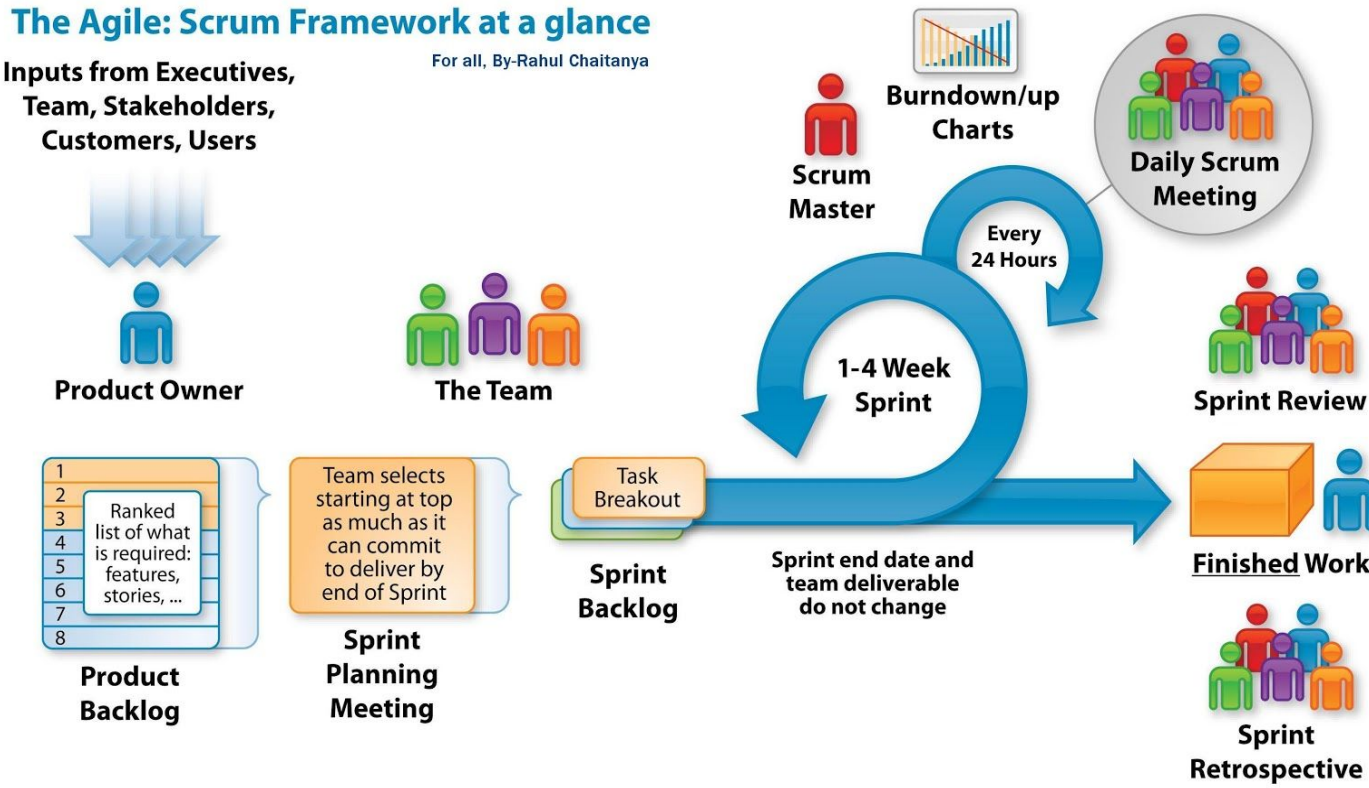
The Bears

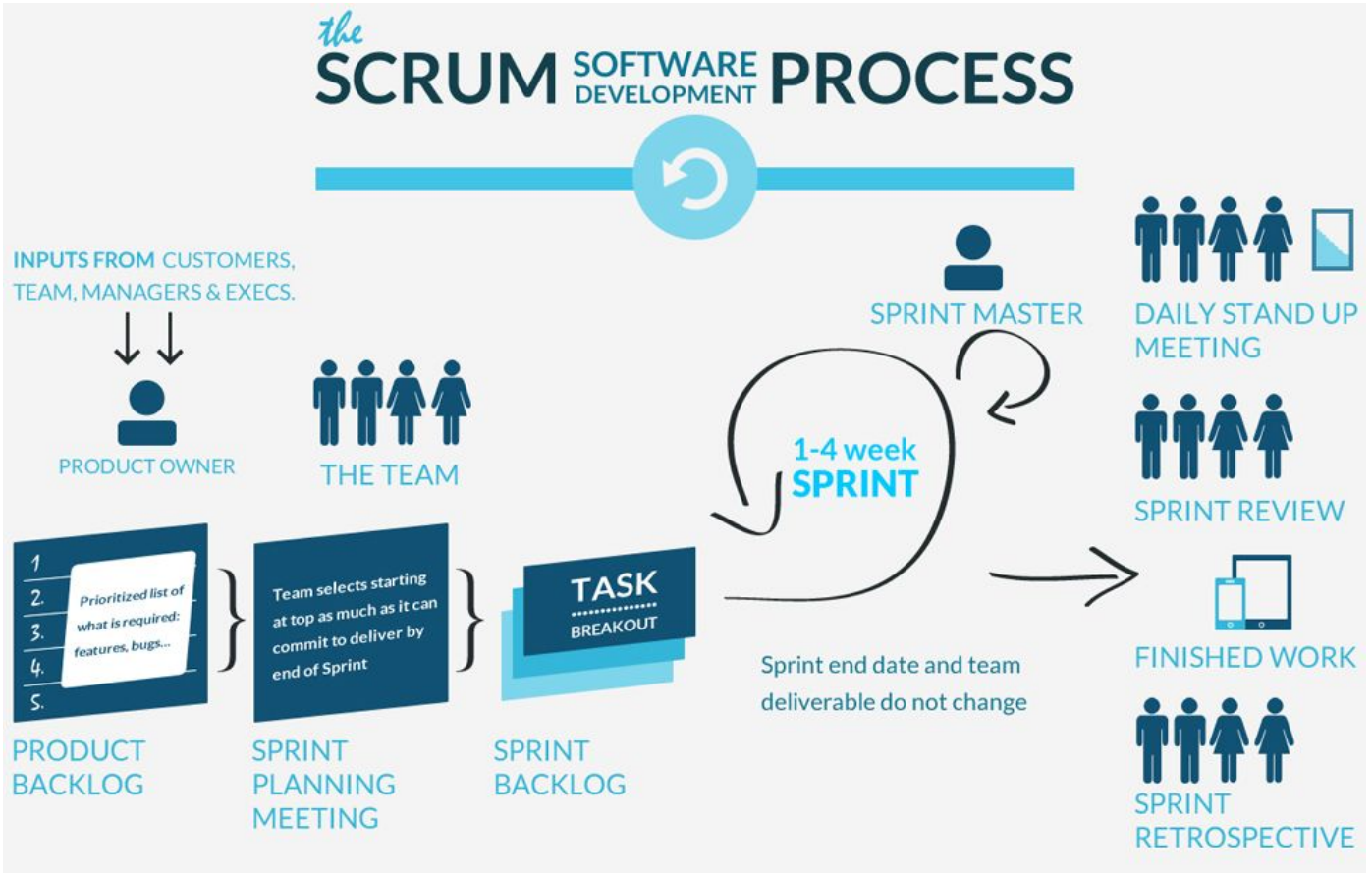
Final notes til SYS

Scrum

The Bears


The Agile: Scrum Framework at a glance
For all, By-Rahul Chaitanya
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

The Bears



XP

Extreme Programming (XP) is a no nonsense, code first approach to software delivery that emphasizes [four basic activities]():

1. Coding	You **code** because if you do not code, you will haven't done anything
2. Testing	You **test** because if you don't test, you don't know when you are done coding
3. Listening	You **listen** because if you don't listen you don't know what to code or what to test
4. Designing	You **design** so you can keep coding and testing and listening indefinitely

The Bears

4 (5)Values

Værdierne skal sikre, at individet er orienteret mod projektets interesser (fælles kultur)

Meddelelse
Sørg for kontinuerlig og relevant kommunikation
XP-praksis er ikke mulig uden kommunikation

Enkelhed
Bedre at opbygge enkle løsninger, som vil blive ændret end at opbygge komplekse løsninger, der aldrig bliver brugt

Feedback
"Optimisme er en erhvervsmæssig risiko for programmering" vs. "Spørg mig ikke - spørg systemet" (konkret feedback)

Mod
Vær klar til at ændre systemet - selv dramatisk og sent

Respekt
Alle giver og føler den respekt, de fortjener som et værdsat holdmedlem

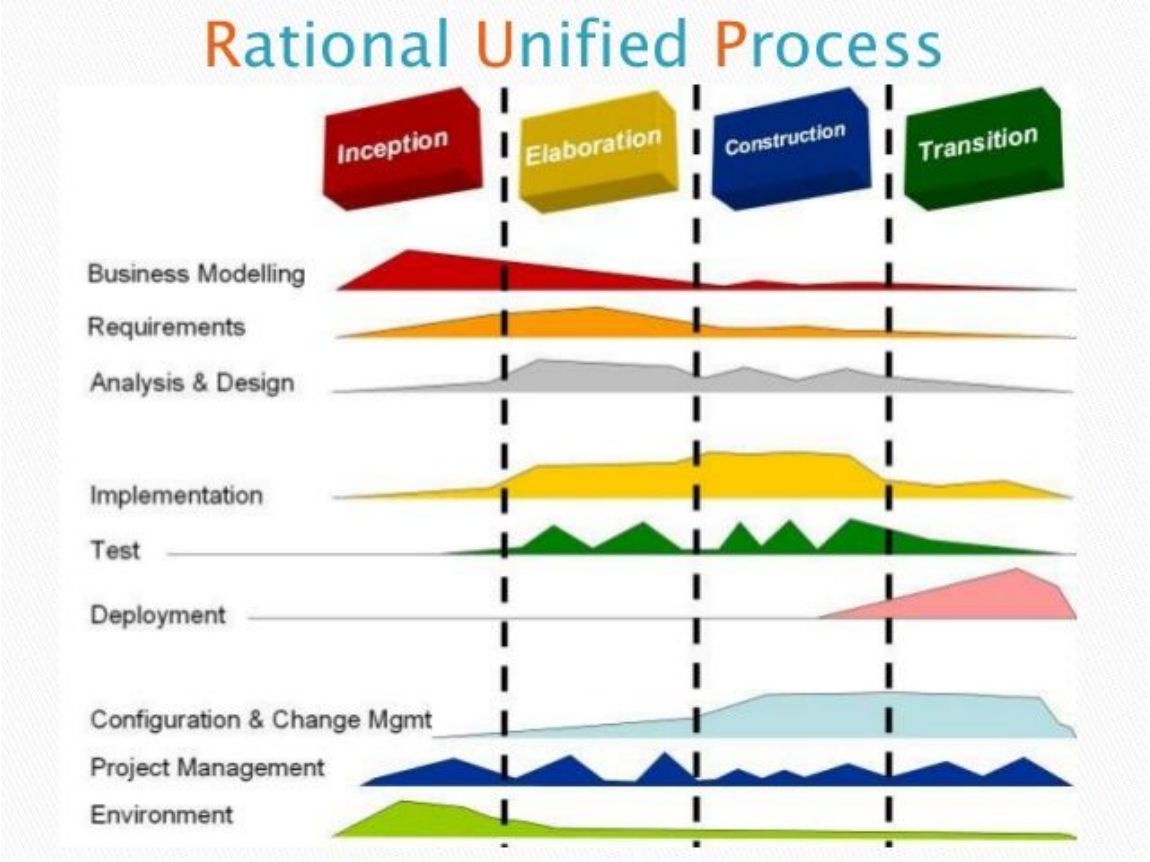
De 12 XP Praktikker
Planning game (User stories)
Small releases (Små iterationer)
Metaphor (standardiserede navne ordninger, XP-systemudvikling kræver, at der overholdes et sæt standarder for elementer som variable navne, klassenavne og metoder)
Collective ownership
Coding standard
Simple design
Refactoring
Testing
Pair programming
Continuous integration
40 hour week

