, and then you start the next one. So there is a little bit of overlap among cycles, and we'll, we'll talk about that more. And finally, I want to stress one more that each cycle contains four phases, and each one of these phases might be further splayed in iterations. So that's kind of a high level view of how the whole process will work.


CYCLES: EXAMPLE
Cycle 1: Basic Withdrawal Facilities
Inc
Elab
Con
Trans
Cycle 2: Account and System management
Inc
Elab
Con
Trans
Cycle 3: Full account management and cross selling
Inc
Elab
Con
Trans

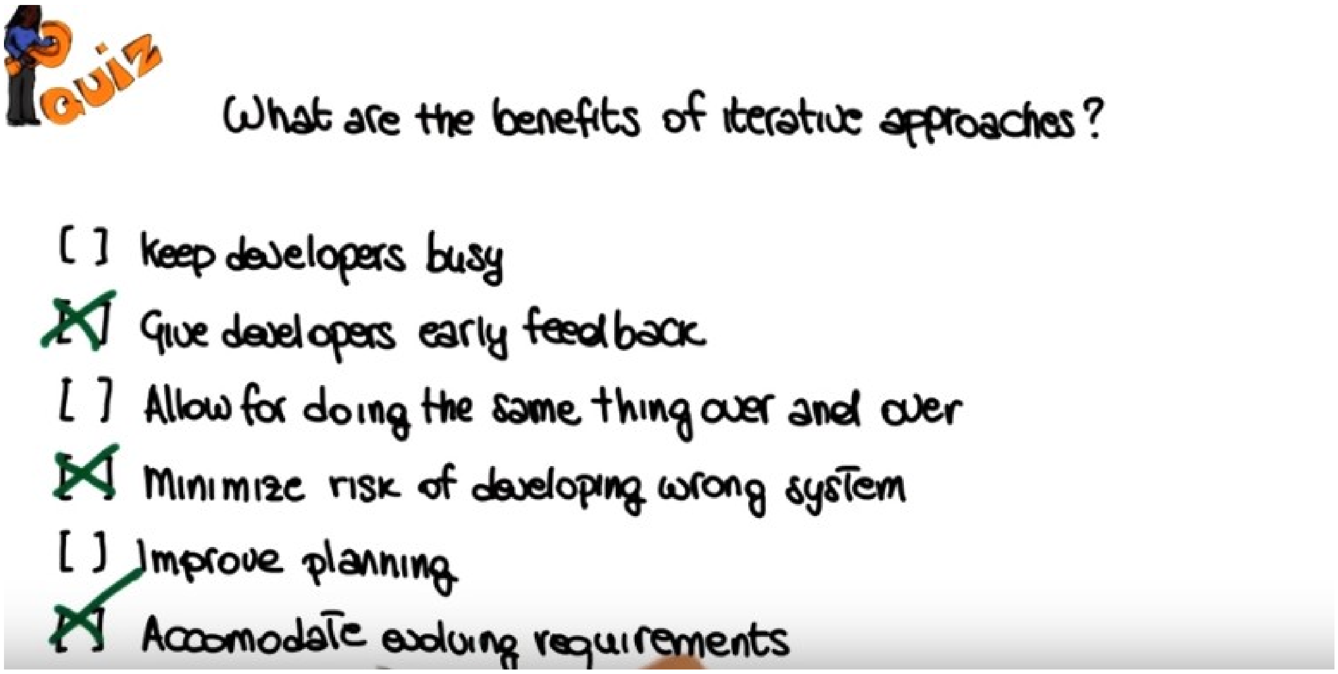
12. Now let's go back to the phases within a cycle. because I want to show you how they relate to traditional activities of software development. Because this is the first time that we talk about inception, elaboration, construction and transition. So we will know what they mean, in terms of the traditional software development. So I'm going to first discuss these relations and then I'm going to talk about each individual phase in further detail. So I'm going to start by representing the four RUP phases here with possible internal iterations. I1, E1 and E2, C1, C2, and so on. And just as a reference, this is the way in which time will progress. So we will start with inception and we will finish with transition. So what I'm want to do now is to show the actual, traditional, software development activities here on the left. And I also want to show you, using this diagram, how these activities are actually performed in each of the RUP phases. So, let's see what this representation means. Basically, what I'm showing here, is that requirements engineering actually starts in the inception phase. So, you can see the height of this bar as the amount of effort devoted to this activity in this specific phase. So you can see that requirements engineering starts in inception phase, is mostly performed in the elaboration phase, and then it continues, although to a lesser extent, throughout all phases up until the end of the transition. But the bulk is really performed here in the elaboration phase. Similarly, if we consider analysis and design, we can see that analysis and design are mainly performed in the elaboration phase. But a considerable amount of it also continues in the construction phase, and then it kind of phases out. So there's very little of that done in the transition phase. Looking now at implementation, you can see that the implementation happens mostly in the construction phase, which is, unsurprisingly, the phase that is mostly concerned with actual code development, as we will see in a minute. Testing, on the other hand, is performed throughout most phases, with, peaks in some specific point, for example, at the end of some iterations, like here and here. To conclude, we have the business modeling activity that happens mainly in the inception and a little bit in the elaboration phase and the deployment activity which happens a little bit throughout, but the bulk of it is really in the transition phase, which is actually the phase that has to do with deployment and then maintenance. So I hope this kind of high level view give you a better understanding of what is the mapping between these new phases and, the typical software development activities that we are more familiar with. So to further this understanding, later in the lesson, I'm also going to talk about these specific phases individually. First, however, I want to spend a little more time discussing what happens inside each one of these iterations, just to make sure that we are all understand what an iteration is exactly.



13. So what happens, exactly, within an iteration? In almost every iteration, developers perform the following activities. So they identify which pieces of functionality this iteration will develop, will implement. After doing that, they will create a design, for the considered use cases, and they will do that guided by the chosen architecture. So the set of use cases plus the architectural guidelines will result in a design for the selected use cases. Once the design is defined, then the developers will implement the design, which will result in a set of software components. They will then verify the components against the use cases to make sure that the components satisfy the use cases, they suitably realize the use cases. And they will do that through testing or some other verification and validation activity. Finally, after verifying that the code actually implements the use cases, they will release a product, which also represent the end of the iteration. And notice that what I put here is the known icon for the world, in double quotes. Because in many cases the release will be just an internal release or maybe a release that will just go to some of the stakeholders so that they can provide feedback on that. Okay. So it doesn't have to be an external release. It doesn't have to be a release to the world. But it is, nevertheless, a release of a software product. .


ITERATIONS
Identify relevant use cases
Create design
Implement the design
VS
Verify code against use cases
Release a product at the end

14. So now, since we're talking about the incremental and iterative nature of the Rational Unified Process, let's have a quiz on the benefits of iterative approaches. So like for you to tell me what are these benefits. Is one benefit the fact that iterative approaches keep developers busy or maybe that they give developers early feedback, that they allow for doing the same thing over and over in an iterative way. Maybe they also minimize the risk of developing their own system. Can they be used to improve planning, or is it the benefit that they accommodate evolving requirements. Also, in this case, I would like for you to check all the answers that you think are correct.



What are the benefits of iterative approaches?

