

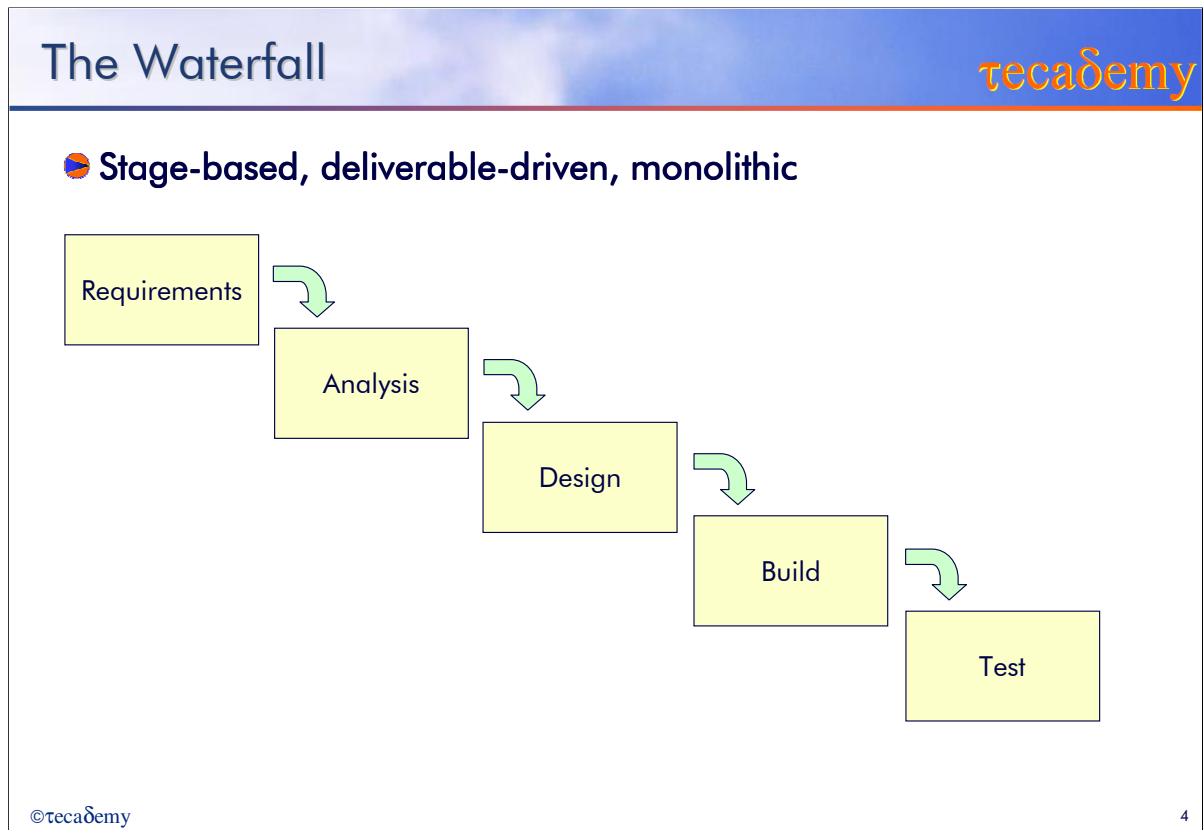
The Waterfall vs. the Whirlpool

Objectives

- To outline the key differences between the 'Waterfall' SDLC and the 'Whirlpool' iterative and incremental SDLC.

🎯 Contents

- The traditional approach – the 'Waterfall'
- The object approach – the 'Whirlpool'