The Bears

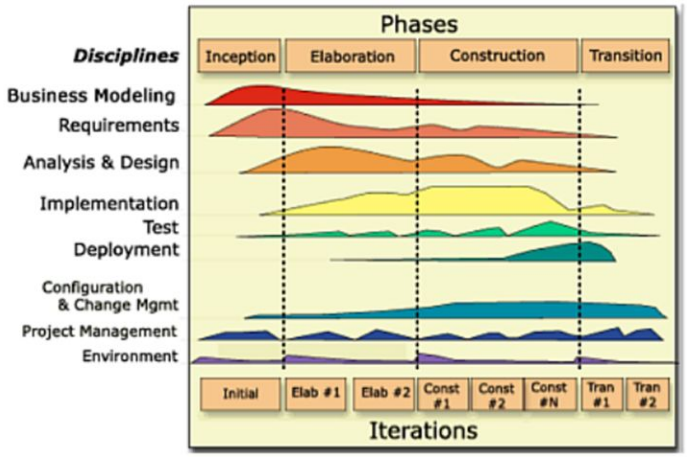
On-site customer

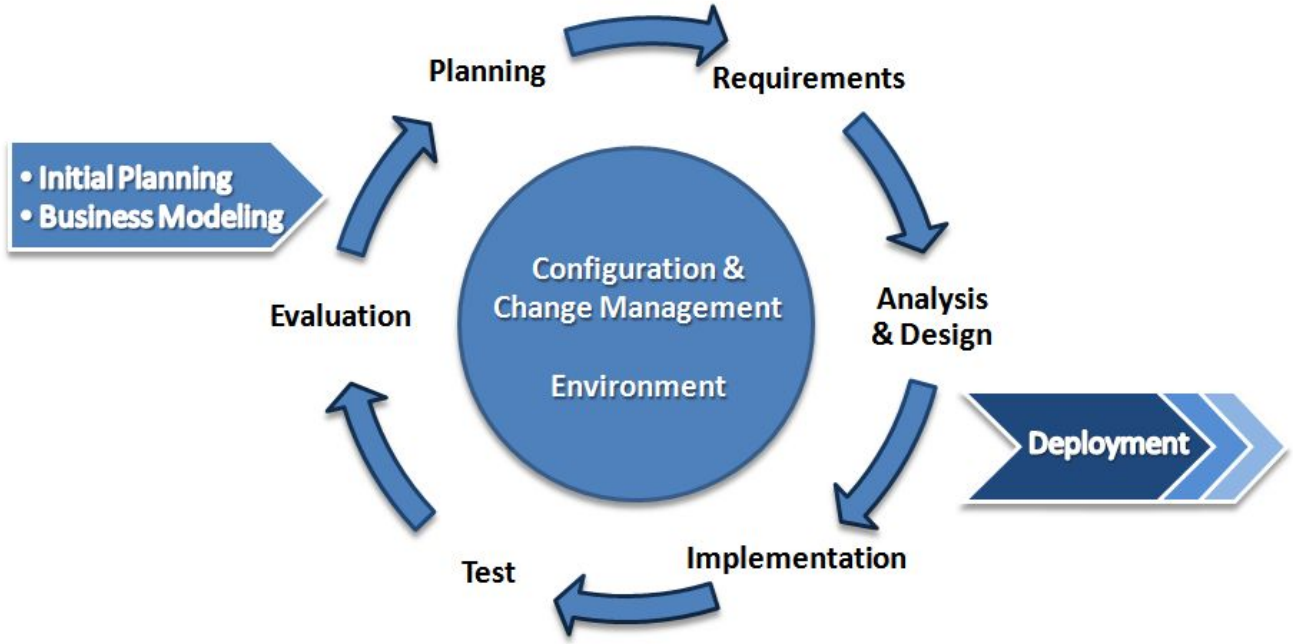
UP

RUP / UP er Bruger-drevet og stærkt afhængig af Unified Modeling Language eller UML.



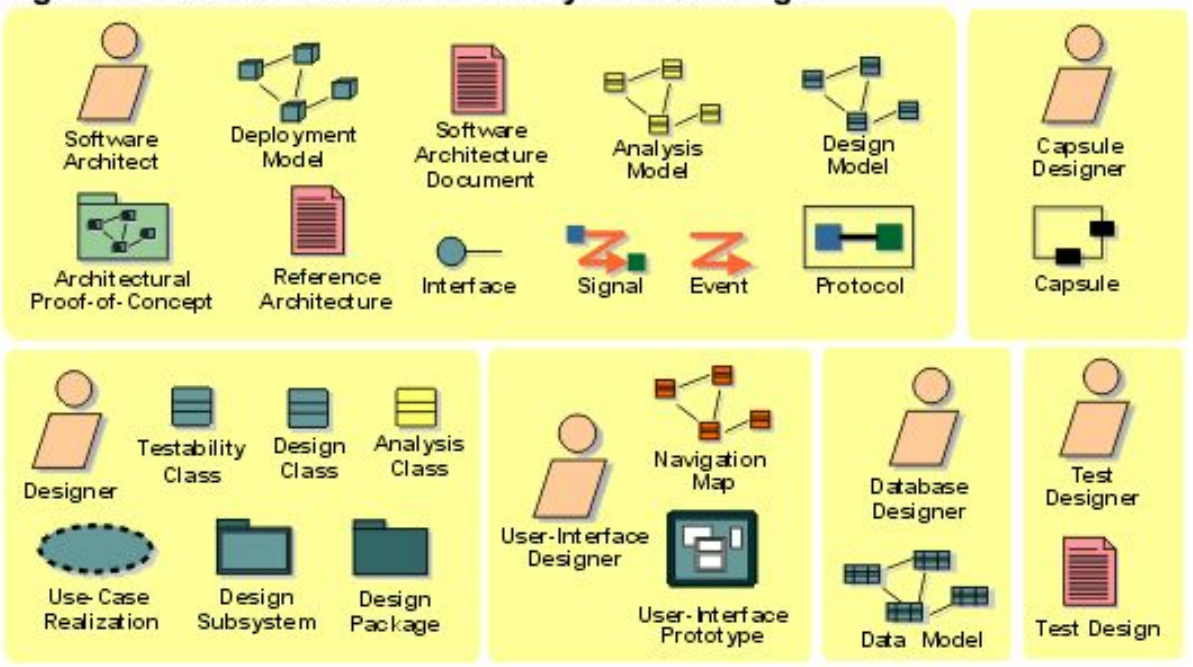
The Bears


Phases
Disciplines
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations


Planning
Requirements
• Initial Planning
• Business Modeling
Configuration & Change Management
Environment
Analysis & Design
Deployment
Evaluation
Test
Implementation

The Bears

Figure 1: The RUP artifacts for Analysis and Design



During the inception phase, you establish the business case for the system and delimit the project scope.

The outcome of the inception phase is:

- A vision document: a general vision of the core project's requirements, key features, and main constraints
- A initial use-case model (10% -20%) complete)
- An initial project glossary (may optionally be partially expressed as a domain model)
- An initial business case, which includes business context, success criteria and financial forecast
- An initial risk assessment
- A project plan, showing phases and iterations
- A business model, if necessary
- One or several prototypes

The Bears

cphbusiness

At the end of the inception phase is the first major project milestone
The Lifecycle Objectives Milestone

The evaluation criteria for the 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 architectural prototype that was developed
- Actual expenditures versus planned expenditures.

The project may be cancelled or considerably re-thought if it fails to pass this milestone.

cphbusiness

The purpose of the elaboration phase is to analyze the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project.

The outcome of the elaboration phase is:

- A use-case model (at least 80% complete) — all use cases and actors have been identified, and most use-case descriptions have been developed
- Supplementary requirements capturing the non functional requirements and any requirements that are not associated with a specific use case
- A Software Architecture Description
- An executable architectural prototype
- A revised risk list and a revised business case
- A development plan for the overall project, including the coarse-grained project plan, showing iterations" and evaluation criteria for each iteration
- An updated development case specifying the process to be used
- A preliminary user manual (optional)

The Bears

cphbusiness

At the end of the elaboration phase is the second important project milestone, the Lifecycle Architecture Milestone. You examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks.

The main evaluation criteria for the elaboration phase :

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the plan for the construction phase sufficiently detailed and accurate? Is it backed up with a credible basis of estimates?
- Do all stakeholders agree that the current vision can be achieved if the current plan is executed to develop the complete system, in the context of the current architecture?
- Is the actual resource expenditure versus planned expenditure acceptable?

The project may be aborted or considerably re-thought if it fails to pass this milestone.

cphbusiness

During the construction phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested.

The outcome of the construction phase is a product ready to put in hands of its end-users. At minimum, it consists of:

- The software product integrated on the adequate platforms
- The user manuals
- A description of the current release

The Bears

cphbusiness

At this point, you decide if the software, the sites, and the users are ready to go operational, without exposing the project to high risks. This release is often called a "beta" release.

The evaluation criteria for the construction phase involve answering these questions:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition into the user community?
- Are the actual resource expenditures versus planned expenditures still acceptable?

cphbusine

The purpose of the transition phase is to transition the software product to the user community.

This typically requires that some usable subset of the system has been completed to an acceptable level of quality and that user documentation is available so that the transition to the user will provide positive results for all parties.

This includes:

- "Beta testing" to validate the new system against user expectations
- Parallel operation with a legacy system that it is replacing
- Conversion of operational databases
- Training of users and maintainers
- Roll-out the product to the marketing, distribution, and sales teams

The Bears

The transition phase focuses on the activities required to place the software into the hands of the users.

Typically, this phase includes several iterations, including beta releases, general availability releases, as well as bug-fix and enhancement releases.

The primary objectives of the transition phase include:

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baseline as rapidly and cost effectively as practical

At the end of the transition phase is the fourth important project milestone, the Product Release Milestone.

At this point, you decide if the objectives were met, and if you should start another development cycle. In some cases, this milestone may coincide with the end of the inception phase for the next cycle.

The primary evaluation criteria for the transition phase involve the answers to these questions:

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

The Bears

Is RUP – Agile?

When it comes to making RUP agile, the critical issues are philosophical:

Encompassing agility isn't about adopting the right tool or technique, it's about the mindset.

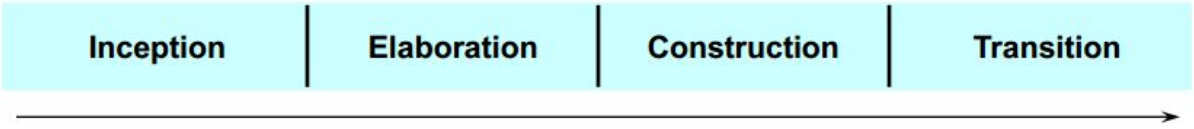
The essence of agility isn't a technique—it's a willingness to focus on people, the development of working software, the pursuit of active stakeholder participation and the embrace of change.