- [ ] Keep developers busy
- [X] Give developers early feedback
- [ ] Allow for doing the same thing over and over
- [X] Minimize risk of developing wrong system
- [ ] Improve planning
- [X] Accomodate evolving requirements

15. Okay so let's look at the first one. Well I don't think that the fact of keeping developers busy is really one of the highlights or the main benefits of iterative approaches. Developers are really busy without any need for additional help. So I will just not mark this one. The second one conversely is definitely one of the advantages of iterative approaches. So the fact that iterative approaches give the developers a early feedback, is a great advantage which has in turn additional advantages. For example, it increases the project tempo, so it gives the developers not easy but more focused. It's easier to be focused when you have a short term you know, deadline, or a short term goal rather than a release that is planned in six months or even later. Another advantage of this early feedback is the fact that developers are rewarded for their efforts so, there is sort of an immediate rewards because you can see the results of your effort instead of having to wait a long time to see such results. And last, but not least the fact of getting early feedback also minimizes the risks of developing the wrong system. So why is that? Well because getting early feedback will also allow us to find out whether we're going in the wrong direction early in the development rather than at the end. And therefore, will minimize this risk. Going back to the previous question, yeah, I don't think that, you know, doing the same thing over and over is a great advantage. And in fact, iterative approaches do not do the same thing over and over. So they keep iterating, but they keep augmenting the amount of functionality in the system. They don't just repeat the same thing. As for improving planning, actually improving planning is not really a strength of these approaches, because sometimes the number of iterations is hard to predict, so it's hard to do a natural planning when you are using an iterative approach. So finally, are iterative approaches good for accomodating evolving requirements? Most definitely. First, iterative approaches, and in particular, the one that we're discussing consider requirements incrementally, so they can better incorporate your requirements. So if there are new requirements, it's easier to accommodate them using an iterative approach. Second, these approaches realize a few requirements at a time. Something from the most risky ones, as we said. So any problem with those risky requirements will be discovered early, and suitable course corrections could be taken. So in case you still have doubts about alternative approaches, it might be worth it to go back to mini course number one, lesson number two to discuss the life cycle models. Because we talk about alternative approaches and their advantages and their characteristics there in some detail.

16. Let's talk a little bit more about phases. The rational unified process phases are fundamental aspects of this process and which just touched on them so we just give a quick overview. And I want to look at these phases in a little more detail. So, what I'm going to do is, for each phase, I'm going to discuss what it is, what it produces and how is the result of the phase suppose to be,. Assessed, and what are the consequences of this assessment. So let's start with the first phase, the inception phase. The first phase goes from the idea of the product to the vision of the end product. What this involves is basically to delimiting the project scope. And making the business case for the product presented. Why is it worth doing? What are the success criteria? What are the main risks? What resources will be needed? And so on, specifically these phases answer three main questions. The first one is, what are the major users or actors, to use the UML terminology. And what will the system do for them? To answer this, these phases produce a simplified use-case model where only a few use-cases are represented and described. So this is a sort of initial use-case model. The second question is about the architecture, what could be an architecture for the system? So in this phase we will normally also develop a tentative architecture. So an initial architecture that describes the most crucial subsystems. Finally this phase also answers the question, what is the plan and how much will it cost? To answer this question. This phase will identify the main risks for the project and also produce a rough plan with estimates for resources, initial planning for the phases and dates and milestones. Specifically, the inception phase generates several deliverables. It is very important that you pay attention so that you understand what this deliberate approach are. Starting from the first one, which is the vision document. And this is a document that

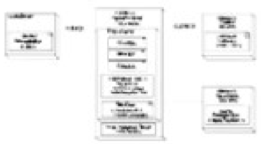
provides a general vision of the core projects requirements, key features and main constraints. Together with this, the inception phase also produces an initial use case model, as I just mentioned. So this is a use case model that includes an initial set of use cases, and then will be later refined. Two additional variables are the initial project glossary, which describes the main terms, using the project and their meaning, and the initial business case which includes business context. And success criteria. Yet another deliverable for the inception phase is the initial project plan, which shows the phases, iterations, roles of the participants, schedule and initial estimates. In addition, the inception phase also produces a risk assessment document, which describes the main risks and counters measures for this risk. Finally, and this is an optional deliverable, in the sense that it, it might or might not be produced, depending on the specific project. As part of the inception phase we might also generate 1 or more prototypes. For example, we might develop prototypes to address some specific risks that we have identified or to show some specific aspect of the system of which we are unsure to the stakeholders. So basically all the typical users of prototypes that we discussed before. So when we're done with the inception phase we hit the first milestone for the cycle we are currently performing. And so there are some evaluation criteria that will tell us whether we can consider the inception phase concluded or not. And the first of this criteria is stakeholder concurrence, which means that all the stakeholders must agree on the scope, definition, and cost schedule estimates for the projects. The second criteria needs requirements understanding, out of the initial primary use cases that we have identified so far, the right one for our system. And other criteria is the credibility of the cost schedule estimates, the priorities, defined the risks identifies and the countermeasures for those risks, and the development process that we're following. Finally, in the case we produce prototypes as part of the inceptional phase, this will also be evaluated and assessed to judge the overall outcome of the phase. So what happens if the project fails to pass this milestone? So if the outcome of the inception phase is considered to be inadequate with respect to one or more of these criteria. Well at this point, since we're kind of an initial phase of the cycle the project may be cancelled or considerably re-thought. So to summarize all of these in one sentence the Inception Phase is the phase in which we produce. Then you shall vision, used case model, project plan, risk assessment and possibly, prototypes for the project. And we have to make sure, that all of this deliverables satisfy a set of criteria, so that we can continue on the project. And otherwise, we'll either cancel the project or rethink its scope, or other aspects of it.

INCEPTION PHASE

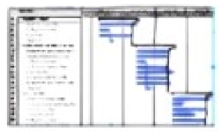
From idea to vision of end product
Delimit project scope
Business case for the product presented

What are the major users (actors) and what will the system do for them? Simplified use-case model

What could be an architecture for the system? Tentative architecture



What is the plan and how much it will cost?
Main risks identified, rough planning

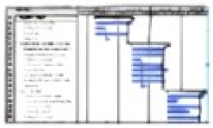


INCEPTION PHASE

Vision document with general vision of the core project's requirements, key features, and main constraints

