Software Engineering Fundamentals

Software Development Processes

Dr. Petru Florin Mihancea

Based on:
I. Summerville - Software Engineering 8, Ch. 4 Software Processes,
Ch. 17. Rapid Software Development
R. Pressman - Software Engineering, Ch. 2 The Process
M. Fowler - UML Distilled, Ch2. Development Process
P. Deemer & G. Benefield - SCRUM Primer

Software (Development) Process

A software process is a set of activities and associated results that produce a software product

Fundamental activities:

1. **Software specification**
   defining the software to be produced
2. **Software development**
   designing and programming
3. **Software validation**
   checking to ensure that it is what the customer required
4. **Software evolution**
   modification to adapt to customer / market requirements

A software process model = a simplified representation of a process

Generic Models

1. **Waterfall**
   separate steps/phases for specification, design, etc.
2. **Iterative/Evolutionary**
   interleaves specification, development, validation activities, etc.
   fast development of an initial system and refine it with the customer
3. **Component-based software engineering**
   process focuses the integration of reusable components in order to build the software

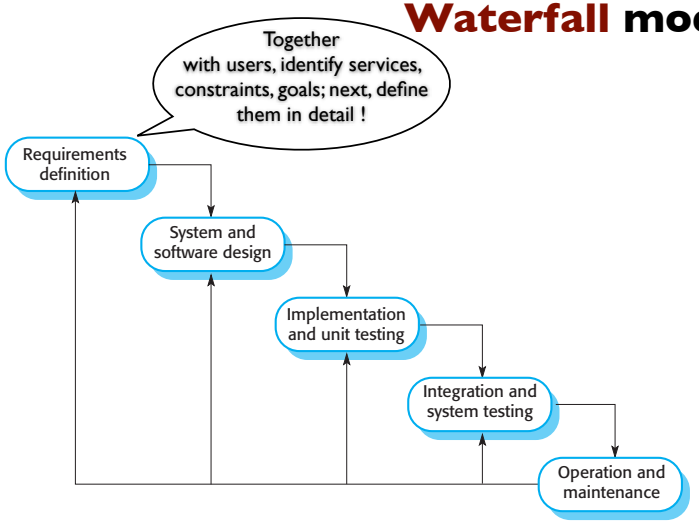
The main contrast

Waterfall	vs.	Iterative/ Evolutionary
breaks down the project based on activity		***usually, breaks down the project by subsets of functionality***
e.g., 2-months analysis phase followed by 4-months design phase followed by 3-months coding phase followed by 3-months testing phase		e.g., 3 iterations 1st iteration takes 1/4 of requirements and perform analysis,design, code, test; when iteration ends, we have a system doing 1/4 of all needed functionality 2nd iteration takes 1/4 of requirements and etc.