The Bears

Collaboration is the foremost hallmark of agility. How do you achieve this within RUP?

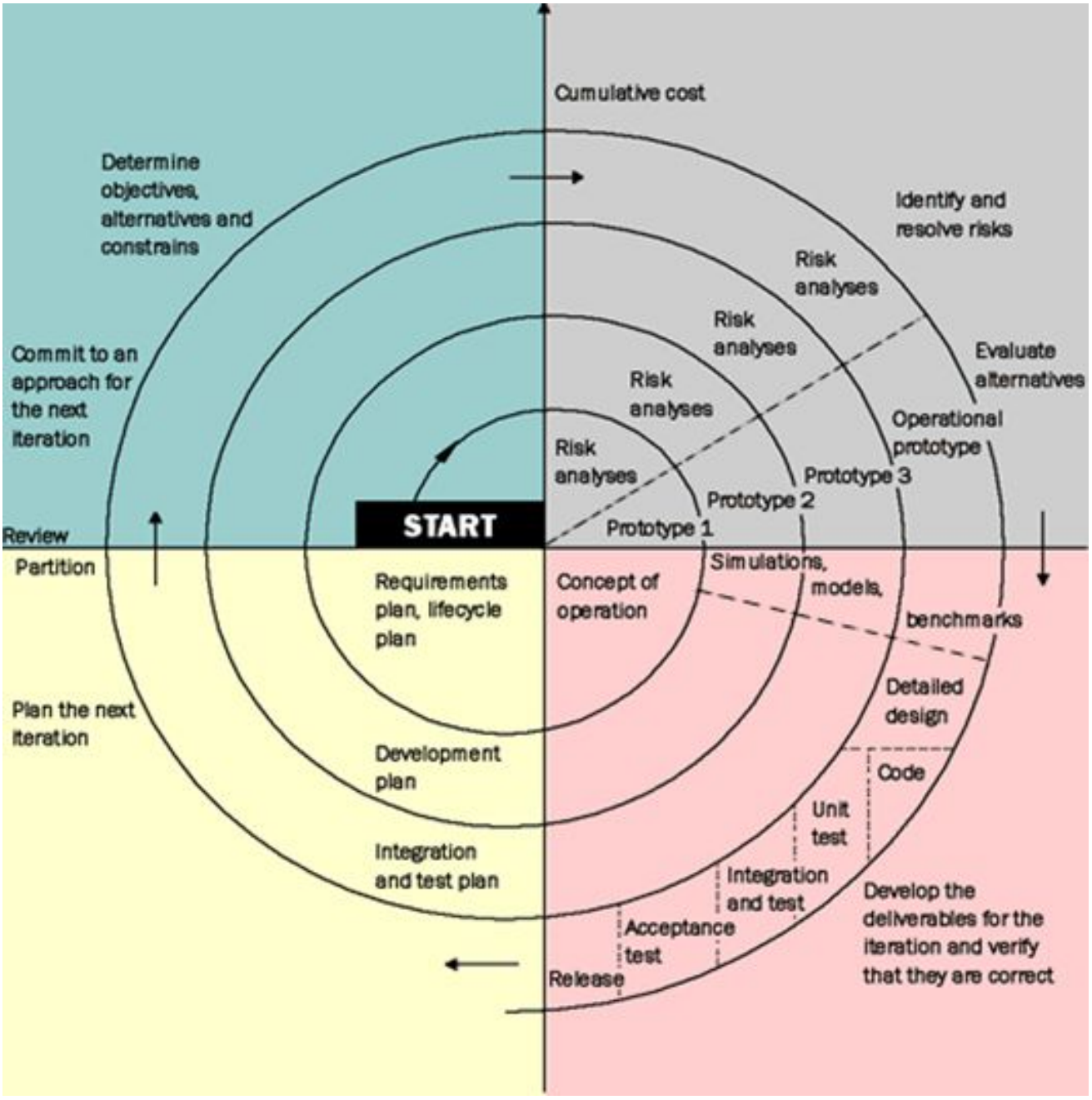
cphbusine

1. Build a team of people who want to work together, learn from each other and succeed together.
   Seems obvious, but how many projects have you been on that included people simply because they were available?
2. Support everyone's learning efforts with training, mentoring and coaching
3. Break down communication barriers: Co-locate the team and get your project stakeholders actively involved in development
4. Recognize common "process smells" that indicate serious communication problems.
   For example, the belief that you need detailed documentation is one such smell



Hvornår skal du bruge UP?
Du kan hævde, at du skal bruge UP til:
Komplekse og sofistikerede systemer, der udvikler sig trinvist i cyklusser
Du kan hævde, at du ikke bør bruge UP til:
Enkle eller velkendte systemer, hvor det kan være muligt at definere hele problemet sekventielt, designe hele løsningen, bygge softwaren og derefter teste produktet.

The Bears


Cumulative cost
Determine objectives, alternatives and constrains
Identify and resolve risks
Risk analyses
Risk analyses
Risk analyses
Risk analyses
Evaluate alternatives
Commit to an approach for the next iteration
Operational prototype
Prototype 3
Prototype 2
Prototype 1
START
Review
Partition
Requirements plan, lifecycle plan
Concept of operation
Simulations, models, benchmarks
Plan the next iteration
Detailed design
Development plan
Code
Unit test
Integration and test plan
Integration and test
Acceptance test
Release
Develop the deliverables for the iteration and verify that they are correct

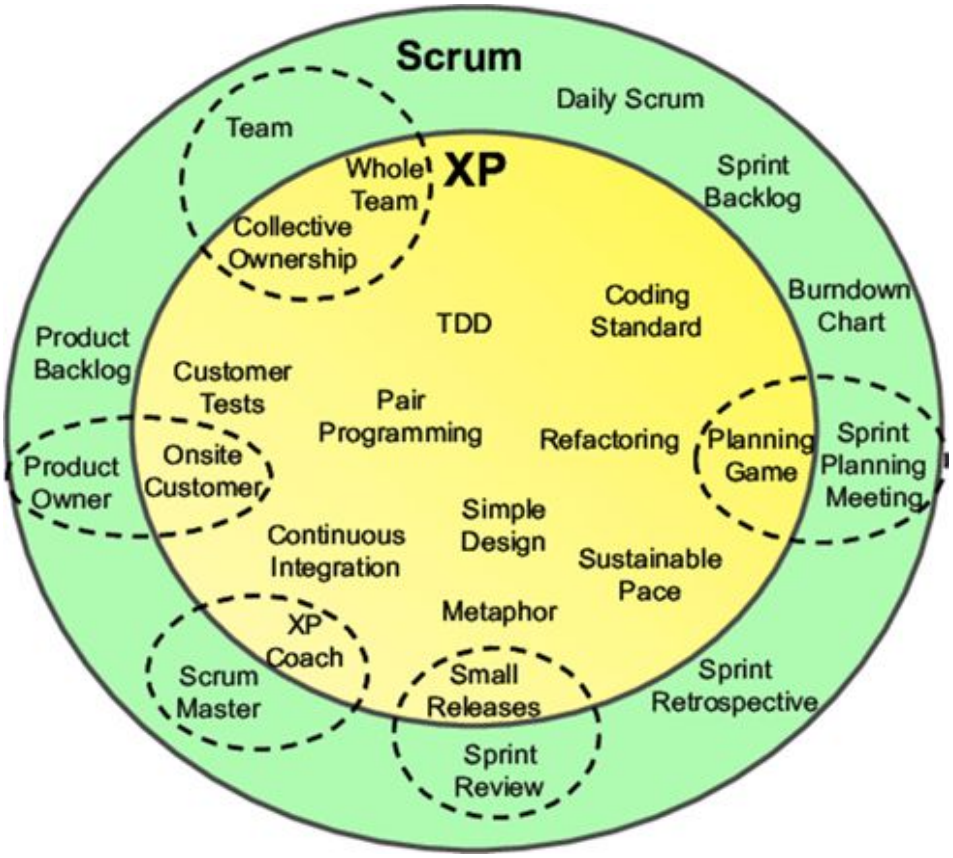
The Bears

Scrum vs UP

	UP	SCRUM
Approach	Iterative	Iterative
Cycle	Formal Cycle is defined across 4 phases, but some workflows can be concurrent.	Each sprint (iteration) is a complete cycle.
Planning	Formal project plan, associated with multiple iterations, is used. The plan is end-date driven and also has intermediate milestones.	No end-to-end project plan. Each next iteration plan is determined at the end of the current iteration (NOT end-date driven). Product Owner (Key Business User) determines when the project is done.
Scope	Scope is predefined ahead of the project start and documented in the Scope document. Scope can be revised during the project, as requirements are being clarified, but these revisions are subject to a strictly controlled procedure.	Instead of scope, SCRUM uses a Project Backlog, which is re-evaluated at the end of each iteration (sprint).
Artifacts	Vision/Scope Document, Formal functional requirements package, system architecture document, development plan, test plan, test scripts, etc.	The only formal artifact is the operational software and the user stories
Type of Project/ Product	Recommended for large, long- term, enterprise-level projects with medium-to-high complexity.	Recommended for quick enhancements and organizations that are not dependent on a deadline.

XP vs Scrum

SCRUM til ledelse
XP til udvikling



Scrum hold arbejder typisk i iterationer (kaldet sprints), der er fra to uger til en måned lang. ? XP-teams arbejder typisk i iterationer, der er EN eller TO uger lang.

Scrum hold tillader ikke ændringer i deres sprints. Når sprint planlægningsmødet er afsluttet og en forpligtelse til at levere et sæt af produktets backlog items, forbliver det sæt af varer uændret i slutningen af sprinten.

XP-hold er meget mere modtagelige for at ændre sig inden for deres iterationer. Så længe holdet ikke har startet arbejdet med en bestemt

funktion, kan en ny funktion af tilsvarende størrelse byttes ind i XP holdets iteration i bytte for den ikke startede funktion.

Ekstreme programmering arbejder i en streng prioriteret rækkefølge. Funktioner, der skal udvikles, prioriteres af kunden (Scrums Product Owner) og holdet skal arbejde på dem i den rækkefølge. Til gengæld prioriterer Scrum-produktejeren produktets backlog, men holdet bestemmer den rækkefølge, hvorpå de vil udvikle backlog items.