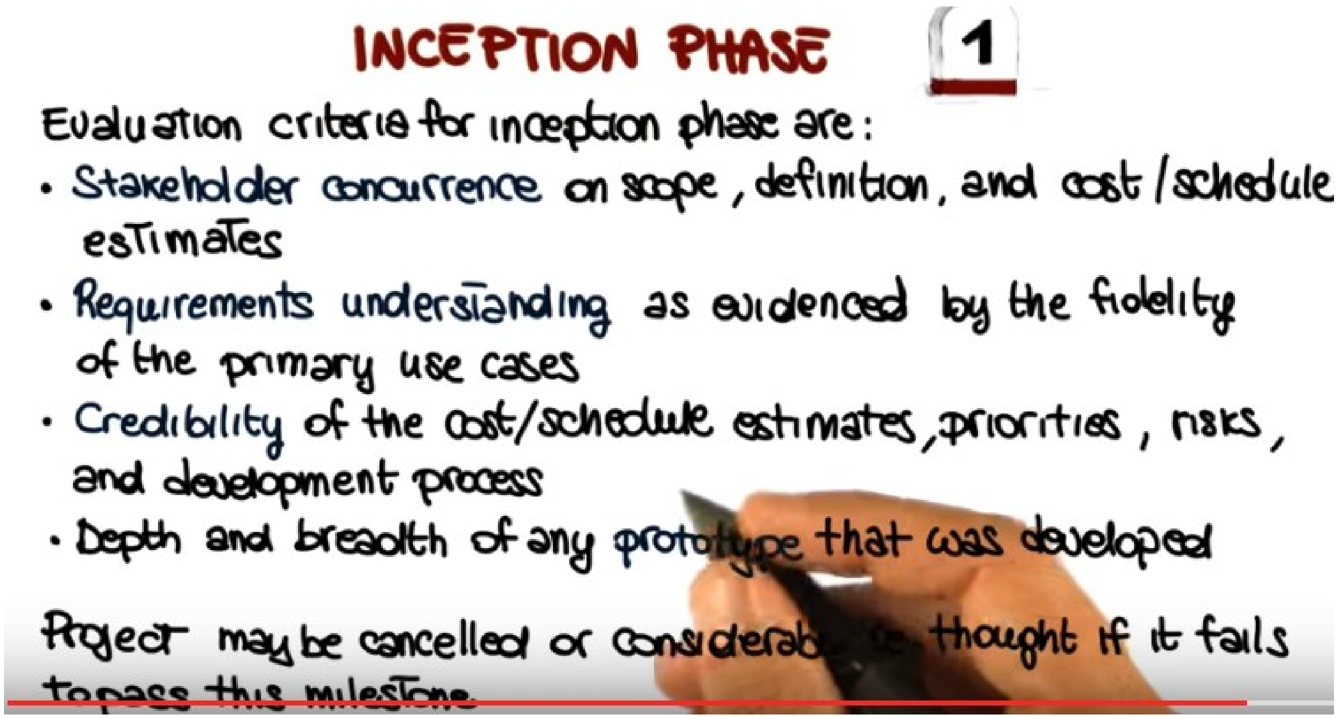
Initial use-case model

Initial project glossary
Initial business case



Initial project plan and risk assessment

[optionally] One or more prototypes



INCEPTION PHASE 1

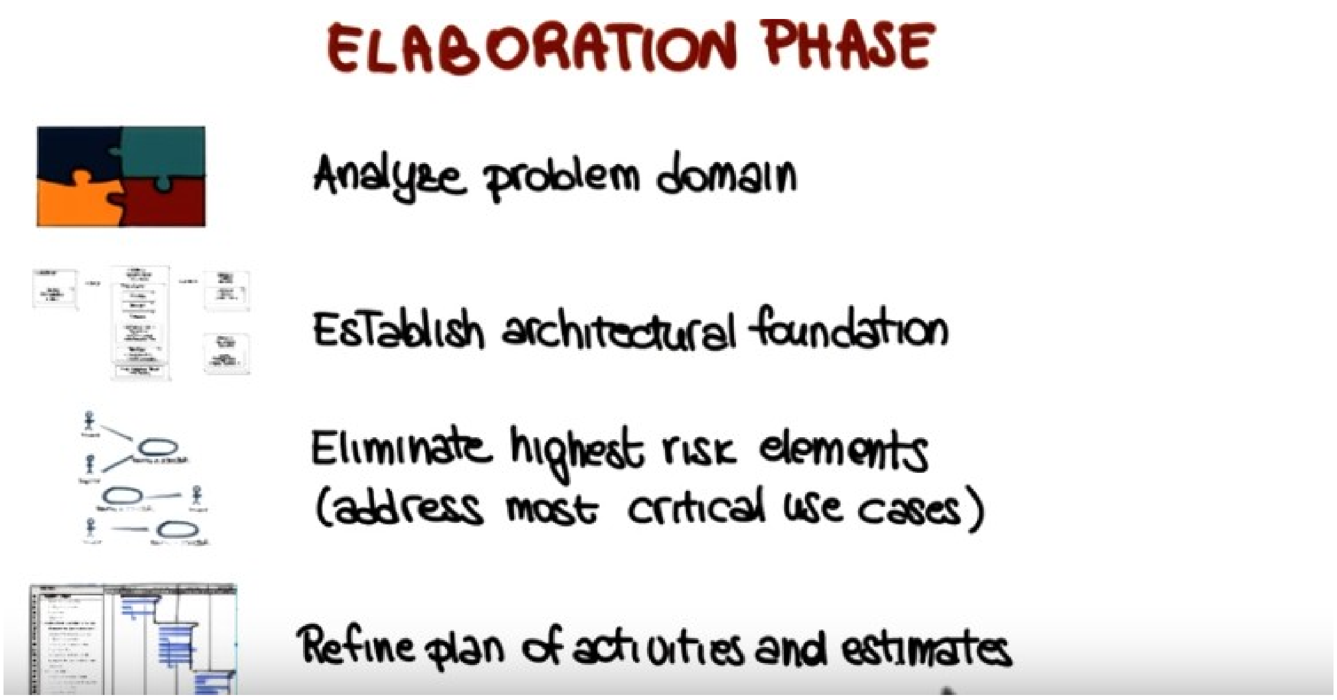
Evaluation criteria for inception phase are:

- Stakeholder concurrence on scope, definition, and cost/schedule estimates
- Requirements understanding as evidenced by the fidelity of the primary use cases
- Credibility of the cost/schedule estimates, priorities, risks, and development process
- Depth and breadth of any prototype that was developed

Project may be cancelled or considerable re-thought if it fails to pass this milestone