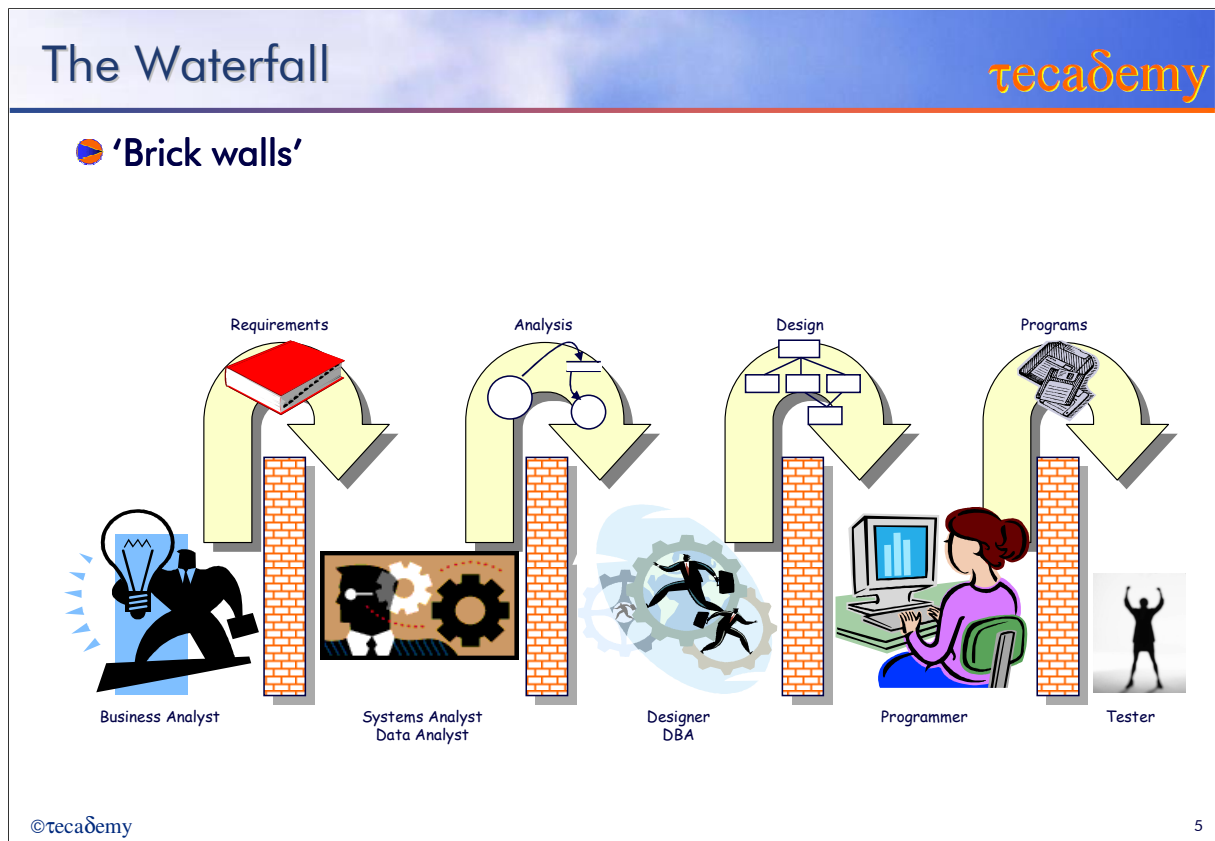
Early in the history of software development, it was recognised that the most volatile aspect of any system was the set of requirements. The solution offered by the waterfall model was to establish the requirements early, to get them agreed and signed off (effectively ‘freezing’ them) before beginning the development proper. All stages were defined by a set of deliverables, each one of which would be agreed and signed off before proceeding to the next stage.

Changes to the established requirements were discouraged by the expense of retro-fitting the changes into the already quality-checked and signed-off deliverables. This is why it was called the ‘Waterfall’ model – you can’t go *up* a waterfall!

The effect of ‘freezing’ the requirements so early was to guarantee that even the most perfectly-managed project would not meet the business requirements *at the time of delivery* – the best that could be achieved would be to meet the requirements as stated six, nine, twelve months earlier. In that time, of course, the business would have moved on, and the requirements would have changed – meaning that at best, a new stage would commence (that of maintenance and enhancement) or, at worst, the system becoming irrelevant. In practice, it has always been necessary to amend or augment requirements, and this has always compromised the product in some way.

The enhancibility of a system is determined by the degree of foresight employed by the designers, but inevitably a point will be reached in the life of a system where enhancement is no longer viable. At this point either business processes are frozen to fit the system, or a new system is built from scratch.

The advantage of the waterfall model was that it provided a comfortable fit with established project planning and control mechanisms used in other engineering disciplines. PRINCE2 is an example of such a project management method. This is an effective approach when building a structure with well-defined objectives and a relatively stable architecture (such as a bridge or a tunnel). It is arguable whether business software systems can be described thus.