1

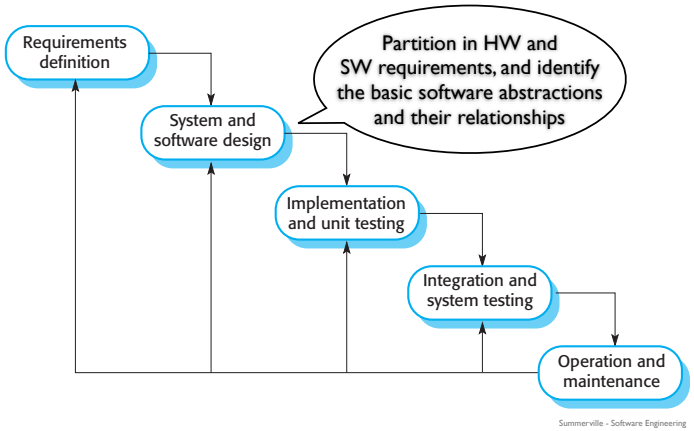
Waterfall model

Waterfall model



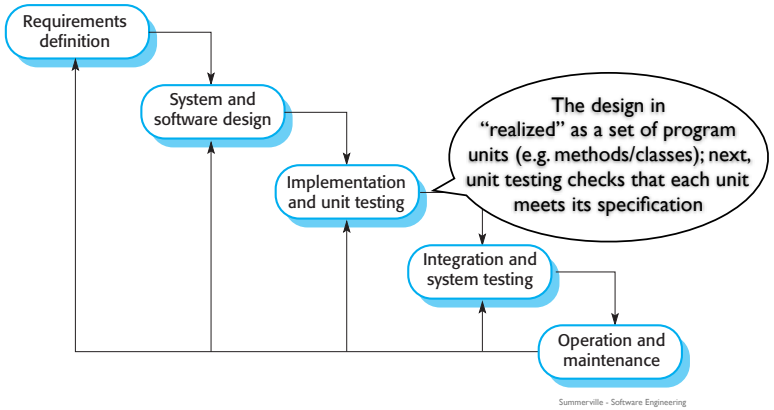
Summerville - Software Engineering

Waterfall model



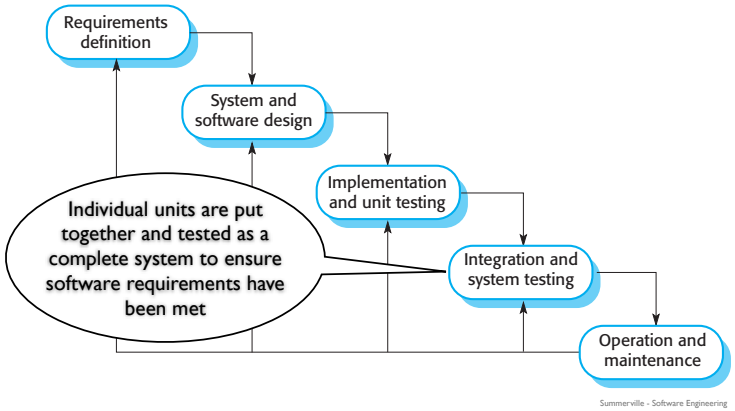
Summerville - Software Engineering

Waterfall model

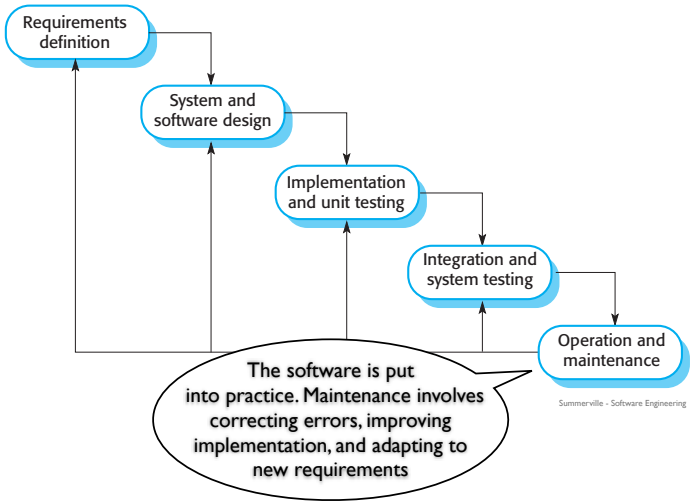


Summerville - Software Engineering

Waterfall model

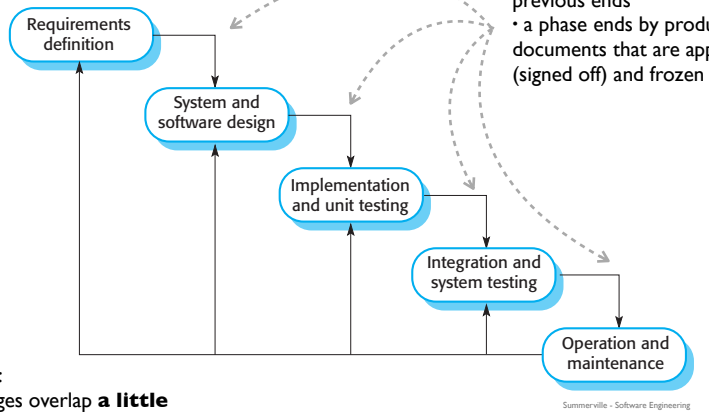


Waterfall model



Waterfall model

In principle:
- a phase starts only after the previous ends
- a phase ends by producing documents that are approved (signed off) and frozen



In practice:
- these stages overlap **a little** and feed information to each other but still, after a short time, the previous phase is **frozen**

Pros/Cons

Pros

Documentation is produced at each phase

Cons

inflexible partitioning of the project into distinct stages

premature "freezing" is dangerous during:

requirements - it is not easy for a customer to explicitly state all the requirements; the software won't do what the user wants

design - leads to bad design and the flaws will be eliminated via programing hacks

commitments (frozen decisions) made at an early stage make very difficult to respond to changing customer requirements

customer must have patience - a working version comes late in the project time-span

When to use ?

only when

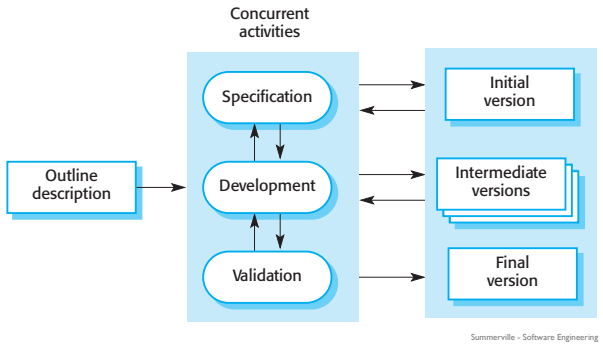
- requirements are well understood
- and unlikely to radically change during development

2

Iterative/Evolutionary models

Two fundamental types ...

Basic idea

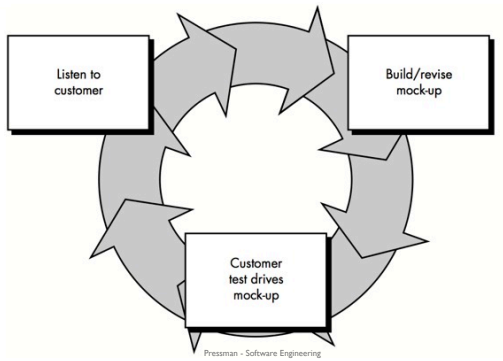


... develop an initial implementation, exposing this to the user and refine it through many versions until an adequate system has been developed

4

Throwaway prototyping

The objective is to understand the customer's requirements and hence develop a better requirements definition



- developer and customer meet and define the overall objectives of software, identify whatever requirements are known
- a quick design occur focusing those aspects visible to the user
- a prototype is built that is then evaluated by the customer to refine requirements, enabling developer to better understand the requirements

The Danger

Both user and developer like the approach

users get quickly a feel of the actual system

developers get to build something immediately

But ...

the users see a “working” version that is held together “with chewing gum and baling wire”; DO NOT transform the prototype into a product by applying “a few fixes” as the customer will beg :)

the developer will make implementation compromises, will neglect long-term maintainability, etc. only to get fast a working prototype

after a while a developer might forget the reason for which some design/ implementation decisions were inappropriate and make them part of the system

The Danger

Both user and developer like the approach

users get quickly a feel of the actual system

developers get to build something immediately

But ...

the users see a “working” version that is held together “with chewing gum and baling wire”; DO NOT transform the prototype into a product by applying “a few fixes” as the customer will beg :)

the developer will make implementation compromises, will neglect long-term maintainability, etc. only to get fast a working prototype

after a while a developer might forget the reason for

... clearly state that the prototype is built to serve as a mechanism for defining requirements ... it is then discarded and another paradigm will be used to build the actual software

Dr. Petru Florin Mihancea
13

B Exploratory development

The objective of this type of processes is to work with the customer to explore their requirements and deliver a final system

start with the parts that are well understood

the system evolves by adding new features proposed by the customer

Pros

specification can be developed incrementally

producing systems that meet immediate needs

Cons

the process is not so visible (from a managerial perspective)

higher risk of poor structure (because of continuos change)

B Exploratory development

The objective of this type of processes is to work with the customer to explore their requirements and deliver a final system

start with the parts that are well understood

the system evolves by adding new features proposed by the customer

Pros

specification can be developed incrementally

producing systems that meet immediate needs

Cons

the process is not so visible (from a managerial perspective)

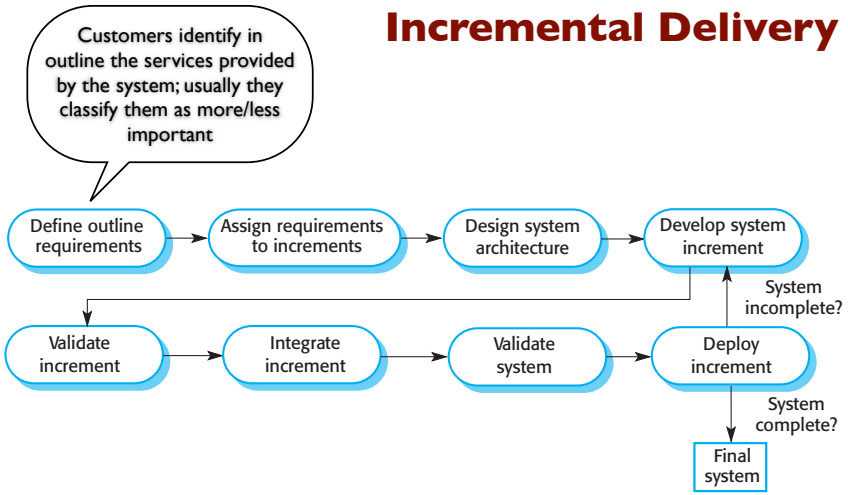
higher risk of poor structure (because of