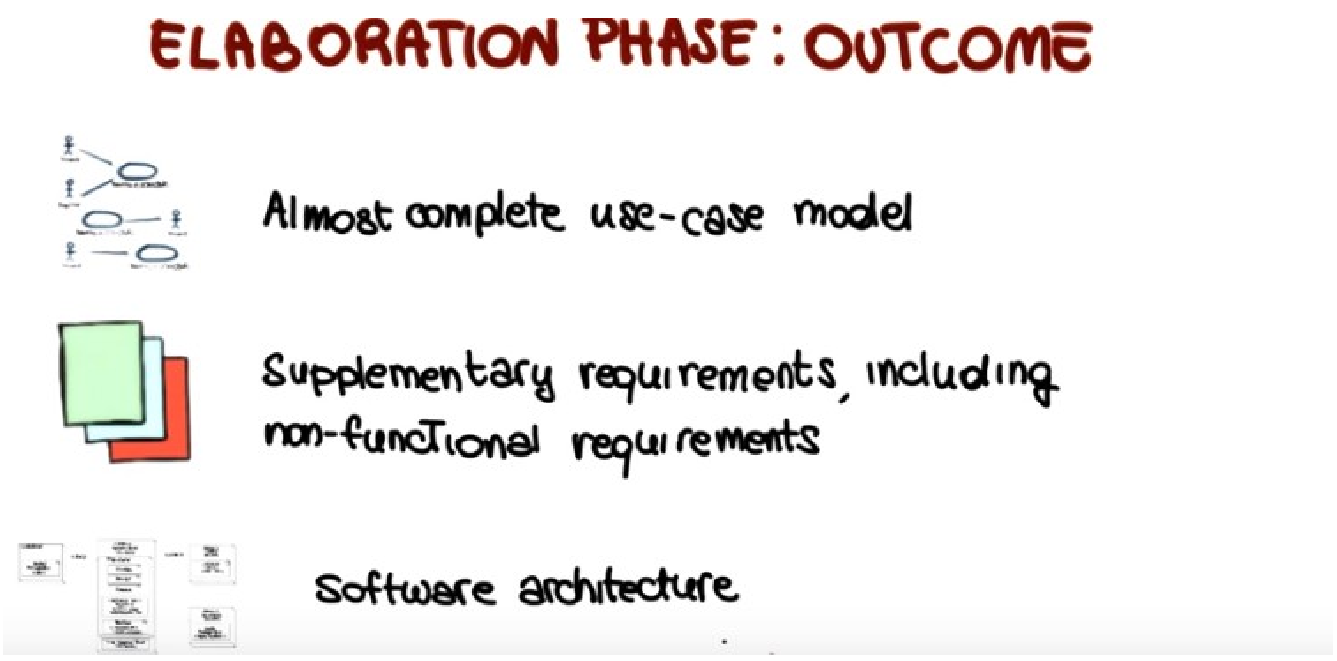
17. Now that we've discussed the inception phase, let's move on to the second phase of RUP, which is the elaboration phase. And there are four main goals and activities for the elaboration phase. Analyzing the problem domain to get a better understanding of the domain. Establishing a solid architectural foundation for the project. Eliminating the highest risk elements which basically means addressing the most critical use-cases. And finally, refine the plan of activities and estimates of resources to complete the project. The outcome of the elaboration phase reflects these activities and also in this case produces several artifacts. The first one is an almost complete use-case model with all use-cases and actors identified and most case-use descriptions developed. As part of this phase we also identify a set of what we called supplementary requirements. So these are basically all the requirements that are not associated with a use-case. And these sets includes in particular all non-functional requirements such as security, reliability, maintainability and so on. So all the ones that are relevant for the system that you're developing. We mentioned before that the software architecture is developed in an incremental way, so it's not created at once. And this is exactly what happens in the elaboration phase, that we take the initial architecture that was defined in the inception phase and we refine it until we get to a software architecture which is complete. And that is part of the deliverables for this phase. And the list continues, so let me make some room. In addition to producing a complete architecture for our system, in the elaboration phase we also define the lower-level design for the system. And, therefore, as part of this phase, we produce as deliverables a design model, and together with that, a complete set of test cases, and an executable prototype. We also produce a revised project plan. Now that we have more information about the project we can refine the various estimates and the various pieces of information in the project plan. And also an updated risk assessment document. Finally, in this phase we also generate a preliminary user manual that describes to the users how the system can be used and should be used. So now let's see what are the evaluation criteria for the elaboration phase which is our second milestone. So I'm just going to list them here. The first one is whether the vision and the architecture are stable or they're still changing so they will converge into a final complete vision for the system. Does the prototype show that the major risks that we have identified have been resolved or at least addressed in this phase? Is the project plan sufficiently detailed and accurate? Do all stakeholders agree that the vision can be achieved with the current plan? Is the actual resource expenditure versus the planned expenditure acceptable? So now we study consumer resources and therefore we can check

whether our estimates were correct. And also in this case the project might be cancelled or considerably reshaped if it fails to pass this milestone.



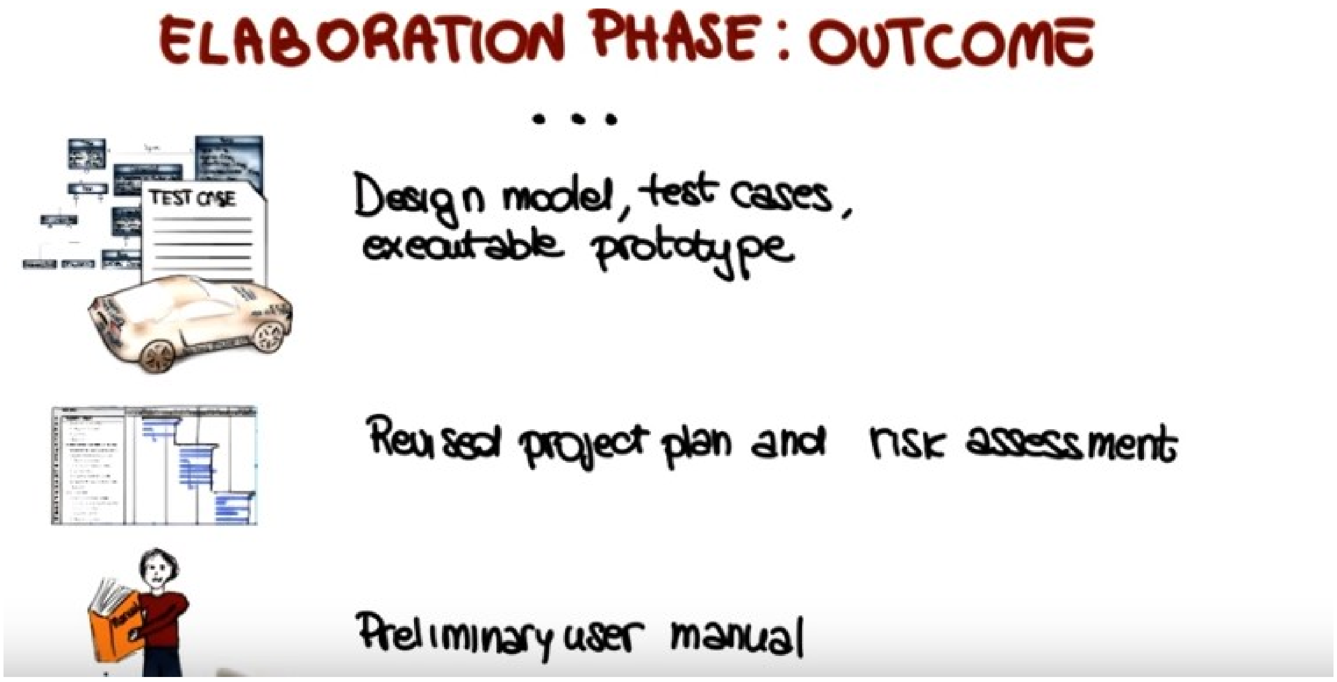
ELABORATION PHASE

- Analyze problem domain
- Establish architectural foundation
- Eliminate highest risk elements (address most critical use cases)
- Refine plan of activities and estimates



ELABORATION PHASE: OUTCOME

- Almost complete use-case model
- Supplementary requirements, including non-functional requirements
- Software architecture