It is often the case that project teams using the waterfall model organise themselves into stage-oriented teams of requirements gatherers, analysts, designers, programmers and testers, with maybe some team members straddling adjacent disciplines. Each team has their own set of tools and diagrams aimed at producing their stage's deliverables; those of analysts, say, differing from those used by programmers.

Each team develops the model to the point where it can be 'thrown over the wall' to the team responsible for the next stage, who then get on with the job of producing a set of deliverables to be passed to the next team. It could be argued that the documentation goes through a process of continual translation - the analyst translating the requirements into a form they hope the designers will understand, the designers translating the analysis into a model they hope the programmers will understand, and the programmers translating the design into code they hope their compilers will understand!

In the waterfall approach, a distinction is also drawn between data and process. This too is reflected in the organisation of the team - data modellers having a very different view of the system than their process-modelling colleagues.

It requires only that all these diverse models be brought together to produce a system that meets all of the requirements on time and on budget. Easy peasy!

The Whirlpool

Iterative, Incremental development

- Use Case driven
- Architecture-centric
- Risk tackled early

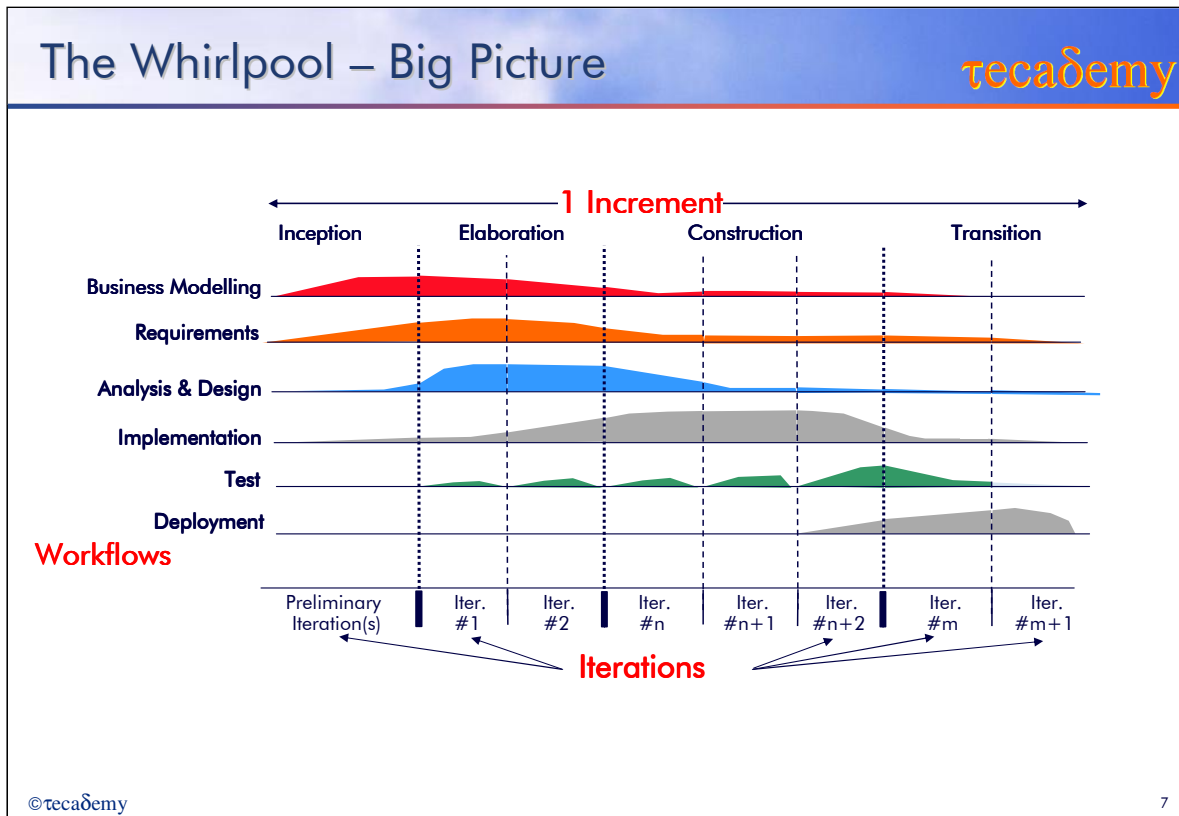
Workflows vs. Stages

- The Big Picture
- Inception
- Elaboration
- Construction
- Transition

'Whirlpool' projects embrace the fact that requirements are volatile. Instead of trying to deliver a system that satisfies 100% of the requirements in one go, the focus is on delivering an agreed subset of the requirements at intervals. Use cases are the building blocks of requirements and provide a convenient basis for risk assessment and prioritising. The riskiest and most significant use cases are tackled first, within the context of a clearly understood architectural vision. Tackling risk early identifies problems before they become too expensive or threatening, and helps lead to more stable architectures. The system evolves as a sequence of increments, each one an enhancement to the already-delivered functionality.

The development process for each increment is founded on the idea of a *workflow*, consisting of *activities*. For example, instead of having an analysis stage followed by a design stage, a worker might be engaged in analysis activities and design activities, depending on the job in hand. It is important to recognise that analysis activities focus on the problem space, while design activities focus on the solution space. A good understanding of one part of the problem may well lead to faster progress to design in that area. It is also possible that design activities highlight shortcoming in the analysis or even the requirements, in which case these are amended to reflect the newer (and hopefully better) understanding.