There are different models/ variations of this kind

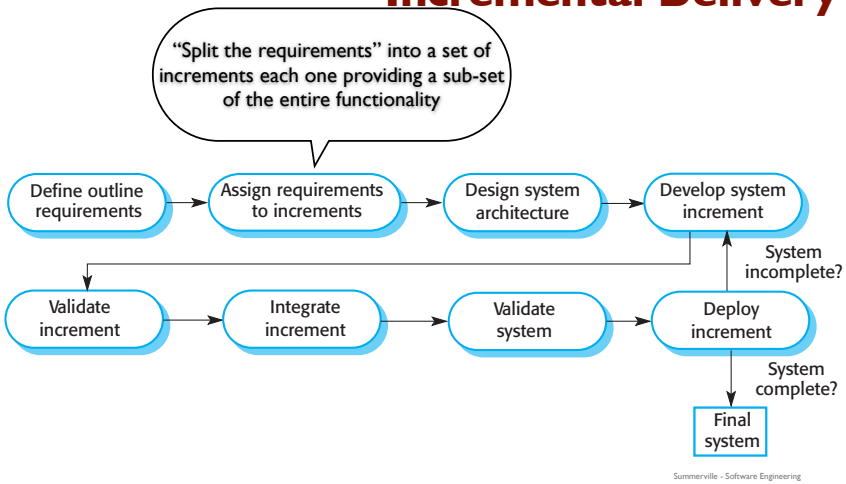
Dr. Petru Florin Mihancea
14

Incremental Delivery

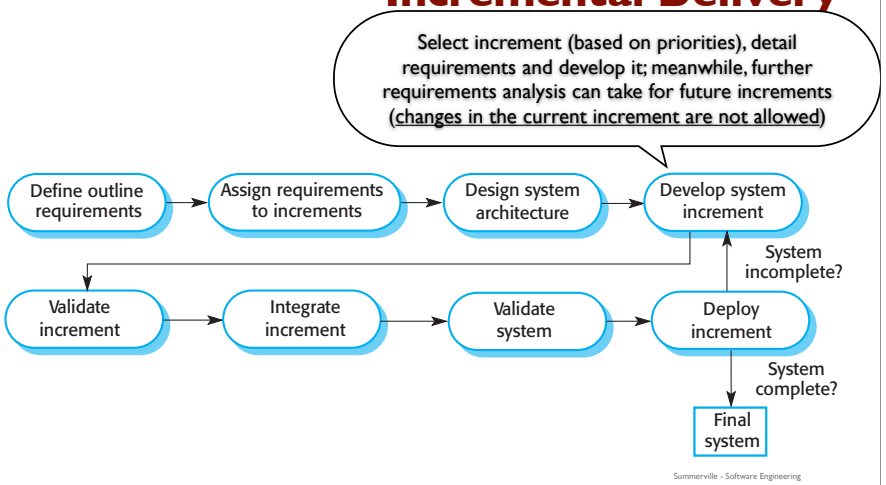
Incremental Delivery



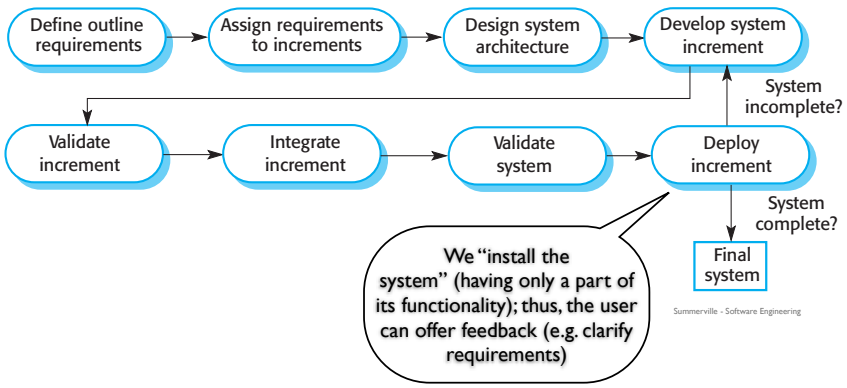
Incremental Delivery



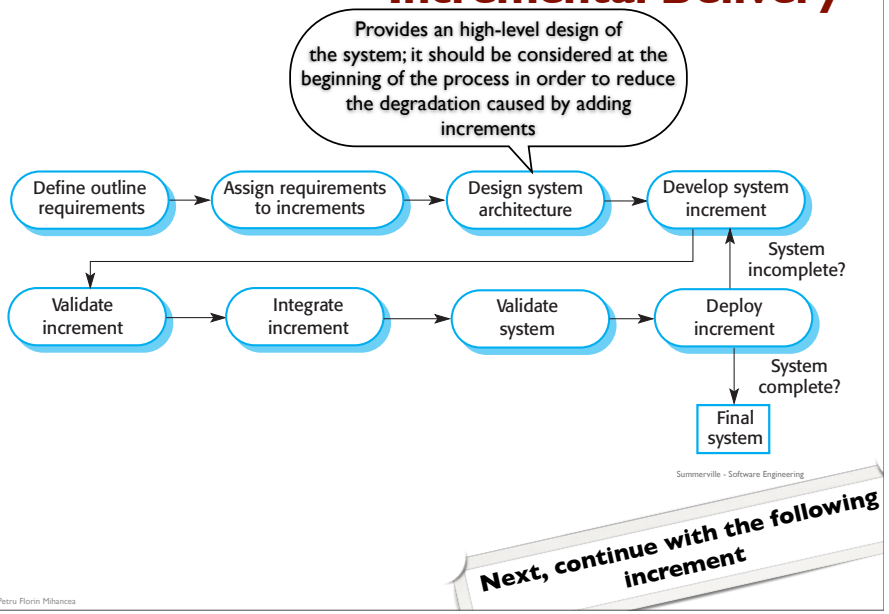
Incremental Delivery



Incremental Delivery



Incremental Delivery



Pros

Can cope with requirements changes

Customers do not have to wait

the increments from the beginning of the process can contain the most important functionality and it can be used by the users

The customer is directly involved

due to user feedback:

- **more likely to meet the desired requirements**
- **new requirements can be discovered / better refinement**

Since increments with higher priority are delivered first

the most important functionality receives the most testing

Cons

Management problems

they like processes that generate regular deliverables to assess progress; it may not be time effective to write documents for very fast iterations

Contractual problems

usually based on specifications; it may be difficult to design a contract where requirements are not fully defined from the beginning

Validation problems

due to minimizing documentation and iterative specification, how do we validate the system ?