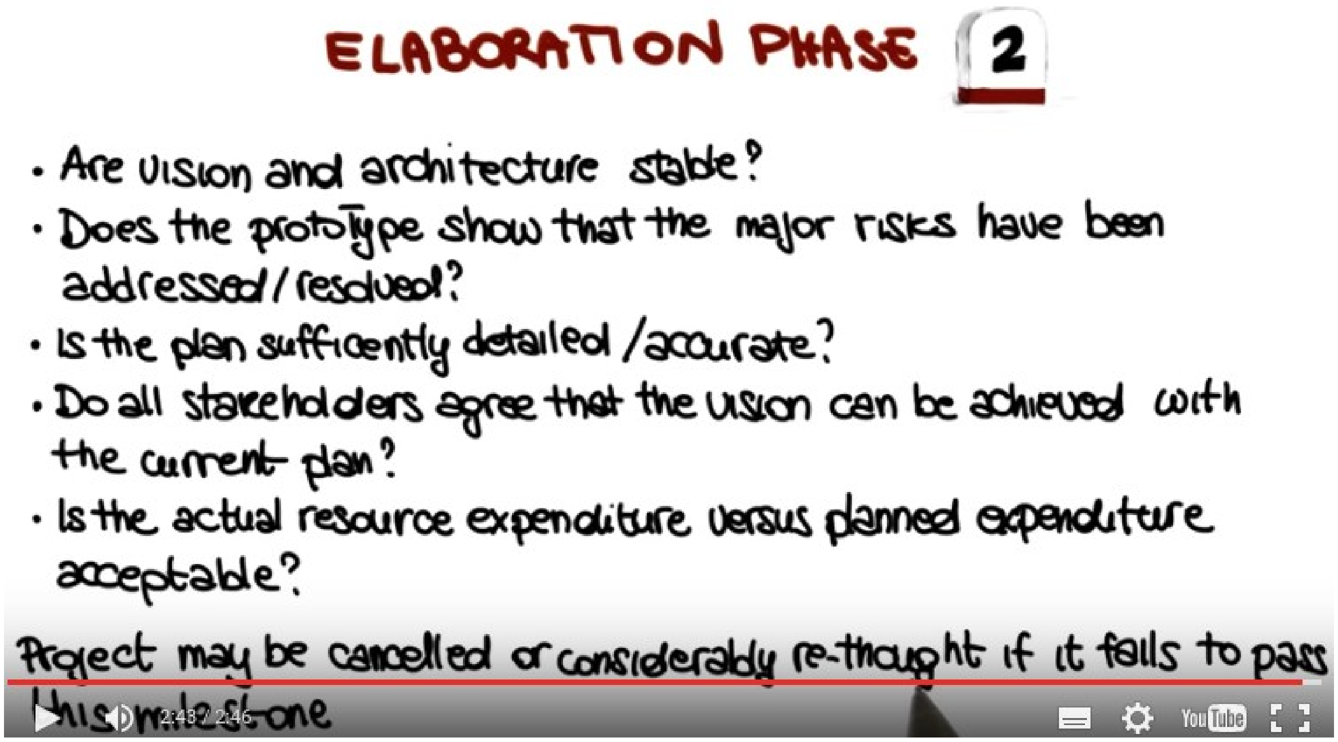
ELABORATION PHASE: OUTCOME

...

Design model, test cases, executable prototype

Revised project plan and risk assessment

Preliminary user manual



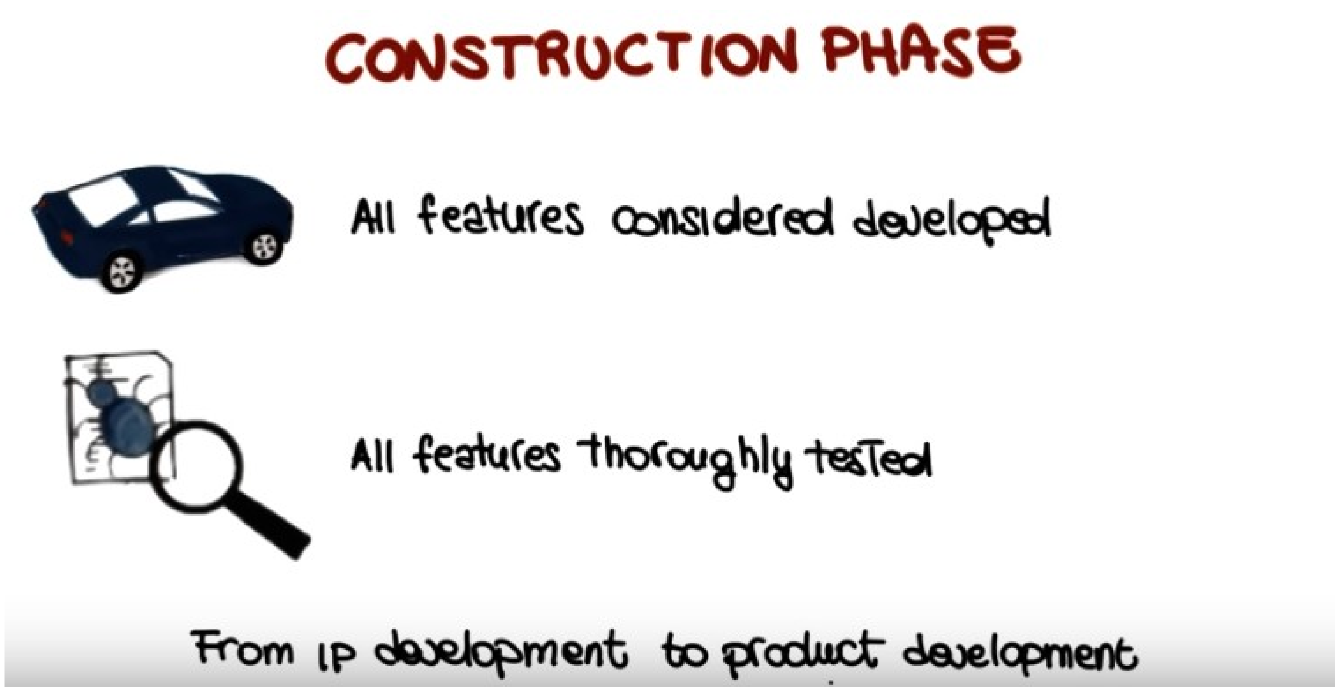
ELABORATION PHASE 2

- Are vision and architecture stable?
- Does the prototype show that the major risks have been addressed/resolved?
- Is the plan sufficiently detailed/accurate?
- Do all stakeholders agree that the vision can be achieved with the current plan?
- Is the actual resource expenditure versus planned expenditure acceptable?

Project may be cancelled or considerably re-thought if it fails to pass this milestone

18. So if the elaboration phase is successful, we then move to the construction phase, which is our third phase. And the construction phase is basically the phase in which most of the actual development occurs. In short, all the features considered are developed. So we'll continue with our car metaphor that we used for the prototype. And in this case we will have our complete car ready. Not only the features are developed but they're also thoroughly tested. So we have performed quality assurance. We have verified and validated the software, the system and we know that it works correctly. Or at least that it

works correctly as far as we can tell. So, in general, the construction phase is the phase in which there is a shift in emphasis from intellectual property development to product development. From ideas to products. So, what is the outcome of the construction phase? Well, basically the construction phrase produces a product that is ready to be deployed to the users. Specifically, the phase generates the following outcomes. First of all, at the end of this phase, all the use cases have been realized with traceability information. What does that mean? It means that not only all the functionality expressed by the use cases have been implemented, but also that we have traceability information from the use cases, to the different artifacts. So for example, we know which part of the design realizes which use case. We know which part of the implementation is related to a given use case. Which use cases were derived from a use case, and so on and so forth. And in this way we can trace our requirements throughout the system. Throughout the different artifacts that were developed during the software process. As we were saying, we also have complete software product here, which is integrated on all the needed platforms. Since the system, the software product, has to be thoroughly tested, we will also have a complete set of results for our tests. As part of this phase, we will also finalize the user manual, so you'll have a user manual ready to be provided to the users, and ready to be used. And finally, we will have a complete set of artifacts that include design documents, code test cases, and so on and so forth, so basically all of the artifacts that have been produced during the development process. So roughly speaking, we can consider the product that is produced at the end of this phase as a typical beta release. So in case you're not familiar with that, a beta release is an initial release normally meant for a selected subset of users. So it is something that is not quite yet ready for primetime, but almost. So let's see also in this case, without the evaluation criteria for the construction phase. So how do we assess, that this third milestone has been accomplished, successfully? We pretty much have a complete product ready to be shipped right? So the first question we want to ask is, whether the product is stable and mature enough to be deployed to users. At the end of this phase it has to be. Are the stakeholders ready for the transition into the user community? Are we ready to go from development to production? Are the actual resource expenditures versus the planned expenditures still acceptable? So what this means is that at this point we can really assess whether our estimates were accurate enough with respect to what we actually spent for the project up to this point. So unless we can answer in a positive way to all these questions, the transition might be postponed by one release. Because that means that we're still not ready to go to the market. We're still not ready to deploy our product.