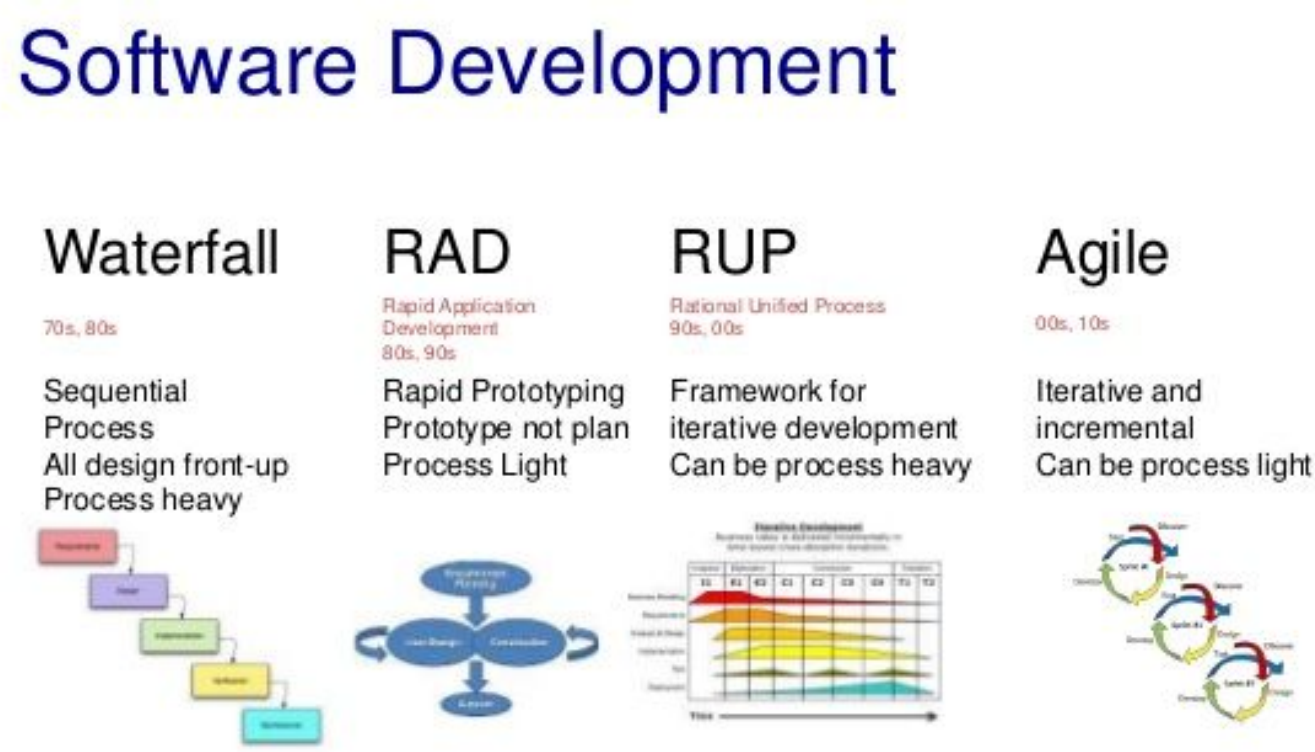
Scrum foreskriver ikke nogen engineering praksis - XP gør.

Scrum fokuserer mere på ledelsens side af projekter, ekstrem programmering vil fokusere mere på den faktiske teknik praksis.

Scrum	XP
• Changes in sprint are not allowed • Once tasks for a certain sprint are set, the team determines the sequence in which they will develop the backlog items • The Scrum Master is responsible for what is done in the sprint, including the code that is written • The validation of the software is completed at the end of each sprint, at Sprint Review	• As long as the team hasn't started working on a particular feature, a new feature, of equivalent size can be swapped into the interation in exchange for an un-started feature • Tasks are taken in a strict priority order • Developers can modify or refactor parts of code as the need arises • The software needs to be validated at all time, to the extent that tests are written prior to the actual software

The Bears

RAD (Rapid-application development)

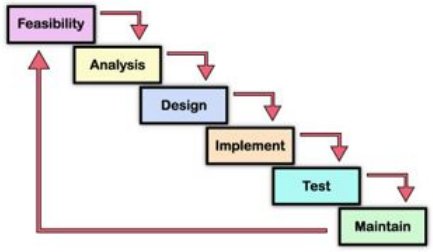


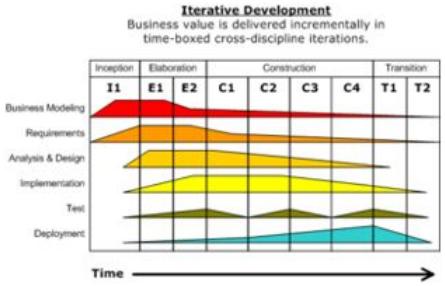
Hurtig applikationsudvikling (RAD) er både en generel betegnelse, der refererer til alternativer til den konventionelle vandfaldsmodel for softwareudvikling samt navnet på James Martin's tilgang til hurtig udvikling. Generelt lægger RAD-tilgang til softwareudvikling mindre vægt på planlægning og større vægt på proces. I modsætning til vandfaldsmodellen, som kræver en nøje defineret specifikation, der skal etableres inden indgangen i udviklingsfasen, fremhæver RAD-tilgange tilpasningsevnen og nødvendigheden af at tilpasse kravene som følge af viden opnået som projektet skrider frem. Prototyper bruges ofte ud over eller undertiden endda i stedet for designspecifikationer.

RAD er særligt velegnet til (men ikke begrænset til) at udvikle software, som er drevet af brugergrænsefladskrav. Grafiske brugergrænsefladebyggere kaldes ofte hurtig applikationsudviklingsværktøjer. Andre tilgange til hurtig udvikling omfatter agile metoder og spiralmodellen.

Waterfall vs UP vs Scrum/XP

Waterfall	UP	Agile (Scrum/XP)
• Sequential • Process • All design up front • Process heavy	• Iterative • Incremental • Can be process heavy	• Iterative • Incremental • Can be process light



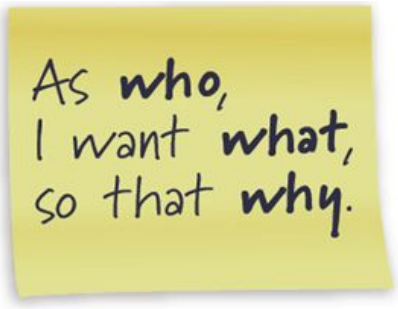


Waterfall vs Agile (Iterativ process)

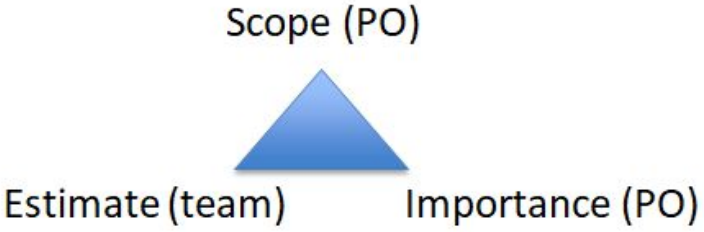
Waterfall	Agile
Proven Model to execute Projects	Suggested model for execution Products.
Sets expectations up front for cost, schedule	Continuous delivery and feedback cycles (iterative and incremental development)
Requirements must be validated and exit criteria must be met before proceeding to next phase	Changing requirements are welcome
Customer can focus on other things in the meantime	Early testing and continuous integration
"Measure twice, cut once" means less potential for rework	Customer collaboration and acceptance of each feature as it's developed

User stories vs Use cases

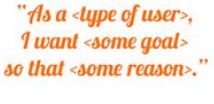
User stories

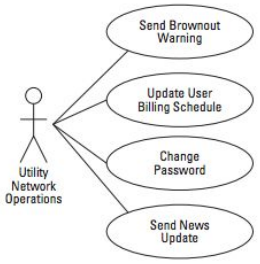


Formuleret af hold (inklusive PO)
Beregnet af teamet
Bruges som basis for sprint planlægning / planlægning spil



User Stories	Use Cases
Promise of a conversation	Record of a conversation
A way of prioritizing work	A way of describing requirements
Demands questions	Demands answers
What & Why	How
Business	System





Black-White-Grey box

Black box - testing

Test uden at have kendskab til applikationen
Typisk vil en tester interagere med systemets brugergrænseflade ved at levere input og undersøge output uden at vide, hvordan og hvor inputerne arbejdes på.

Advantages	Disadvantages
Well suited and efficient for large code segments	Limited coverage, only a number of test scenarios is performed
Code access is not required	Inefficient testing, due to the fact that the tester only has limited knowledge about an application
Clearly separates user's perspective from the developer's perspective	Blind coverage, since the tester cannot target specific code segments or error-prone areas
	The test cases are difficult to design

The Bears

White box - testing

Detaljeret undersøgelse af kodeksens interne logik og struktur
En tester skal kende kodenes interne funktion
Også kendt som glasprøvning eller open-box test

Advantages	Disadvantages
Easier to find out which type of data can help in testing the application effectively	Due to the fact that a skilled tester is needed the costs are increased
Helps optimizing the code	Difficult to maintain white-box testing, as it requires specialized tools like code analysers and debugging tools
Due to knowledge about the code, maximum coverage is attained during test scenario writing	Impossible to look into every corner to find hidden errors that may create problems, as many paths will go untested.

Grey box - testing

Test med begrænset kendskab til den interne drift af en ansøgning
Adgang til design dokumenter og databasen.

Advantages	Disadvantages
Offers combined benefits of black-box and white-box testing	The ability to go over the code and test coverage is limited due to lack of source code
Don't rely on the source code; instead they rely on interface definition and functional specifications	The tests can be redundant if the software designer has already run a test case
Excellent test scenarios for communication protocols and data type handling based on the limited information	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time
The test is done from the point of view of the user and not the designer	

Black vs White vs Grey boxing

The Bears

Black-box	Grey-box	White-box
The internal workings need not be known	Limited knowledge of the internal workings	Full knowledge of the internal workings
Also known as closed-box testing, data-driven testing, or functional testing	Also known as translucent testing	Also known as clear-box testing, structural testing, or code-based testing
Performed by end-users and also by testers and developers	Performed by end-users and also by testers and developers	Normally done by testers and developers
Based on external expectations - Internal behaviour is unknown	Based on high-level database diagrams and data flow diagrams	Internal workings are fully known, the test data can be designed accordingly
It is exhaustive and the least time-consuming	Partly time-consuming and exhaustive	The most exhaustive and time-consuming
only be done by trial-and-error	Data domains and internal boundaries can be tested, if known	Data domains and internal boundaries can be better tested

Experiment

Hypotese:
Vores Hypotese er at hvis man lægger flere tal sammen af samme datatype - Float og Double, så er de to resultater ikke det samme. Hvis man sammenligner de to tal er de ikke ens fordi at double har flere decimaler end float.

Experimentet:
Vi lægger dem sammen og gemmer tallene float og double i hver deres variable og laver en if else sætning og sammenligner de to variabler og hvis de er ens udskriver den ”De er ens” og hvis ikke det er tilfælde så udskriver den ”De er ikke ens”.

```
private static float a = 0.005f;
 private static float b = 0.0003000f;
 private static float c = 5.5f;
 private static float pi = 3.14f;
```


```
private static double d = 0.005;
private static double e = 0.0003000;
private static double f = 5.5;
private static double g = 3.14;

public static void main(String[] args) {

//long startTime = System.nanoTime();
// do something you want to measure
//long elapsedTimeNs = System.nanoTime() - startTime;

float resultfloat = a + b + c + pi;
double resultdouble = d + e + f + g;

System.out.println(resultfloat);
System.out.println(resultdouble);
if(resultfloat == resultdouble){
System.out.println("Det var det samme");
}
else{
System.out.println("Det var ikke det samme");
}
```

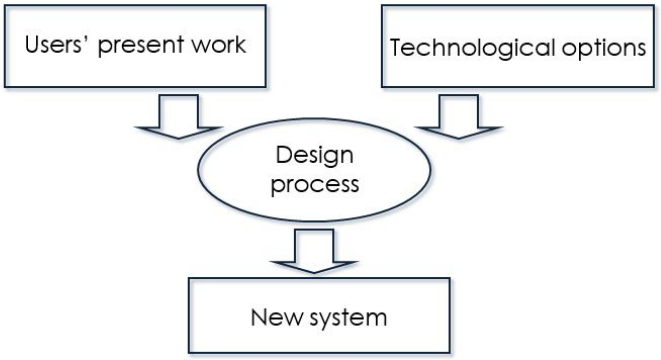
Eksekvere:
Det var ikke det samme

Evaluering:
Vi kan konkludere at der er forskel præcisionen mellem floats og doubles. Så hvis man skal lave et banksystem så skal det være ultra præcist, så skal man bruge den rigtige datatype af enten den ene eller den anden, det dur ikke at blande de to typer sammen.