Maintenance

continual change tends to degrade the structure of the system; thus special actions must be considered during development to avoid degradation

Difficulties

Rework issues

due to modifications, redesign / deletion (in later iterations) of some code might be seen as a waste; but it is better than having poorly designed / patched code

A set of basic functionalities are common for different parts of the system

it may be difficult to identify them since requirements are not described in detail

Increments should be small and should deliver some functionality

it can be difficult to map requirements onto increment of the right size

Solution for the increment length

Time boxing

iterations must be of a fixed length of time :)

if you can't build all you intended to build in an iteration

... slip some functionality from the iteration and

DO NOT SLIP THE DATE of the iteration

good exercise for learning about requirements prioritization

When NOT appropriate ?

extremely large systems where teams are working in different locations

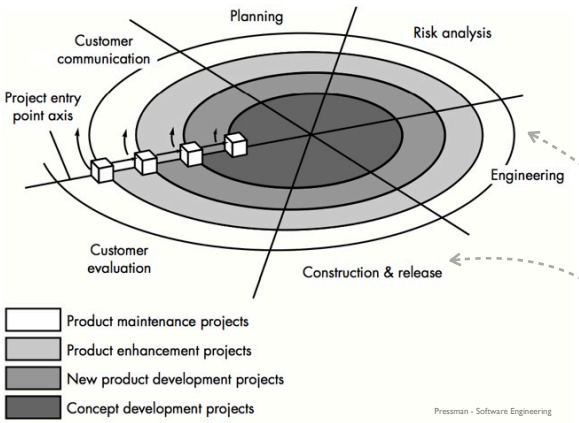
embedded systems where software depends on hardware development

extremely critical systems where all requirements must be analyzed to be able to check for interactions that may compromise safety, security and other critical issues

Spiral Model

Combines the iterative nature of prototyping with the controlled and systematic aspects of waterfall and provides potential for rapid development of incremental versions

Spiral Model

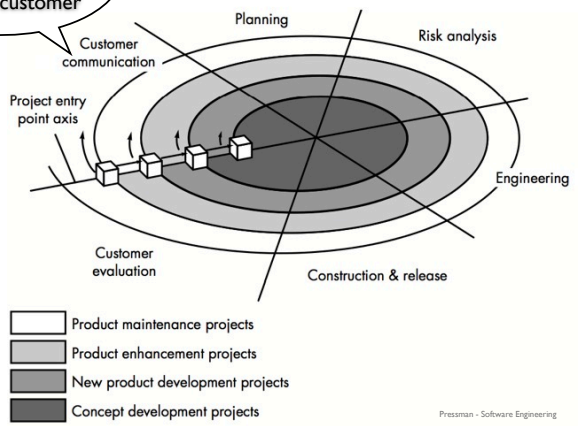


3 to 6 task regions

Pressman - Software Engineering

Spiral Model

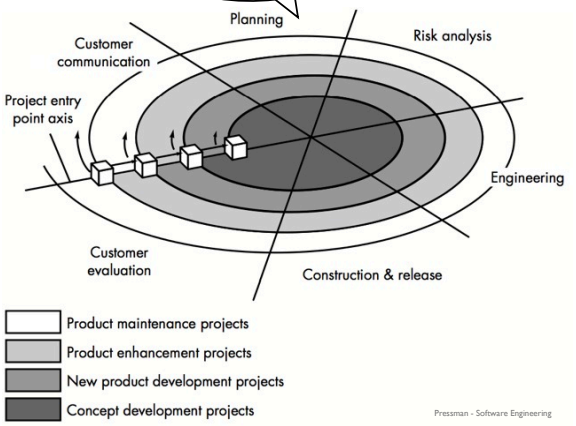
Tasks required to establish effective communications with the customer



Pressman - Software Engineering

Spiral Model

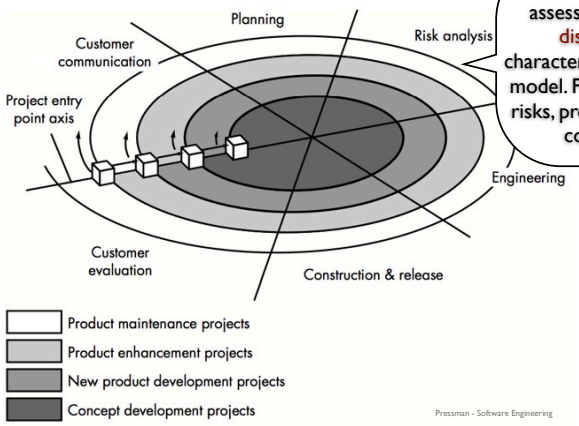
Tasks required to define resources, timelines, etc.



Pressman - Software Engineering

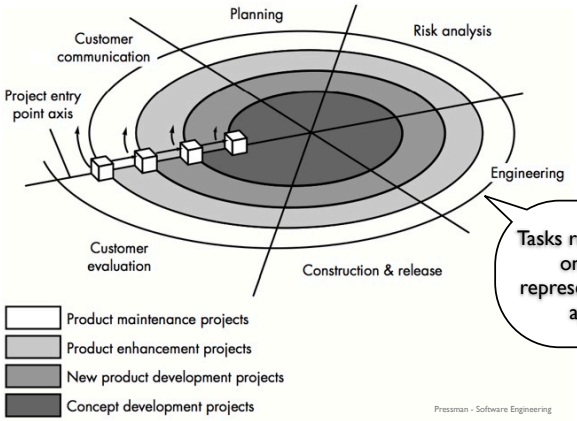
Spiral Model

Tasks required to assess risks. This is a distinguishing characteristic of the spiral model. For requirements risks, prototypes may be constructed