Successive iterations converge on a more complete, correct and consistent model, leading ultimately to implementation and delivery.



The Unified Process(UP) is an example of an iterative, incremental lifecycle. UP splits the project into four time phases: Inception, Elaboration, Construction & Transition. At first sight, this does not appear to be much different to most other software methods, but there is a difference in focus.

Each phase is characterised by the level of detail in the model. For example, some requirements are captured during Inception, but only enough to assess the overall project scope and risk. There is unlikely to be significant implementation or testing activity at this early phase.

More requirements are captured during Elaboration, but still probably only the highest-risk 80%. But there will be more analysis and design going on than before, maybe with some of the better-understood parts actually being coded. Some remaining requirements might be captured and analysed during Construction, potentially right up to the time the beta version is completed at the end of the phase, but the dominant activities here are to do with implementation and testing.

The overall shape is still somewhat like the waterfall approach, only there may be several trips around the waterfall - this fits with many people's experience of what actually happens on waterfall projects!

All workflows use the same models and notation, adding different levels of detail. The closer to Transition, the more consistent the level of detail across the whole model.

We now look more closely at the four phases.

First... Inception

- Understand the problem
- Define the scope
 - Identify as much functionality as possible
- Model the requirements
 - Identify the project risks
 - Prioritise Use Cases against project risk



The customer chooses a payment method. The details are checked, and recorded. May be successful or unsuccessful. May be retried.

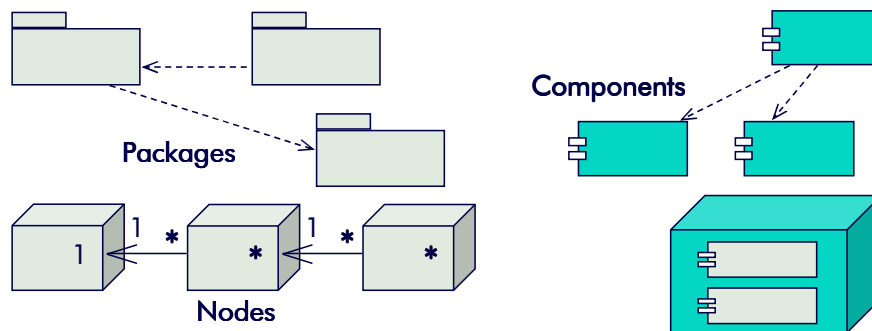
Use Case

The inception phase should define what the project is to achieve. It may be necessary to do this to a great level of detail or it may be sufficient to provide a glossy high level view. A high level view may contain just actors, a deeper view may also contain packages, a still deeper view could have names of many or all use cases. The deepest view would contain completed use cases.