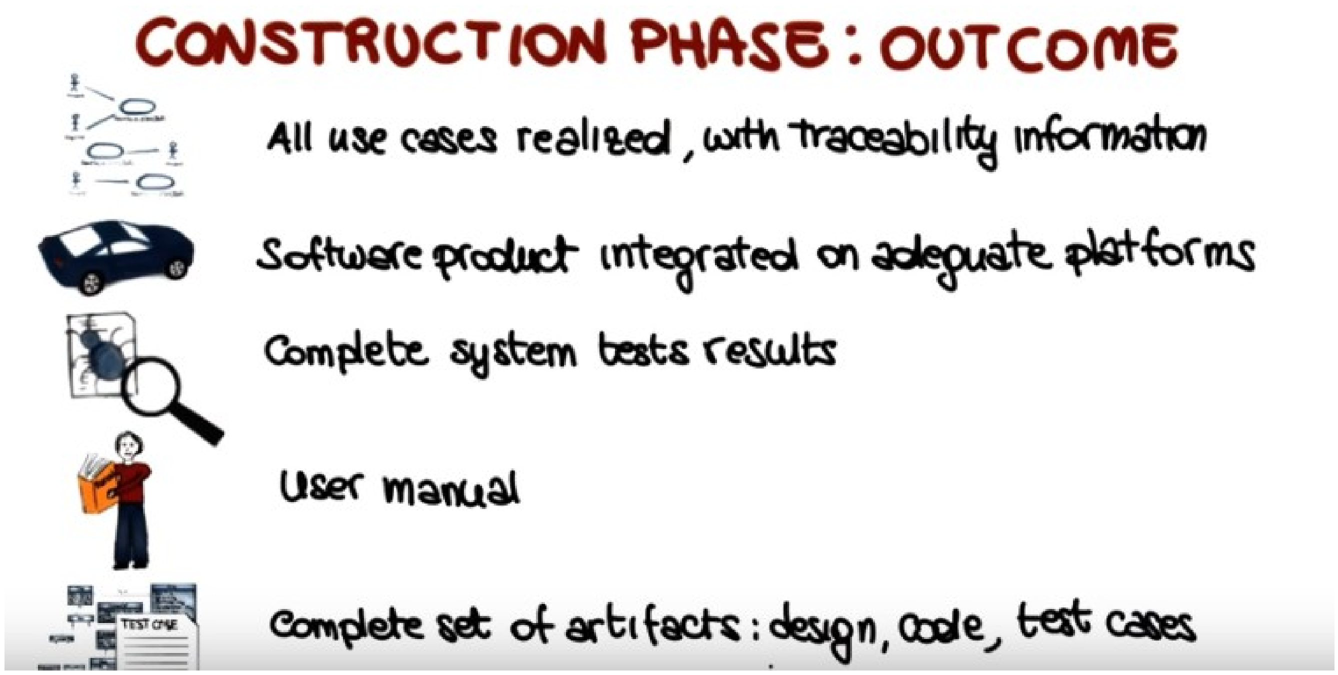


CONSTRUCTION PHASE

All features considered developed

All features thoroughly tested

From ip development to product development



CONSTRUCTION PHASE: OUTCOME

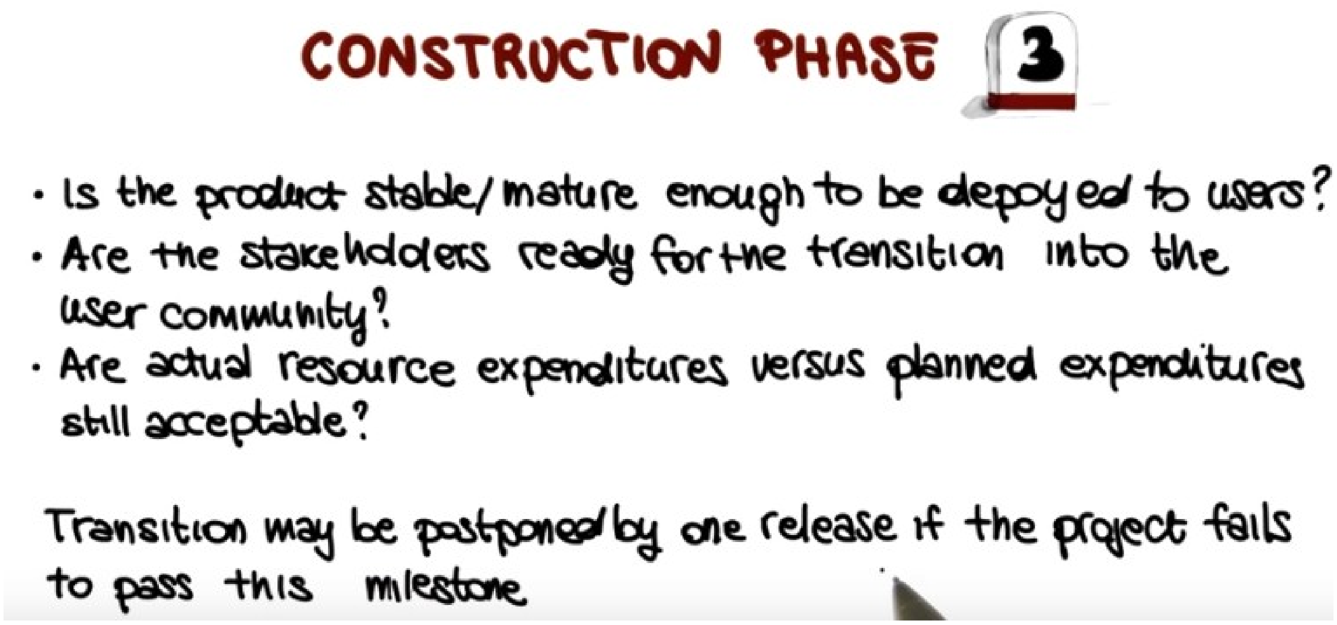
All use cases realized, with traceability information