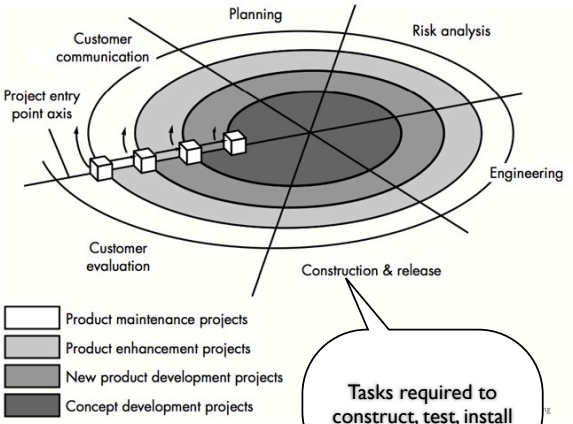
Pressman - Software Engineering

Spiral Model



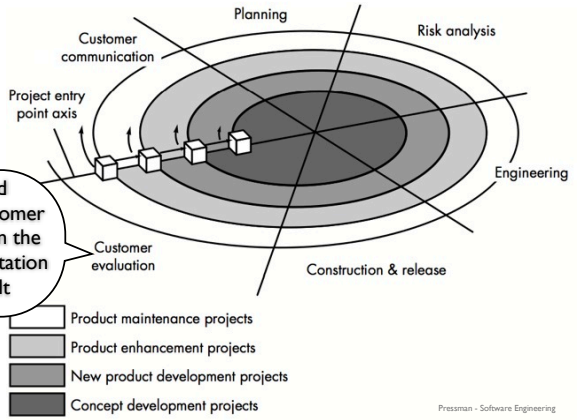
Tasks required to build one or more representations of the application

Spiral Model



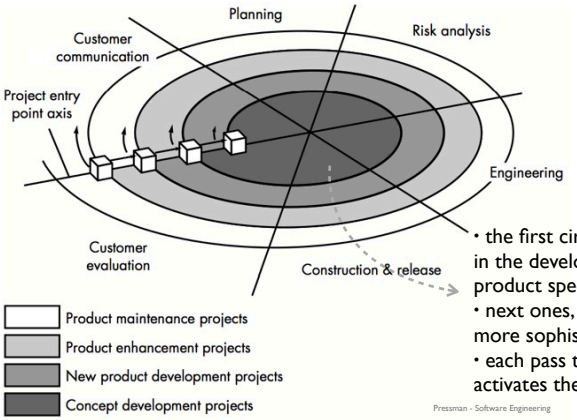
Tasks required to construct, test, install

Spiral Model



Tasks required to obtain the customer feedback based on the software representation previously built

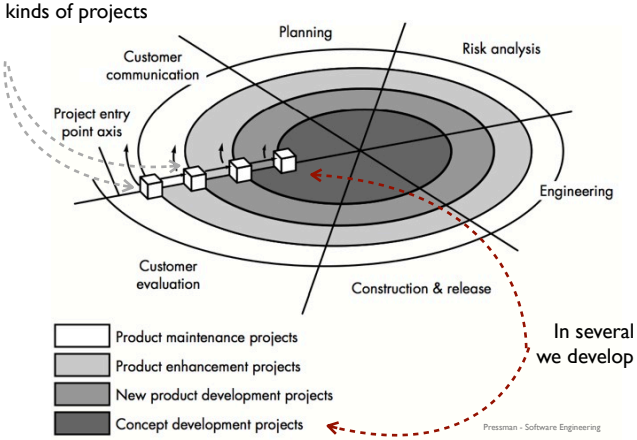
Spiral Model - View I



- the first circuit might result in the development of a product specification
- next ones, progressively more sophisticated versions
- each pass through a region activates the specific tasks

Spiral Model - View 2

Each cube can be used to represent the starting point for different kinds of projects

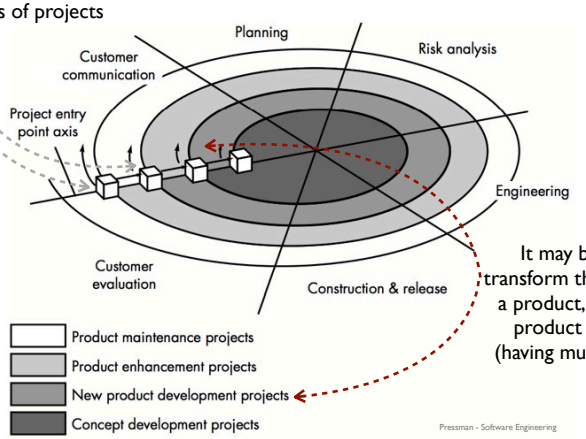


In several iterations we develop a "concept"

Can be adapted to apply through the entire life of software

Spiral Model - View 2

Each cube can be used to represent the starting point for different kinds of projects



It may be decided to transform the "concept" into a product, and thus a new product project starts (having multiple iterations)

And so on, e.g. where a change is initiated, the process starts at the appropriate entry point e.g. product enhancement

Can be adapted to apply through the entire life of software

Pros/Cons

Pros

explicitly consider risks and uses prototyping as a risk reduction mechanism (at any stage in the evolution of the product)

maintains the systematic stepwise approach suggested by the classic life-cycle, but integrates it into an iterative framework

Cons

might be difficult to convince customers that the evolutionary approach is controllable (especially in contract situations)

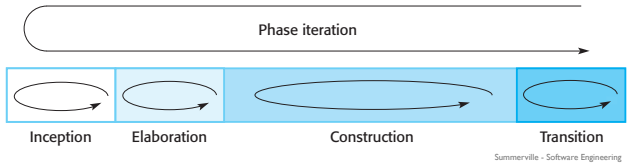
considerable risk assessment expertise

the model has not been as widely used as other approaches, so difficult to discuss about its success :(

Rational Unified Process

Dynamic perspective

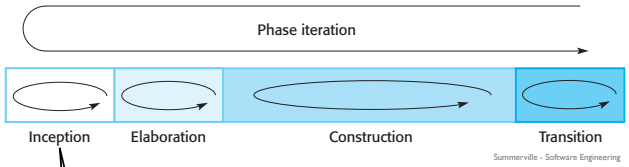
The phases of the process model over time



Somehow, reminds us about Waterfall but ... the phases are related to business concerns (not technical activities)

Dynamic perspective

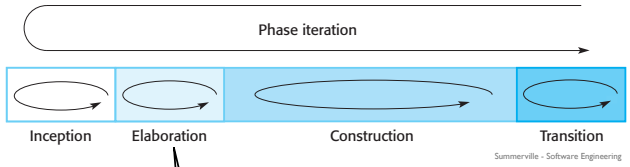
The phases of the process model over time



Establish business use cases for the system. Identify external entities (people, other systems) that interact with the system. Next, assess the contribution of the system to the business (should we continue?)