The Bears

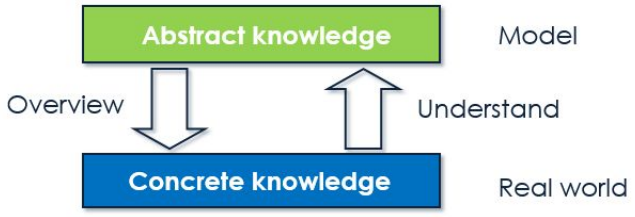
Communication model - The process

We have to create something new through integration of knowledge



Communication model- Process II

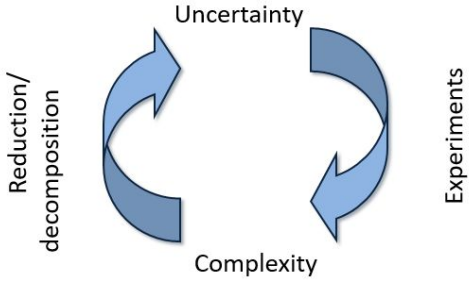
- Abstract knowledge gives us overview
- Concrete knowledge makes/helps us understand abstract knowledge



Abstract knowledge gives us overview, but introduces uncertainty
Concrete knowledge reduces uncertainty, but introduces complexity

The Bears

Principle of limited reduction - combine the two approaches



Spiral paradigm - iterations!

Communication Model - 6 Areas of Knowledge

	Users' present work	New system	Technological opt.
Abstract	Relevant structures	Design proposals	Technological options
Concrete	Experience with the Users' present work	Experience with The new system	Experience with tech. options

What is an experimental approach?

What is an **experiment**?

- Actions where
  - Knowledge/learning is the reason to perform the action
  - All other results are considered irrelevant

What is an **experimental approach**?

- An approach that uses experiments as a central way to obtain knowledge

The Bears

Situations where experiments are relevant

Complexity
- Too much information or a too huge problem

Uncertainty
- Lack of or uncertain knowledge

Pragmatism
- Accept non-perfect solutions
  - Economic reasons

Examples
- User requirement - Horizontal prototypes
- Technological possibilities - Verticals prototypes

When to use experiments

- Experiments are the **only source** to knowledge

  OR

- Experiments are expected to be the **cheapest/fastest** way to get knowledge

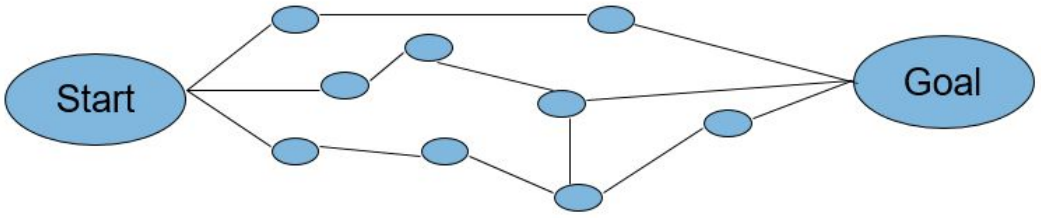
  OR

- You personally find experiments is a **preferable** way to get knowledge

Typer af situationer

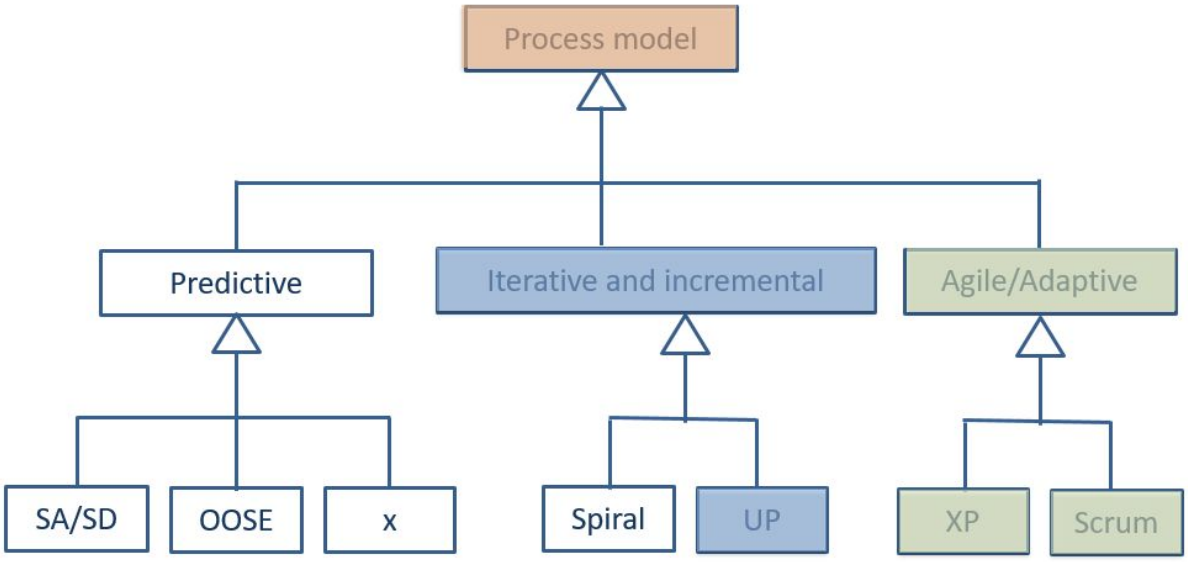
Attributes in Relation to / Situation	Assignment or problem	Working practice	Uncertainty
Routine	Known	Known	Small
Problem solving	Known	Unknown	Intermediate
Problem setting	Unknown	Unknown	Large

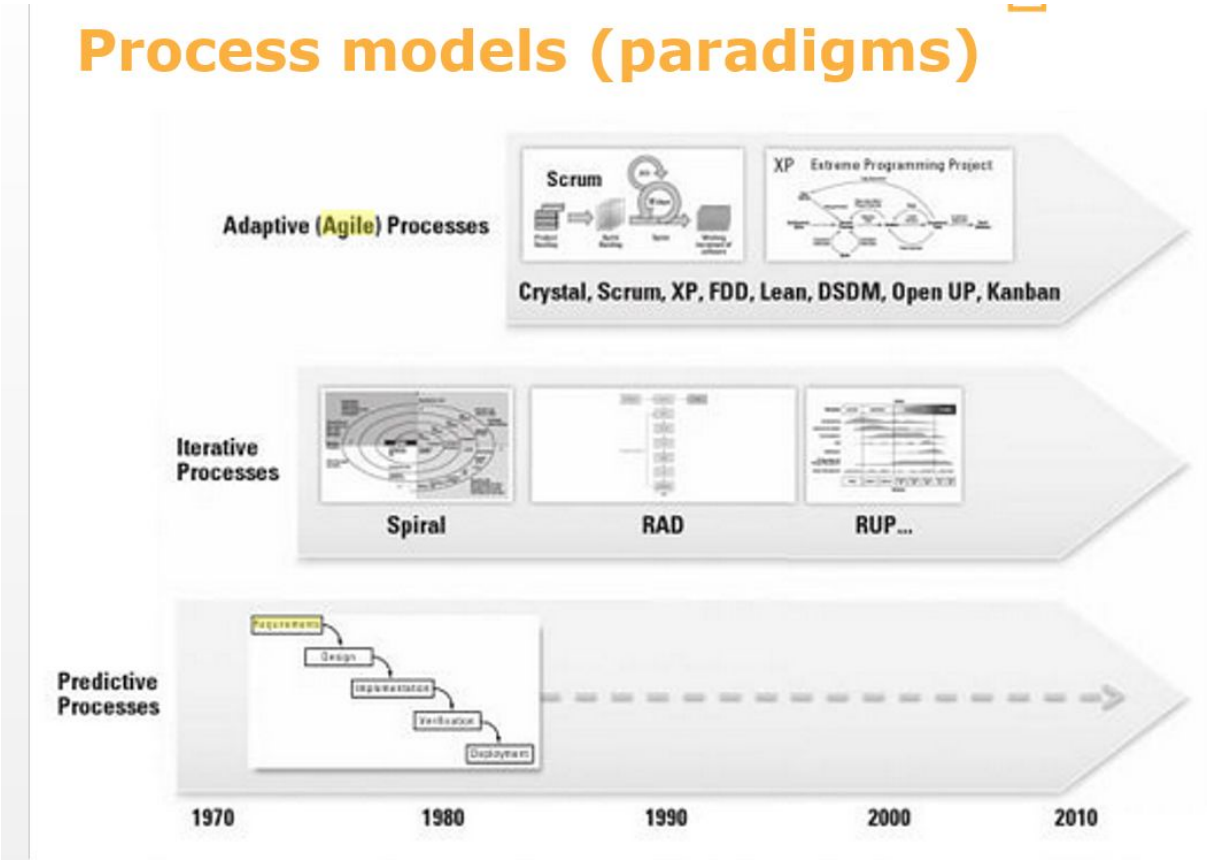
Systemudvikling er en kreativ proces!
Rutinemæssig - Problemløsning - Problemdefinition
I systemudviklingen træffes mange beslutninger



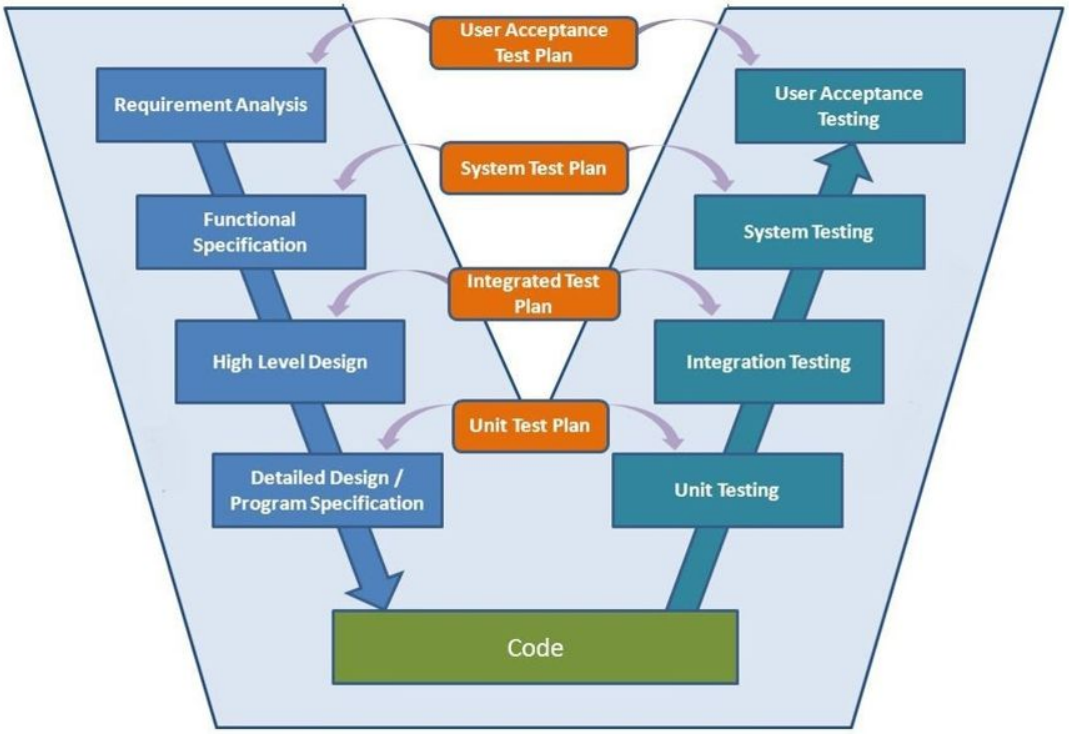
Process modeller


The Bears


Process model
Predictive
Iterative and incremental
Agile/Adaptive
SA/SD
OOSE
x
Spiral
UP
XP
Scrum