Software product integrated on adequate platforms

Complete system tests results

User manual

Complete set of artifacts: design, code, test cases



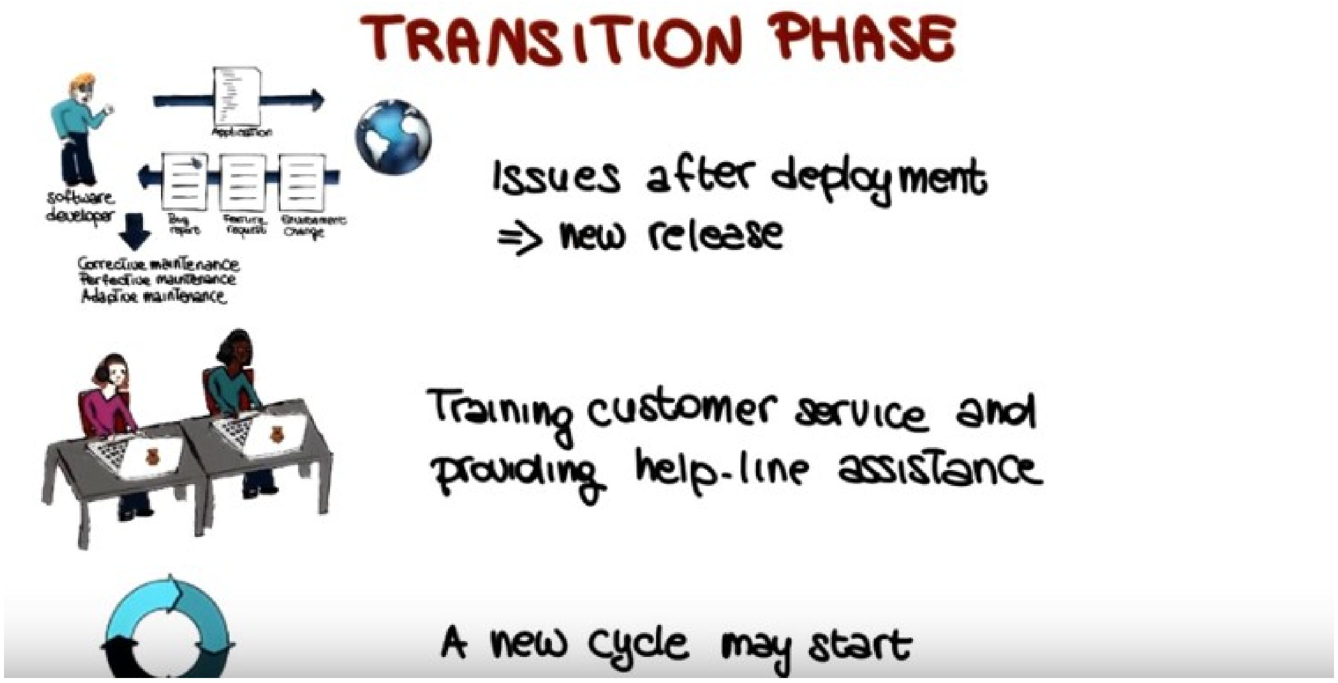
CONSTRUCTION PHASE 3

- Is the product stable/mature enough to be depoyed to users?
- Are the stakeholders ready for the transition into the user community?
- Are actual resource expenditures versus planned expenditures still acceptable?

Transition may be postponed by one release if the project fails to pass this milestone

19. But if we are ready to go to the market, if we are ready to deploy our product, then we can move to the transition phase, which has mainly to do with deployment and maintainence of a system. So what are the main activities in the transition phase? As we discussed in our initial lessons, in most real world cases, there are issues that manifest themselves after deployment, when we release our software and actual users interact with the software. Specifically, users might report failures that they experienced while using the system. So, what we call bug reports. Or they might report improvements they might want to see in the software. So typically these will be new feature requests. And in addition, there might be issues that don't come necessarily from the users but that are related to the fact that our system has to operate, has to work in a new execution environment. For example, the new version of an operating system, or the new version of a set of libraries. When this happens, we have to address these issues by performing maintenance. Specifically, corrective maintenance for bug reports, perfective maintenance, for feature requests, and adoptive maintenance, for environment changes. And the result of this is that we will have a new release of the software. Other activities that are performed in this phase include training, customer service, and providing help-line assistance. Finally, if you remember what we saw when we were looking at the banking IT system example, the cycle within a development are not necessarily completely dis-joined, but they might overlap a little bit. So something else that might happen in the transition phase is that a new cycle may start. So there might be some activities that are related to the fact that we're starting to think about a new cycle. So now let's see what kind of outcome these activities would produce. The first one is a complete project with all the artifacts that we mentioned before. Another outcome is that the product will be actually in use. So the product will be in the hands of the users and the users will start using it, will start interacting with it, for real, not just in a beta testing setting. Another outcome will be a lesson learnt. What worked. What didn't work. What should we do different in the next cycle or in the next development? And this is a very important part of the whole process, because it what provides feedback between cycles, and between projects. And as we said before, in case we have a next released planned or a next cycle coming up, we might want to start planning for the next release. So another outcome will be the plan for the next release. So similar to the other phases, also for the transition phase, we have a milestone, which is the fourth milestone in this case. And therefore we have evaluation criteria for the transition phase that will define whether we've reached the milestone or not. And in this case, one important assessment is whether the user is

satisfied. So users are actually using our product now, so we can get feedback from them, we can see whether the product makes them happy or not. And we continue assessing whether our expenditures are fine with respect to our estimates. And in this case, problems with this milestone might lead to further maintenance of the system. So for example, we might need to produce a new release to address some of the issues that the users identified, as we discussed a couple of minutes ago.