Dynamic perspective

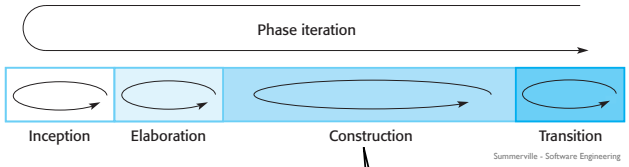
The phases of the process model over time



Understand the problem domain, establish an architectural framework, and the project plan, identify risks. At the end we must have: requirements model, architectural description, development plan

Dynamic perspective

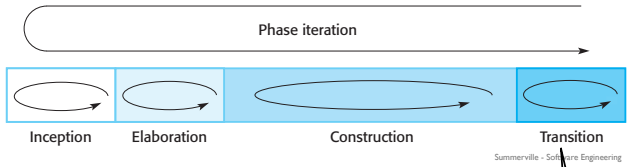
The phases of the process model over time



System design, programming and testing. Parts of the system are developed in parallel and integrated during this phase. At the end, we must have the working system + documentation.

Dynamic perspective

The phases of the process model over time

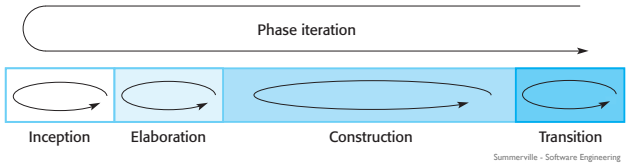


Summerville - Software Engineering

"Move" the system from the developer community to the user community and make it work in the real environment (this is something omitted b other models). At the end: documented and working system in the actual working environment

Dynamic perspective

The phases of the process model over time



Summerville - Software Engineering

Two ways for iterating:
a. the result of each phase can be built incrementally
b. all the phases can be enacted incrementally

Strong connections with UML

Static perspective (1)

The activities (named workflows)

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.

Summerville - Software Engineering

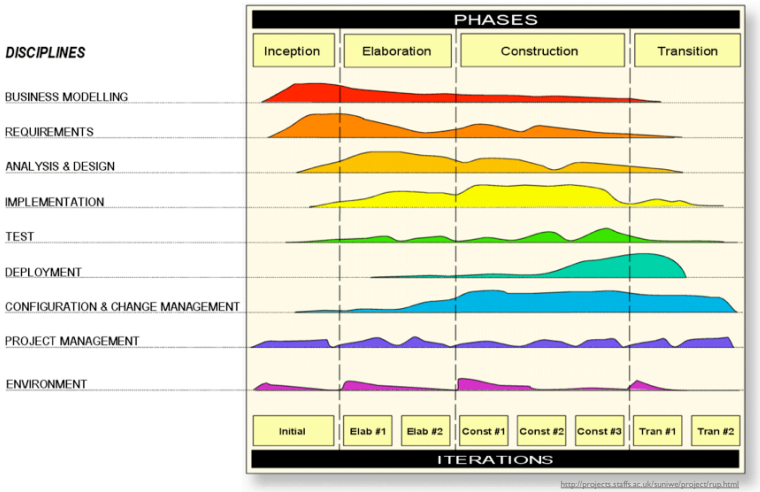
Static perspective (2)

The activities (named workflows)

Workflow	Description
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Summerville - Software Engineering

A combined perspective



Practice perspective

Good practices to be used during RUP

Develop software iteratively

plan increments based on customer priorities and

deliver highest priority increments first

Manage requirements

explicitly document user requirements (we will see use cases)

and keep track of changes

Visually model software

UML e.g. class diagrams, sequence diagrams

Others: Component-based architectures, Verify quality, Control changes.

Innovations

separation of phases and workflows

- **phases are dynamic and have goals**
- **workflows are static and are technical activities that are not associated with a single phase**

recognition that deploying software in the user environment is part of the process

3

Agile Methods

The Beginning

80's - early 90's vision

the best way to achieve better software is through careful planning, formalized quality assurance, the use of analysis methods of CASE tools and controlled and rigorous software development processes

this view came from communities developing

large, longed-lived, critical systems

with geographically dispersed teams, very long development time

e.g. 10 years from specification to deployment for modern aircraft software

Significant overhead (but necessary for such systems) appears due to intense planning, design, documentation, etc.

The Problem - Dissatisfaction

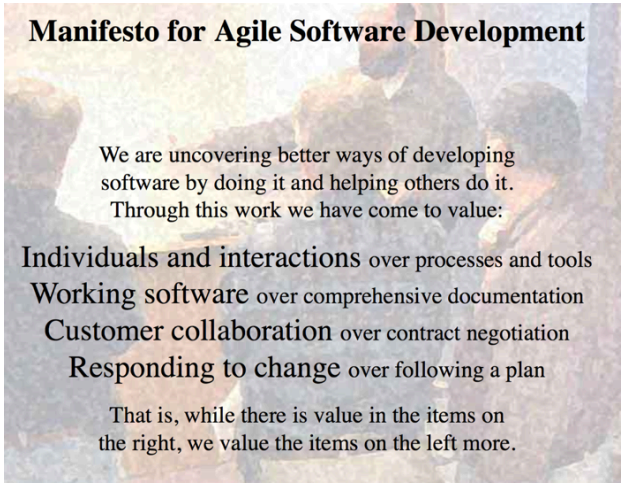
for small and medium sized business software the overhead became so large that it sometimes dominated the development process

Proposed solution

the agile methods developed in the 90's

allow teams to focus on the software itself rather on design and documents

Agile Manifesto



http://agilemanifesto.org/

Principles

1. **People not processes**

   The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive process

2. **Incremental delivery**

   Software is delivered (rapidly) in increments with the customer specifying the requirements to be included in each increment

3. **Customer involvement**

   Closely involved in the development, providing new and prioritizing requirements and evaluate each iteration

4. **Embrace change**

   Expect the requirements to change (even in case of rapid changes during development or very late changes)

5. **Maintain simplicity**

   Focus on simplicity in software and process; work to eliminate complexity from the system

Difficulties

The need of a customer who is willing and able to spend time with the development team

Prioritizing changes can be difficult

Maintaining simplicity requires extra work

Contractual problems - the customer pays for the time required for development and not for a specific set of requirements

When ?

probably best suited for small to medium sized software (without critical aspects that may affect security and safety and thus, requiring a previous detail analysis of all the requirements)

Extreme Programming (XP)

Basics

Pushes good practices (e.g. iterative delivery) and customer involvement to "extreme"

new versions may be built several times a day

increments are delivered to customer roughly every 2 weeks

all tests must be run for every build and must be successful; only in that case the build is accepted

Story Card

The form in which requirements are captured; it encapsulates a customer's need from the system
(usually in terms of an interaction scenario with the system)

Downloading and printing an article
First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it—this can either be through a subscription, though a company account or by credit card. After this, you get a copyright form from the system to fill in. When you have submitted this, the article you want is downloaded onto your computer. You then choose a printer and a copy of the article is printed. You tell the system printing has been successful. If the article is a print-only article, you can't keep the PDF version, so it is automatically deleted from your computer.