Process models (paradigms)
Adaptive (Agile) Processes
Scrum
XP Extreme Programming Project
Crystal, Scrum, XP, FDD, Lean, DSDM, Open UP, Kanban
Iterative Processes
Spiral
RAD
RUP...
Predictive Processes
1970
1980
1990
2000
2010

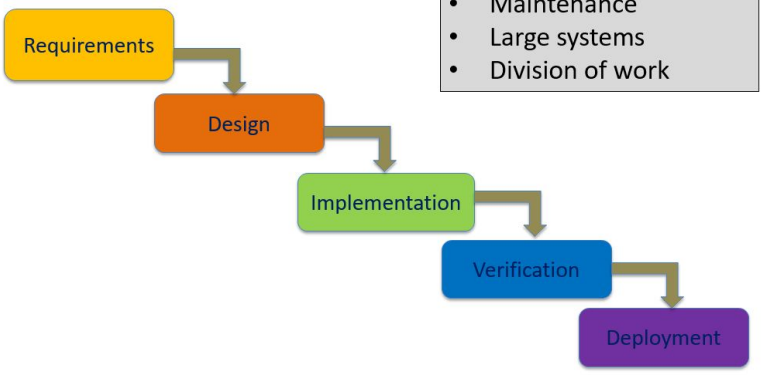
The Bears



Vandfaldsmodellen

Waterfall, predictive, plan driven

- "The devil" is known by many names ☺

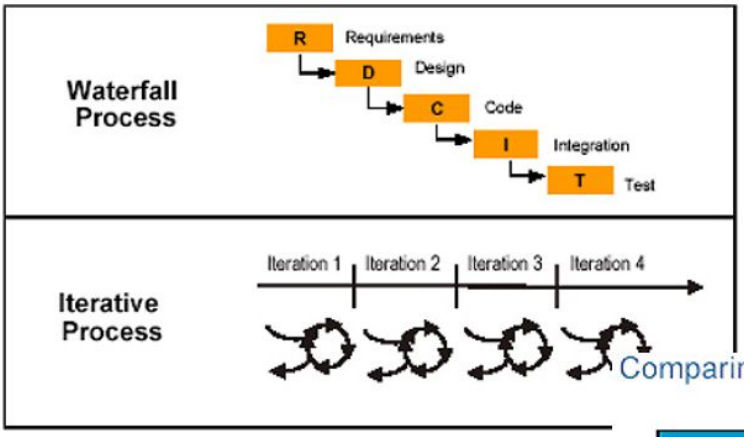


- Known goals/tasks
- Maintenance
- Large systems
- Division of work

The Bears

Waterfall vs. Iterative Process

cphbusiness



There can be feedback loops between these steps, but the dominant practice is that a particular discipline is mostly completed before moving on to the next discipline.

In an iterative process you accomplish a small amount of each of the disciplines (requirements, analysis, design, implementation, and testing) in every iteration

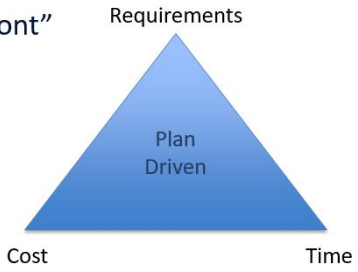
Comparing Waterfall To Iterative

Waterfall	Iterative
Risk averse	Actively attacks risk
Subjective measurement of progress	Objective measurement of progress
Delays integration and testing	Continuous integration and testing
Nothing runs until the end	Something "runnable" produced every iteration
Difficulties at the end of the project	Difficulties at the start of the project

Kilde: http://www.ibm.com/developerworks

Waterfall, predictive, plan driven

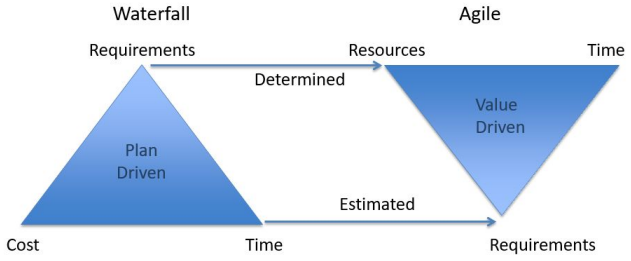
- The process model presupposes **requirements** can be specified ”up front”
- **Cost** and **time** can thus be estimated based on the requirements
- Fixed price contract
- But what about **quality**?

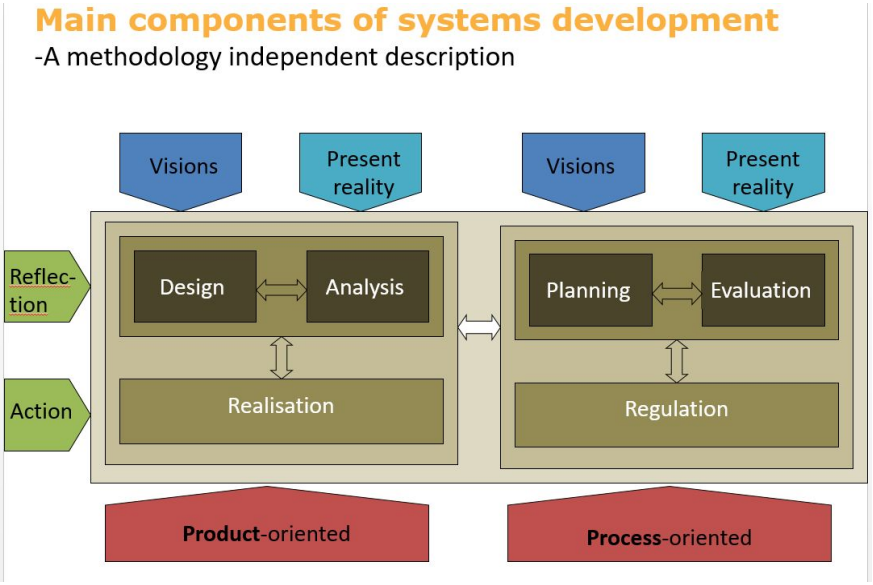


- Better than ”code ‘n’ fix”
- Immediately logical and predictable
- Most software is developed like this
- Fixed price contract is a must

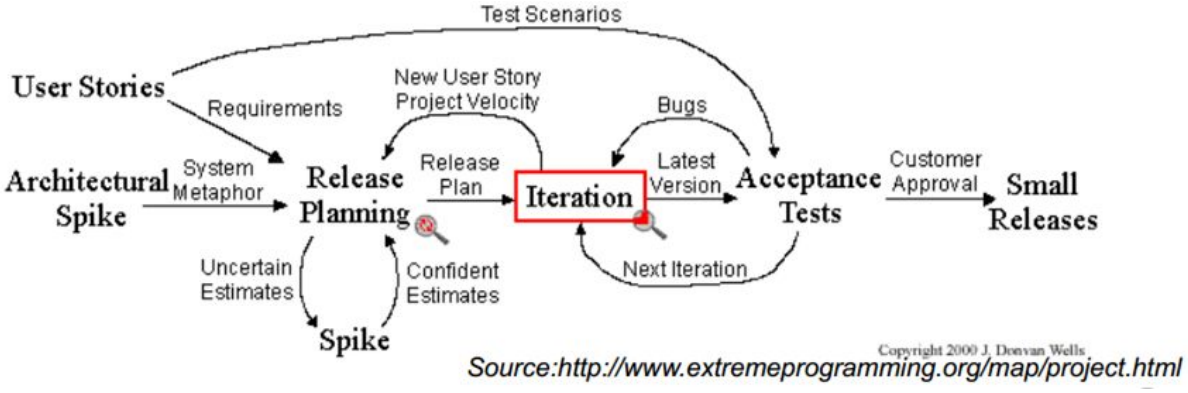
Value driven instead of plan driven

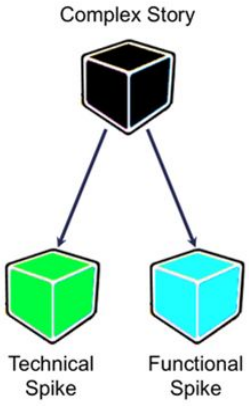
Goodbye to the iron triangle





Spike / Prototype



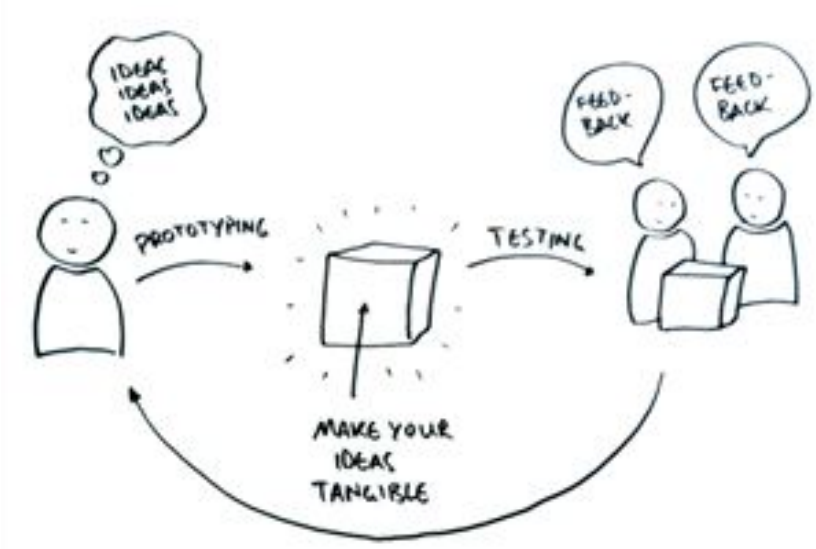


Hvad er en prototype?

Altid en del af den skæve proces i systemudvikling
Hvad skal vi producere?
Hvordan skal vi gøre det?
Kan vi gøre det?