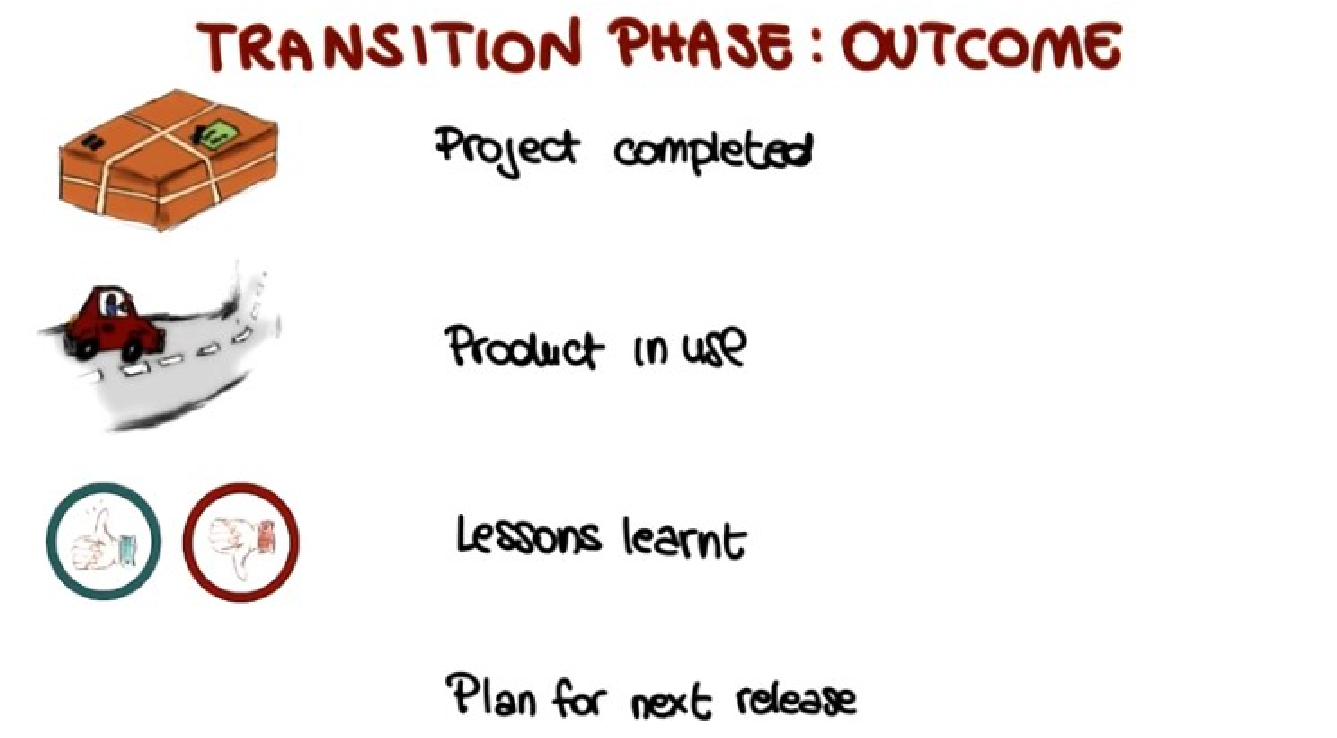
TRANSITION PHASE

Issues after deployment
=> new release

Training customer service and providing help-line assistance

A new cycle may start

Application
software developer
Bug report
Feature request
Environment change
Corrective maintenance
Perfective maintenance
Adaptive maintenance



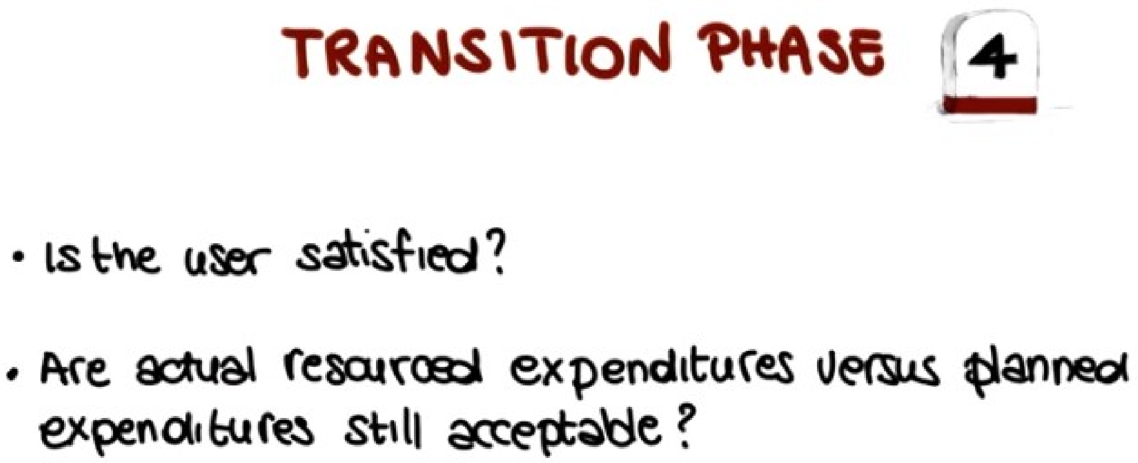
TRANSITION PHASE: OUTCOME

Project completed

Product in use

Lessons learnt

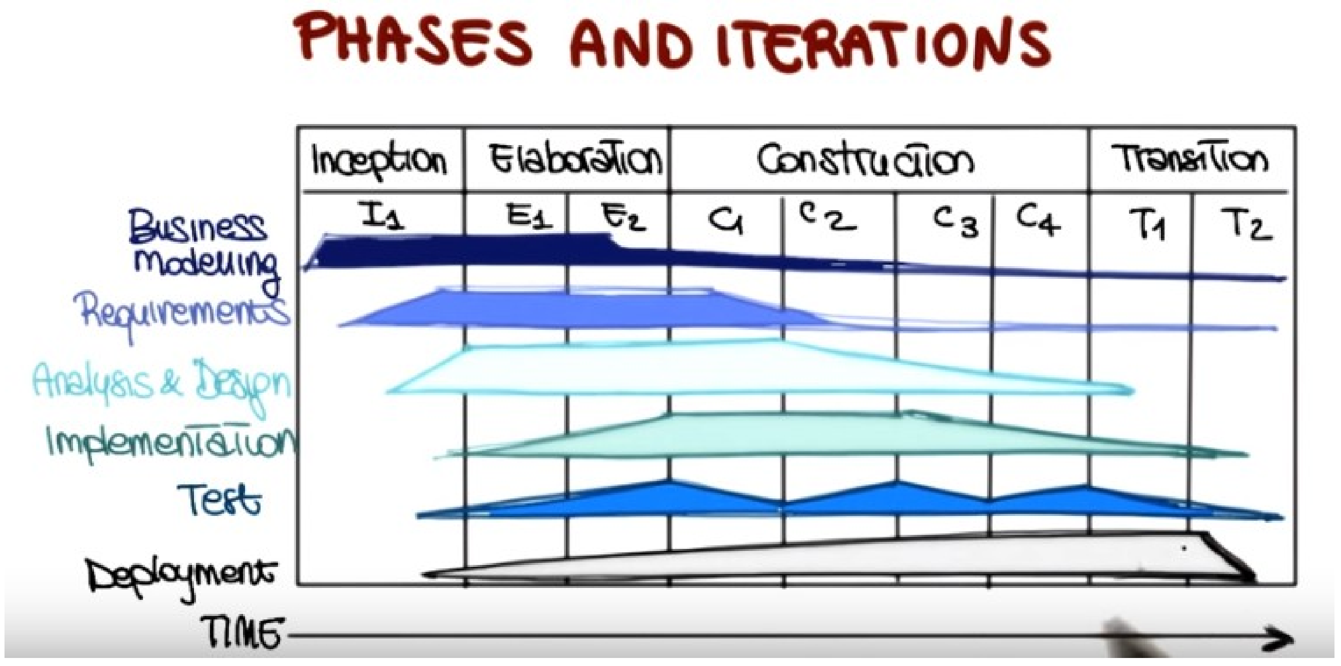
Plan for next release



TRANSITION PHASE 4

- Is the user satisfied?
- Are actual resourced expenditures versus planned expenditures still acceptable?

20. So now I would like to wrap up this lesson by going back to our discussion of rational unified process phases and iterations. So to do that I'm going to bring back the presentation that I used before, the summary representation about phases and traditional software engineering activities. And I want to use this representation to stress and discuss a couple of things. Mainly I want to recap it because I think it is very important. What is the relation between the rational unified process, and the traditional software engineering phases, and software engineering activities? And I like to do it now that we have discussed the phases in a little more detail. So I want to make sure that it is clear by now how and when the traditional software engineering activities, the ones listed here, take place in the context of the RUP phases, the four listed up here. For instance, it should be clear why implementation takes place mainly in the construction phase. Why requirements engineering is prominent in the elaboration phase and why deployment activities occur mostly in the transition phase, and so on. So it should be clear now why the activities are so distributed in the four phases. It should also be clear that although there is normally one main phase for each activity, the activities really span multiple phases. Which is actually one of the interesting aspect of RUP. So the fact that you're not really done with an activity even in later phases. Why? Well, because that allows you, in subsequent iterations, to address problems that came up in previous iterations.


PHASES AND ITERATIONS
Inception
Elaboration
Construction
Transition
I1
E1
E2
C1
C2
C3
C4
T1
T2
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
TIME