Summerville - Software Engineering

Practices (A)

1. On-Site Customer

A representative of the end user should be available full time for the use of XP team (thus it is actually a member of the team) responsible for defining acceptance tests, involved in specifying and in prioritizing requirements (i.e. various story cards)

2. Incremental planning

The stories developed in a particular release are determined / selected based on the time available and story priorities

3. Small releases

The minimum set of functionalities providing useful value from the system is developed first; following releases are frequent and add functionality to the first/previous release

As requirements change, unimplemented story cards may change or may be discarded; more over, new cards may appear and re-prioritization may occur

A more detailed story card

Date: _____ Story Nr: _____	Type of Activity: New ☐ Fix ☐ Enhance ☐ Funct Testing ☐ Priority: User _____ Technical _____ Risk: _____ Technical Estimate: _____
Story Name:	
Story Description:	
Notes:	

Task Tracing

Date	Status	ToDo	Comments

A story is split into tasks

Task 1: Implement principal work flow

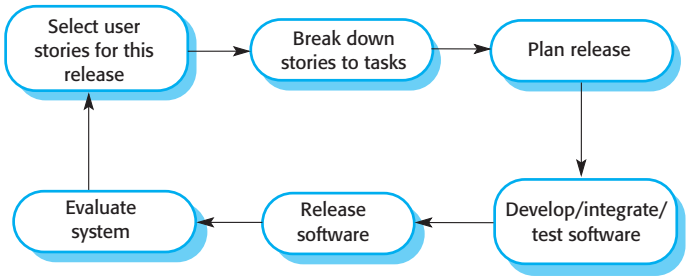
Task 2: Implement article catalog and selection

Task 3: Implement payment collection

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

Summerville - Software Engineering

XP Release Cycle



Summerville - Software Engineering

Practices (B)

4. Simple design

Enough design is performed in order to meet the current requirements and no more - e.g via UML in sketch mode :)

5. Refactoring

Frequent code change tends to degrade the internal structure of the application

Consequently all developer are expected to refactor/reorganize the code asap when the possibility of improvement is found; thus the code should remain simple and maintainable

6. Test-first development

An automated unit test framework is used to write tests for a new piece of functionality before that functionality is implemented

Testing in XP

Since we do not have a full system specification an external team cannot develop system tests

To overcome this issue

A. Test first-development

Writing tests first defines the interface and the specification of the behavior that is going to be implemented

Can help clarify requirements i.e. in order to write a test you must clearly understand the specification

Avoids the tendency to skip testing in order to maintain the schedule

B. Incremental test development from scenarios

Stories are split into tasks each of which represents a single feature of the system; unit tests can be derived then for each task

Example

Task 1: Implement principal work flow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

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

Summerville - Software Engineering

Test 4: Test credit card validity

Input:
A string representing the credit card number and two integers representing the month and year when the card expires
Tests:
Check that all bytes in the string are digits
Check that the month lies between 1 and 12 and the year is greater than or equal to the current year.
Using the first 4 digits of the credit card number, check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expire date information to the card issuer
Output:
OK or error message indicating that the card is invalid

Summerville - Software Engineering

Testing in XP (2)

C. User involvement in test development and validation

The user must help to produce the acceptance tests i.e. tests with user data to check that the application meets the user stories

D. Automated testing mechanisms

Usually, a test will be an executable component simulating the submission of the inputs to the tested entity and checks that the output meets the entity specification

Whenever a new functionality is added, the tests can be run quickly and problems introduced by the new code can be caught immediately

Difficulties

Many programmers do not like writing tests

Some tests might be extremely difficult to write before the system is implemented

The user commitment to the tests development (how do we know that her tests are sufficiently complete ?)

Practices (C)

7. Pair Programming

Very simple, programmers work in pairs; they actually sit together at the same workstation :)

8. Collective ownership

The pairs work in all areas of the system; all the developers own all the code; anyone can change anything; no islands of expertise develop

9. Sustainable pace

Large amounts of overtime are not acceptable

Advantages of Pair Programming

Supports the idea of common code ownership

The code is owned by the team as a whole

Individuals are not responsible for problems, the entire team is responsible

It acts as an informal review process

Each line of code is looked at by at least two people

Not as good as a formal review process, but cheaper and faster

Support for refactoring/restructuring

Restructuring is a long term investment and takes time; thus, an individual practicing refactoring might be seen as less efficient

Due to pair programming and common code, everybody gains from refactoring and it is more likely to support the process

Isn't this a waste of resources ?

There is some evidence that productivity of pair programming is comparable with that of two people working independently

The pair members discuss various solutions so they might have fewer "false starts" and thus, less rework

Errors might be avoided because of the informal review of the code

Practices (D)

10. Continuos Integration

As soon as the work on a task is completed it is integrated into the whole system; after any such integration, all the unit tests must pass

Keeps the team in sync avoiding painful integration cycles

To achieve this:

The entire building of the application (i.e. compiling, copying resource files into place, etc.) should be an automatic process (and ideally, should be fast)

Automated testing mechanism are required

The developers must be announced immediately when some tests fail (e.g. via email)

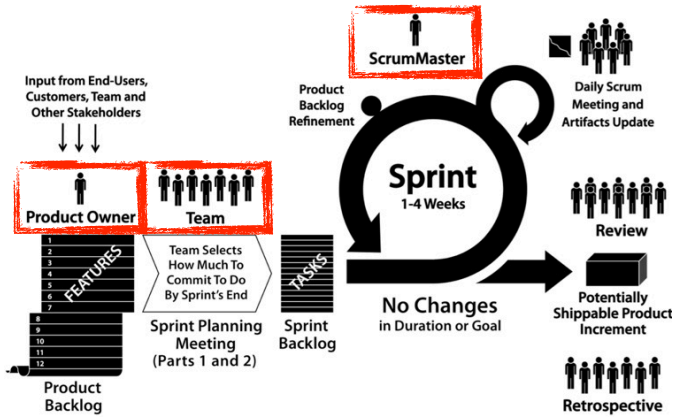
There are dedicated CASE tools to support continuos integration

B

SCRUM

http://en.wikipedia.org/wiki/File:Scrum.jpg

SCRUM



P. Deemer, G. Benefield - Scrum Primer Version 1.2

Iterative & incremental agile process

the cycles are named sprints and are timeboxed

Roles

Product Owner

One person

Identifies product features, prioritize them, decides which of them must be in the top of the list for the next sprint, and continuously re-prioritize and refine the list

Prioritization must be made to maximize the "profit" (e.g. revenues, highest-value-lowest-cost features, satisfy important clients, etc.)