Traditionelt fokuseret på interaktive systemer
Systemer med intensiv brugerinteraktion
Dialog mellem slutbruger og udvikler

Også relevant inden for tekniske områder
Tekniske muligheder
Dialog / videndeling mellem udviklere



Hovedpointer for Prototyper
Prototyping er ikke en systemudviklingsmetode - det er en procedure i næsten alle systemudviklingsmetoder

Anvendelse af prototyper i en samlet proces:
Krav elicitation
Anvendelse af eksplorative prototyper / funktionelle pigge

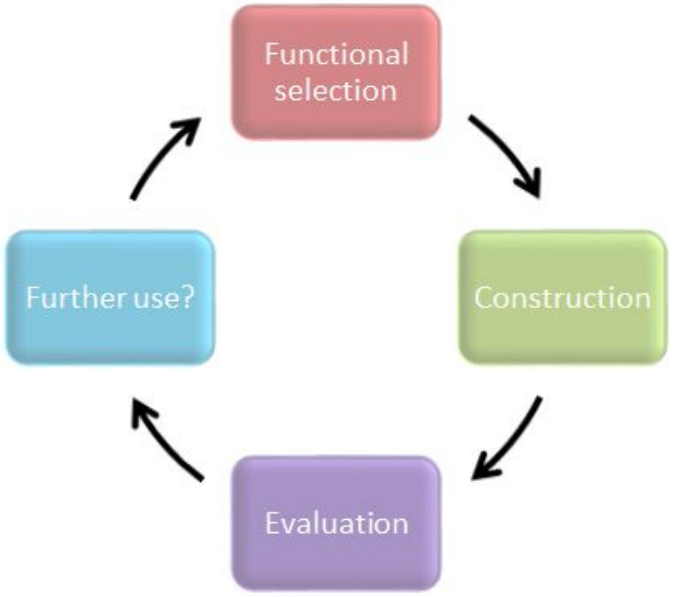
Gå igennem en række eksperimenter, indtil ændringerne er små (stabilitet i krav)

The Bears

Krav er verificeret
Anvendelse af eksperimentelle prototyper / tekniske pigge

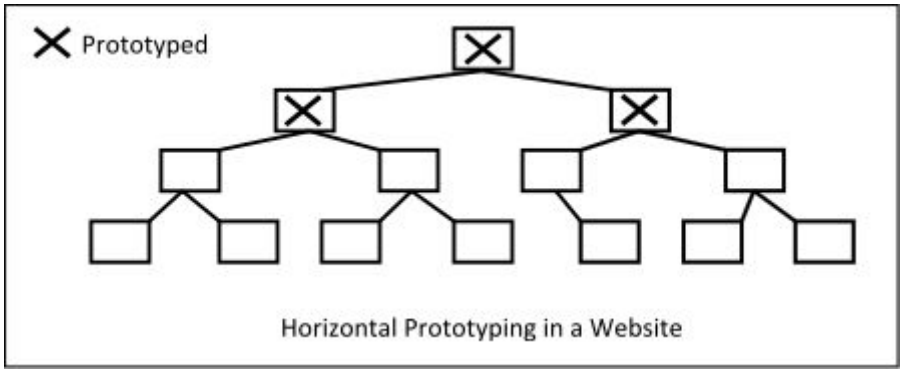
Udvikle systemet ved hjælp af en trinvis tilgang Krav realiseres
Anvendelse af eksperimentelle prototyper.

Prototyping steps
Funktionelt valg
Konstruktion
Evaluering
Yderligere brug

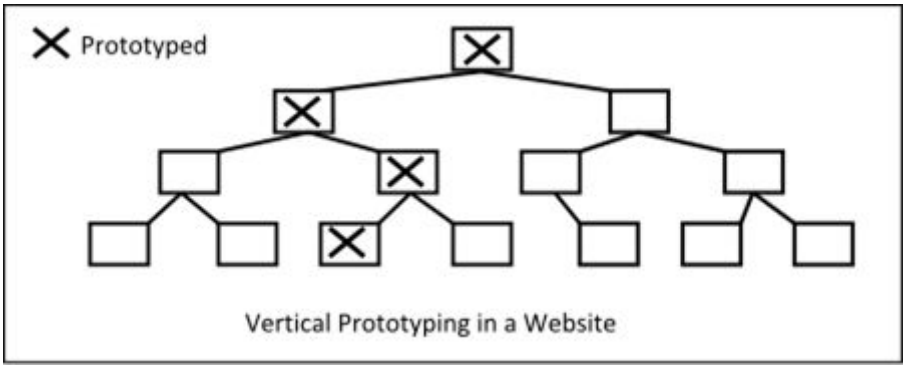


	Advantages	Disadvantages
Low-Fidelity	Lower development cost Evaluate multiple designs Useful communication device Address screen layout issues Useful for identifying market requirements Proof-concepts	Limited error checking Poor detailed specifications for coding Facilitator-driven Limited utility after requirements established Limited usefulness for usability tests Navigation and flow limitations
High-Fidelity	Complete Functionality Fully interactive User-driven Clearly defines navigational scheme Use for exploration and test Look and feel of final product Serves as a living specification Marketing and sales tool	More expensive to develop Time consuming to create Inefficient for proof of concepts Not effective for requirements gathering

Prototyping på et website



The Bears



Paper prototyping

I human-computer interaktion er papirprototyper en udbredt metode i den brugercentreret designproces, en proces, der hjælper udviklere med at skabe software, der opfylder brugerens forventninger og behov - i dette tilfælde især til design og testning af brugergrænseflader.
Det er throwaway prototyper og indebærer at skabe rolige håndskitserede tegninger af en grænseflade til brug som prototyper eller modeller af et design.
Mens papirprototyping virker simpel, kan denne metode til brugervenlighedstest give en hel del nyttig feedback, hvilket vil resultere i udformningen af bedre produkter.
Dette understøttes af mange brugervenlige fagfolk.

Paper prototyping

Pros:

Tillader dig at inkorporere feedback straks
De er hurtige til at skabe
Billig
Tilskynder til samarbejde
Kan anvendes på mange forskellige platforme
Smart telefon

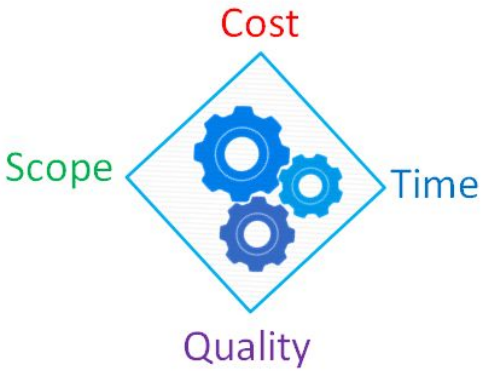
Tablet
Desktop
Smart ur

Cons:

Nogle interaktioner er bare vanskelige at replikere på en elegant måde ved hjælp af papirprototyper.

Det er svært at dele med mennesker, der ikke er fysisk til stede.

Scope, Cost, Time, Quality



Acceptance test

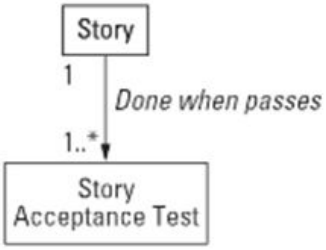
Den vigtigste type af test, som det udføres af kunden / slutbrugerne
Mål om ansøgningen opfylder de påtænkte specifikationer og opfylder kundens krav.
Accepttest er ikke kun beregnet til at påpege simple stavefejl, kosmetiske fejl eller grænseoverskridende huller
Acceptance tests vil også påpege eventuelle fejl i applikationen, som vil resultere i systemnedbrud eller større fejl i applikationen.

Acceptance tests beskriver black-box krav
Identificeret af dine projektinteressenter
Hvilket dit system skal overholde

Accepttest er førsteklasses krav genstande, fordi de beskriver de kriterier, hvormed interessenterne vil afgøre, om systemet opfylder deres behov
De er eksekverbare specifikationer

Acceptance test kaldes også:
Kunde (Acceptance) Test
Funktionalitetstest
"Tilfredsheds Betingelser"

Accepttest oprettes fra brugerhistorier
Under en iteration bliver bruger historierne valgt under iterations planlægningsmødet oversat til accepttest
Accepttesten specificerer scenarier for at teste, når en bruger historie er korrekt implementeret, typisk under sprint revision
En historie kan have en eller flere acceptprøver, uanset hvad det kræver for at sikre, at funktionaliteten virker
Acceptance test er manuelt forretningsmæssigt vendende test
Accepttest er sorte boks systemtest



ATDD - Acceptance criteria

Er et sæt forhold, som historien skal opfylde for at blive accepteret som komplet

Det er ikke en erstatning for samtale med produktets ejer
Det er resultatet af samtalen

The Bears

Format:
Givet [forudsætning] (opsætning)
En specificeret tilstand af et system

Når [Skuespiller + handling] (udløser)
En handling eller begivenhed opstår

Derefter [Observable resultat] (verifikation)
Systemets tilstand er ændret eller en produktion er produceret

Acceptance test
Acceptkriterier er IKKE test
Godkendelseskriterier
+ Eksempler (data + scenarier)
= Acceptance tests

- Check out book: "As a user, I want to check-out a book from the library so that I can bring it home to read"
- **Given**: Book is not checked-out & User who is registered on the system

Books	
Title	Checked out
Great book	No

Users
Name
Sam

- **When**: User checks out a book

Checkout action			
User	Sam	Checks out	Great book

- **Then**: Book is marked as checked out

Books		
Title	Checked out	User
Great book	Yes	Sam

- What if the book is already checked out?
- What if the book does not exist?
- What if the user is not registered in the system?
- Is there a date that the book is due to be checked-in?
- How many books can a user check out?

The Bears

User story	Test case	Precondition	Action	Arguments	Expected behavior	Actual behavior
Check-out book	Check-out book regular - "Happy path"	Borrower is registered Book is not checked-out	n.a.	Borrower name Book title	The book is marked as checked-out by the borrower	

Etablering af accepttest for de spørgsmål, der er rejst på det foregående billede
Lav dine egne forudsætninger om regler, når det er nødvendigt

Brug "Given - When - Then" Format
Og udfyld test sager med data

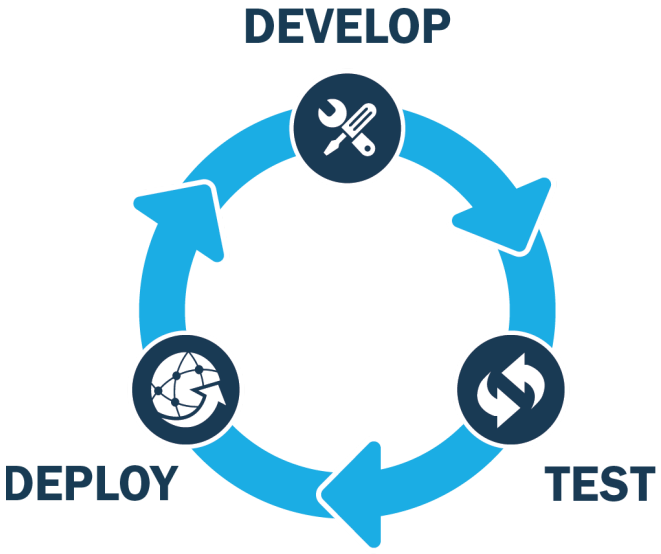
Til slut med accept test

Lad kunden / slutbrugeren udføre testen
Vis ikke produktet selv

Har følgende roller:
En facilitator
Giver instruktioner til brugeren
Uden at påvirke testen
Tilskynder brugerne til at udtrykke deres tanker under testen
Det eneste medlem og udviklingsholdet, der taler under testen
En sekretær (mindst en), der tager noter under testen
Stille observatører for resten af holdet

Continuous integration

Kontinuerlig integration (CI) er processen med at automatisere opbygningen og testen af kode hver gang et teammedlem forbyder ændringer til versionskontrol. CI opfordrer udviklere til at dele deres kode og unit test ved at slå sammen deres ændringer i et delt versionsstyringsregister efter hver lille opgave færdiggørelse. Commits udløser et automatiseret byggesystem til at hente den nyeste kode fra det delte arkiv og at opbygge, teste og validere det til master (github).