The inherent risks should be identified during this phase. The risks will point at possibilities of failure for the project and decisions need to be taken whether to accept the risks, cancel the project, or find alternative less risky projects. Often the risks are unquantifiable until some exploratory work has been done. The exploratory work would dig deeper in some system areas than others, in non-risky areas use case packages may suffice, in moderately risky areas use cases could be written, in do-or-die cases the use cases could be analysed, designed and implemented.

Then... Elaboration

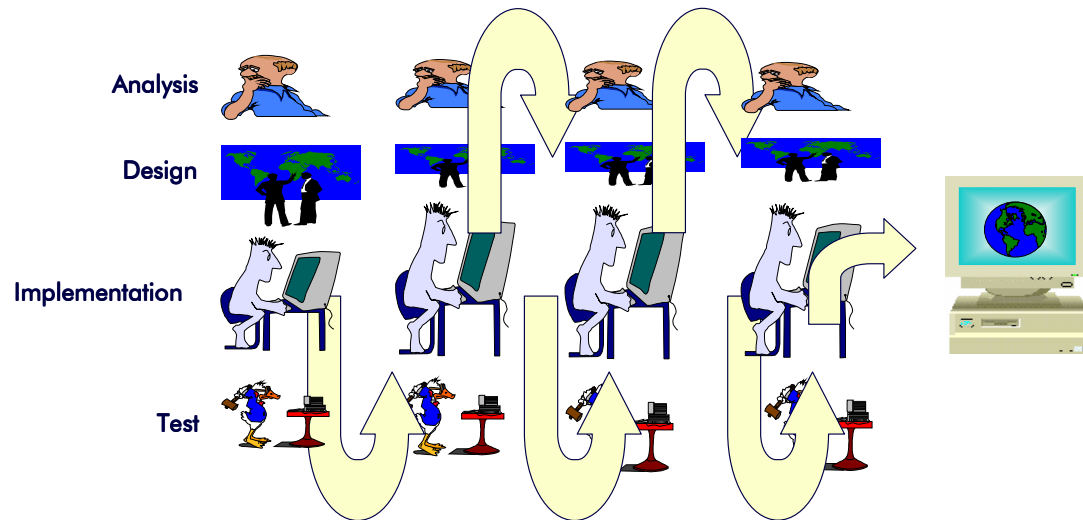
- Build the Use Cases which eliminate risk
- Incrementally define and build the architecture
- Schedule the next iterations



The elaboration phase should build the system architecture. The aim is to stabilise this so that the construction phase additions will not require wholesale changes to the system. For example the style of communication between the GUI and the business objects should be set early. Of course this is not a trivial task, nor one which will be got right first time, therefore during this phase you should find occurrences which will test the communications and implement these, until satisfied that the architecture will be able to cope with forthcoming functionality. These occurrences will be present in certain use cases and not others. Therefore elaboration is about finding the architectural risks, then finding the use cases which exercise those risks, and building them.

Then... Construction

Iteratively build and test the system:



The construction phase is essentially the rest of the functionality which has not been implemented. The starting point for the increments here depend on the detail covered in the inception phase, and the completeness of the increments in the elaboration phase.

As the architecture should be stable, the increments in this phase should be available for parallel attack by different teams. Ensure however that there is little overlap between the iterations otherwise there will be too much communication between the teams.

Finally... Transition

- Roll out the solution to the user community
- Customer acceptance tests
- Finalise documentation
 - User manuals, installation guide
 - Review work to date
- Train users
- Plan next increment
 - ... and round we go again!

Alpha, beta and customer releases have to be planned and implemented here.

BUT!!!

tecademy

- At present, there is no 'brand-name' Whirlpool Project Management (PM) framework
- Established PM frameworks are designed for Waterfall Projects
- Existing PM metrics, control and estimation techniques obsolete
- One day, our PRINCE will come...



©tecademy

12

The 'Whirlpool' approach has many advantages over the 'Waterfall'. But no project will get very far unless it is managed and controlled effectively – which is why we have Project Managers (PMs). Most of the mainstream project management methods (PRINCE II, for example) are based very much on the stage-based, deliverable-driven approach of the 'Waterfall' lifecycle, which is why they seem to 'fit' together so well. Estimating techniques are based on experience gained over a number of projects within a given domain using known skill-sets in a given technological framework.

So what happens when all or most of that changes? What value are the techniques and metrics and control points of yore?