Test - TFD / TDD / Mocking / Junit Test

Test er den systematiske proces med at evaluere et system eller dets komponent(er) med det formål at finde ud af om det opfylder de specificerede krav eller ej.
Dvs. at identificere eventuelle huller, fejl eller manglende krav i strid med de faktiske krav.

The Bears

Verification and Validation

Verification	Validation
Are you building it right?	Are you building the right thing?
Ensures that the software meets all the functionality	Ensures that the functionalities meet the intended behavior
Takes place first and includes the checking for documentation, code, etc.	Occurs after verification and mainly involves the checking of the overall product.
Done by developers	Done by testers and end users

System test

Tester systemet som helhed
Når alle komponenter er integreret, testes applikationen som en helhed for at sikre, at den opfylder de angivne kvalitetsstandarder
Vigtigt fordi:
Ansøgningen testes grundigt for at kontrollere, at den overholder de funktionelle og tekniske specifikationer
Ansøgningen testes i et miljø, der ligger meget tæt på produktionsmiljøet
Systemtest gør det muligt for os at teste, verificere og validere både forretningsbehov og applikationsarkitektur.

Integration test

Test af kombinerede dele af en applikation for at afgøre, om de fungerer korrekt
Bottom-up integration
Begynder med enhedsprøvning efterfulgt af test af gradvist højere niveauer af enheder kaldet moduler eller builds

Top-down integration
Modulerne på højeste niveau testes først og progressivt, moduler på lavere niveau testes derefter
I et omfattende software udviklingsmiljø bliver bottom-up testning normalt først udført, efterfulgt af top-down test.

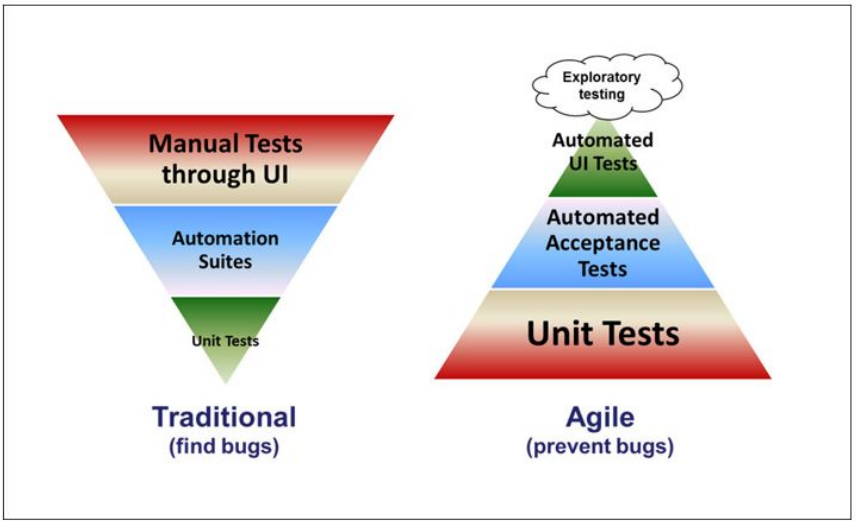
Funktionel test

En type sort-box test, der er baseret på specifikationerne for den software, der skal testes
Ansøgningen testes ved at give input og derefter undersøges resultaterne
Funktionelt test af en software udføres på et komplet, integreret system for at evaluere systemets overholdelse af de specificerede krav.

Steps:
Bestem den funktionalitet, som den tilsigtede applikation skal udføre.
Oprettelse af testdata baseret på ansøgningens specifikationer.
Angiv forventet udgang baseret på testdata og specifikationerne for applikationen.
Skrive testscenarier og de udførte test cases.
Sammenligne faktiske og forventede resultater baseret på de udførte test cases.

Unit test

Målet med unit test er at isolere hver del af programmet og vise, at enkelte dele er korrekte med hensyn til krav og funktionalitet.
Unit test udføres af udviklerne
Begrænsninger:
Testning kan ikke fange hver eneste fejl i en applikation
Det er umuligt at evaluere hver eksekveringssti i alle programmer.



Regression test

Når der foretages en ændring i en softwareapplikation, er det helt muligt, at andre områder inden for applikationen er blevet påvirket af denne ændring
Regressionstest udføres for at verificere, at en fast fejl ikke har resulteret i en anden funktionalitet eller overtrædelse af virksomhedsregel
Formålet med regressionstest er at sikre, at en ændring, som f.eks. En fejlrettelse, ikke må resultere i, at en anden fejl bliver afdækket i ansøgningen
Vigtigt fordi:
Minimér hullerne i testningen, når en applikation med ændringer skal testes
Testning af de nye ændringer for at kontrollere, at de foretagne ændringer ikke påvirker andre områder af applikationen
Testdækning øges uden at gå på kompromis med tidslinjer
Øge hastigheden for at markedsføre produktet.

Non funktionel test

Involver at teste en software fra de krav, der ikke er funktionelle i naturen, men vigtige
Performance Testing bruges mest til at identificere flaskehalse eller præstationsproblemer

The Bears

Load Testing er en proces til at teste adfærd af en software ved at anvende maksimal belastning i form af software adgang og manipulation af store input data
Stresstest omfatter test af adfærd hos en software under unormale forhold
dvs. det kan omfatte at fjerne nogle ressourcer eller anvende en belastning ud over den faktiske belastningsgrænse
Usability testing bruges til at identificere eventuelle fejl og forbedringer i softwaren ved at observere brugerne gennem deres brug og drift (black box-teknik)
UI-test omfatter test af det grafiske brugergrænseflade i form af farve, justering, størrelse og andre egenskaber.

Test dokumentation

Testplan beskriver den strategi, der vil blive brugt til at teste en ansøgning.
Testscenario er en linjeoversigt, der angiver hvilket område i applikationen vil blive testet.
Testscenarier bruges til at sikre, at alle strømme testes fra ende til ende
Testscenarier omfatter flere hvilesager.
Testsager involverer et sæt trin, betingelser og input, der kan bruges, når du udfører testopgaver.
Hovedformålet med denne aktivitet er at sikre, om en software passerer eller fejler med hensyn til dens funktionalitet og andre aspekter.

1. **Test case ID**
2. **Pre-conditions**
3. **Steps**
4. **Expected outcome.**
5. **Actual outcome**
6. **Post-conditions**

The Bears

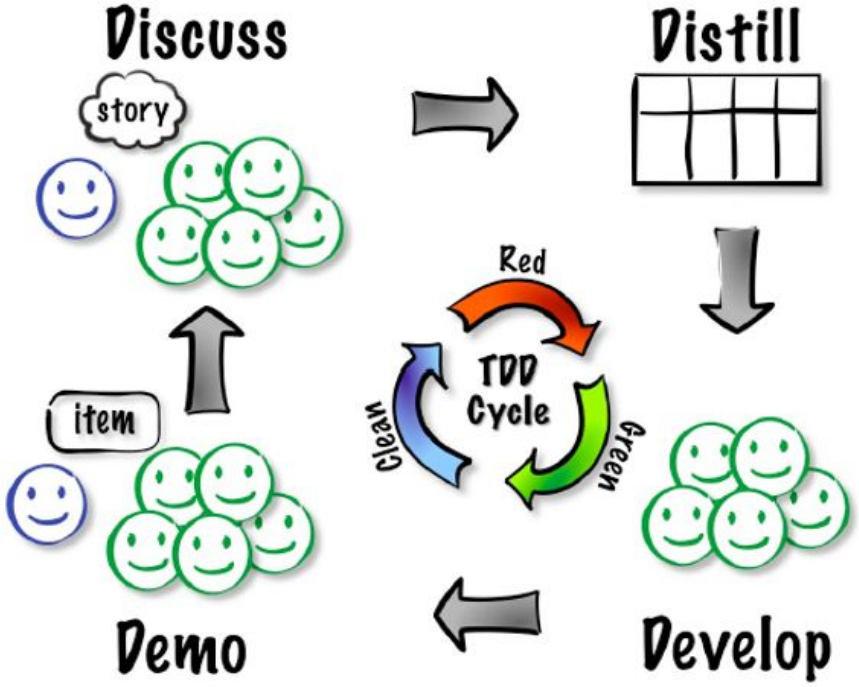
Acceptance Test Driven Development (ATDD) Cycle

En udviklingsmetode baseret på kommunikation mellem erhvervskunder, udviklere og testerne

ATDD omfatter accepttest, men fremhæver skriftlige accepttests, før udviklere begynder kodning

ATDD er tæt relateret til testdrevet udvikling
Det adskiller sig fra fokus på udvikler-tester-business kundesamarbejde

TDD er en programmeringspraksis, ikke en testteknik.
Øvelse af testdrevet udvikling handler om forventninger, der styrer implementeringen
Acceptance Test Driven Development (ATDD) lægger også vægt på tests før kode
I ATDD opretter teamet acceptniveau test for en funktion, før man begynder at arbejde på det.
Disse test er indfanget, når holdet arbejder sammen med forretningsinteressenten for at forstå en historie om efterslæbningen.



Mocks og stubs er to forskellige ting

- Both types replace objects with which a method under test collaborates
- **Mocks** don't implement any logic: They are empty shells that provide methods to let the tests control the behavior of all the business methods of the faked classes
- With mocking we do **behaviour verification**
- **Stubs** provide canned answers to calls made during the test, usually not responding to anything outside the test
- With stubbing we do **state verification**

A. Test without mocking
   - We test with asserts against the warehouse's state after the order has been made
   - State verification

B. Test with mocking
   - We test that the order did the right thing in its interaction with the warehouse.
   - Behavior verification

Mocking – Using Mockito

Mockito har i det væsentlige to faser, hvoraf den ene eller begge er udført som en del af unit test:

Stubbing - state verifikation
Stubbing er processen med at specificere vores mocks adfærd. Sådan fortæller vi Mockito, hvad vi vil ske, når vi interagerer med vores mocks. Stubbing gør det nemt at skabe alle mulige betingelser for vores test. Det lader os styre svarene på vores mocks, herunder at tvinge dem til at returnere enhver værdi, vi ønsker, eller kaste enhver undtagelse, vi ønsker. Det giver os mulighed for at kode forskellige adfærd under forskellige forhold. Stubbing lader os styre præcis, hvad mock vil gøre.

Mocking - adfærd Verifikation
Verifikation er processen med at verificere interaktioner med vores mocks. Det lader os bestemme, hvordan vores mocks blev kaldt, og hvor mange gange. Det lader os se på argumenterne for vores mocks for at sikre, at de er som forventet. Verifikation giver os mulighed for at tage fat på de andre problemer, der er nævnt i første afsnit - det gør os i stand til at sikre, at nøjagtigt de værdier, vi forventer, sendes til vores

samarbejdspartnere, og at intet uventet sker. Verifikation giver os mulighed for at afgøre, hvad der skete med mocken.

Af alle de mange mock-termer, der blev introduceret tidligere, bruger Mockito kun Mock and Spy.

Mock
Med Mock skaber vi en komplet mock eller falsk objekt mock, hvor standardadfærden (som du kan ændre) af metoderne er, gør ingenting
Med en Mock-forekomst kan vi teste både STAT og BEHAVIOR.

Spion
Med Spy bruger vi et rigtigt objekt, og vi spionerer / spotter kun specifikke metoder til det
Spioner bør bruges omhyggeligt og lejlighedsvis, normalt når man beskæftiger sig med arvskode. Bedste praksis er ikke at bruge Spy til delvist at mocke den underprøvede klasse, men i stedet for at delvis hævne afhængigheder.

Klassen under test skal altid være et rigtigt objekt.