The bad news is that there is, at the time of writing, no project management framework of 'brand-name' status to which 'Whirlpool' PMs can subscribe. So it is vitally important that PMs acquaint themselves with and address themselves to the major differences between the lifecycles, and learn new tricks. Not to do so is to virtually guarantee failure to deliver a quality product on time and on budget. QA's course '*Project Management for Object-Oriented Projects*' would be a good place to start climbing that learning curve.

The good news? Well... there isn't any. Yet. Sorry.*

* Unless you count the fantastic opportunity to put together a completely new, 'Whirlpool-friendly' project management framework, get it marketed under your name and become a sought-after guru with considerably enhanced status and income...

Summary

- ▶ **Each increment provides an updated product**
- ▶ **Each iteration includes multiple workflows**
 - ▶ Analysis, design, implementation, testing
- ▶ **Implications**
 - ▶ Increases tempo of project
 - ▶ Early integration testing, increases regression testing
 - ▶ Improved expectation management
- ▶ **Advantages**
 - ▶ Early business exposure and feedback to deliverables
 - ▶ Development team see early impact of decisions
 - ▶ Focus on risk reduction

The iterative and incremental approach has many differences from and benefits over the traditional waterfall approach. Firstly it is not artefact driven, we are not striving for the best class diagram in the world (or use cases or sequence diagrams etc) and hoping to get it signed off so that we never touch it again. Rather in each iteration we provide another increment of the product, some real tangible visible result which can be discussed much more effectively than models, and which highlights the progress being made. As it is the product which is signed off rather than the documentation, the documentation can be changed. This freedom is necessary to avoid mistakes being perpetuated simply because they have been signed off. Of course this needs a rethink of project management and customer involvement.

So what are some of possible outcomes of using an iterative incremental approach? As there are more frequent deliveries, the tempo of the project may increase. Integration testing will start early. More testing will be involved as increments of the system will require both integration and regression testing. As with any release, its audience will need to know what to expect, so with more releases there will be more expectation management required.

And what are the advantages of an iterative and incremental approach? As increments are delivered the business will start to understand the system being delivered and give early feedback. If the quality of the increment delivered is high, the business are more likely to trust the development team. In turn a positive relationship between the two groups is fostered. The business are more likely to take ownership of the system. They have more time to understand the system and consider any business process implications. Finally assimilation into the business community will be easier.

Likewise the development team will experience the impact of any decisions. This will give them an opportunity to refine the way they work and grow in their maturity.

All of the above address major risks in the development lifecycle.

Summary

- ▶ **Iterative & incremental**
 - ▶ Iterative: be prepared to have a few attempts at most things
 - ▶ Incremental: Build the system in stages
- ▶ **Architecture centric**
 - ▶ Architecture is usually a risk
 - ▶ Build early, focus on minimising risks
- ▶ **Use Case driven**
 - ▶ Building-block of requirements
 - ▶ Basic unit of delivery
 - ▶ Used for Test Cases
- ▶ **BUT - No supporting Project Management framework**

The Unified Process is founded on three main ideas, all of which stem from known, practical drawbacks of the Waterfall approach.

When building systems with complex requirements and complex solutions, it's impossible to get everything right first time. So the Unified Process assumes that most deliverables: requirements, analysis, design and code, will have levels of detail added to them over time. An initial attempt will be made to produce it, then reviewed, then a better version produced.

Building large *and* complex systems is hard: building small and complex system is always going to be easier. So Unified Process recommends that we aim to split out systems into pieces, to make designing and testing easier, and so reduce the overall project risk.

The Unified Process also recommends that we concentrate on the overall system architecture early in the project, and don't wait until just before the main coding starts. That's because experience has shown that the earlier we start to think about architecture, the more time we have to understand and address the risks involved.

Finally, the Unified Process uses the idea of use cases as the backbone of the project. Use cases are used to specify system requirements, as the source for the OO models we need to construct the code, as the source of test cases, and as the basis for planning the delivery increments.

The biggest impediment to all the above is the absence of an established project management method supporting the iterative and incremental lifecycle. The second biggest impediment is the Project Manager who fails to educate him/herself in what the differences are between 'waterfall' and 'whirlpool' projects.