Sometimes may be directly the customer (for internal applications)

The Team

7 (+/- 2) people

Builds the product indicated by the Product Owner

100% dedicated to one product during a sprint (avoid multiple-project involvement)

Cross-Functional - includes all the necessary expertise (analysis, development, testing, database design, etc.)

Multi-skilled members (but they not have to be all generalists)

Self-Organizing - the team decides what to commit to in a sprint and how to accomplish that commitment

Roles (2)

Scrum Master

He is not the manager of the team (e.g., does not tell people what to do, does not assign tasks, etc.)

Helps the group learn, apply and reinforce Scrum rules
(e.g, pushback the Product Owner if he tries to add new deliverables in the middle of a sprint)

May be an active member of the Team (in the case of small teams) but cannot be the Product Owner

Entering Scrum

Product Owner constructs and continuously updates a prioritized list of items called the Product Backlog

Item
- **(new) customer features**
- **engineering improvements goals (e.g., solve a scalability issue)**
- **known defects**

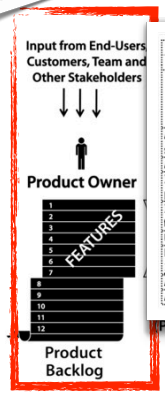
Items are usually expressed using use cases or user stories

The subset of items intended to be solved for a particular release of the product forms the Release Backlog

Entering Scrum

No recipe for estimation
Estimate in terms of relative size using a "point" as unit
After several sprints the Team can find how much can do in a sprint (e.g., 26 points/sprint)
When stable this is the team velocity

Product Owner sets priorities
e.g., highest-value-lowest-cost



Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	New Estimates of Effort Remaining as of Sprint... 1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to to add or delete items on my wish list	...	8	4	20						

(Parts 1 and 2)

Retrospective

P. Deemer, G. Benefield - Scrum Primer Version 1.2

Product Owner (sometimes helped by the ScrumMaster) sets the value of an item

Team provides an estimate of the effort to solve the item

Sprint Planning Meeting

Part 1
Product Owner + Team
- **review highest-priority items in Product Backlog (e.g., goal, context, etc.)**

Part 2
Team (Product Owner is not required)
- **selects the items that the team commits to complete by the end of the sprint (starting from the top of the Product Backlog)**
- **the Team may lobby for a particular low-priority item when it fits easily with a high-priority one**
- **based on estimating the available working hours (e.g., usually 4-6 h / day / member for sprint related-work, but also must consider vacations, etc.)**
- **based on discussing/designing/etc. the top items, the Team breaks them into tasks for each of which it estimates the required time to complete**

The Sprint Backlog

You can hang it on a wall using Post-It for tasks and use different columns to mark progress

Product Backlog Item	Sprint Task	Volunteer	Initial Estimate of Effort	New Estimates of Effort Remaining as of Day... 1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

P. Deemer, G. Benefield - Scrum Primer Version 1.2

NO item is added to the Sprint Backlog during the sprint

Positive influence of the Team feeling protected, knowing exactly its objectives

Product Owner is really thinking on the item prioritization and knows what the Team committed to do

Product Owner can work on updating the remaining of the Product Backlog

People volunteer for tasks one at a time, when it is time to pick a new one

The task status should be marked: Not started, In Progress, Completed

Sprint - Daily Scrum

15 minutes daily meeting - each member is standing and reports:

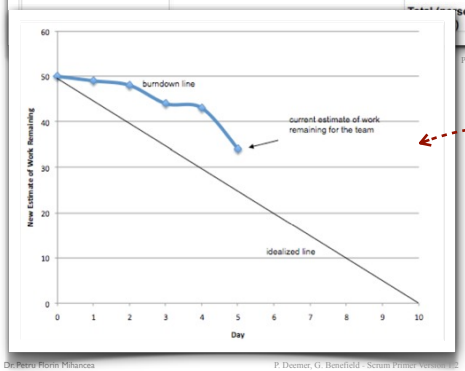
1) What I get done from the last meeting ?
2) What I try to finish by the next meeting ?
3) Found impediments

- **opportunity for synchronization, self-coordination and obstacle report**
- **ScrumMaster must react to blocking situations to try to solve them (e.g., maybe during a follow-up meeting)**

P. Deemer, G. Benefield - Scrum Primer Version 1.2

Sprint - Updating Sprint Backlog

Product Backlog Item	Sprint Task	Volunteer	Initial Estimate of Effort	New Estimates of Effort Remaining at end of Day... 1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5	4	3	0	0	0	
	create webpage (UI)	Jing	3	3	3	2	0	0	
	create webpage (Javascript logic)	Tracy & Sam	2	2	2	2	1	0	
	write automated acceptance tests	Sarah	5	5	5	5	5	0	
	update buyer help webpage	Sanjay & Jing	3	3	3	3	3	0	
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5	5	5	5	5	5	
	complete machine order for pRank		3	3	8	8	8	8	
	change DCP and reader to use pRank http API		5	5	5	5	5	5	
	. . .		. . .						
			50	49	48	44	43	34	



Each day, members estimate the amount of time until the task is done

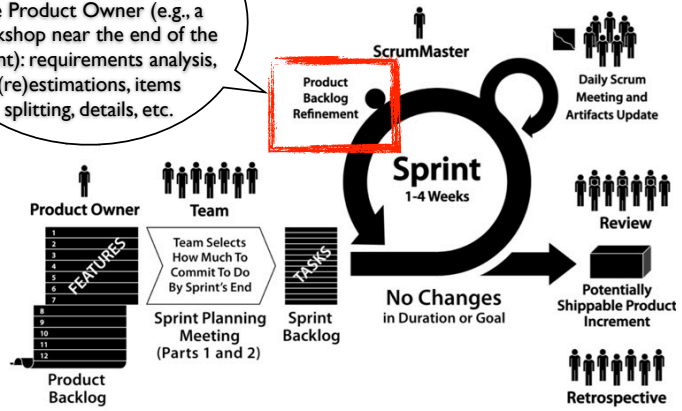
Sprint Burndown Chart

The Team can easily observe if the goal of the sprint can be met in the remaining time or adjustments are required (e.g., slip functionality)

P. Deemer, G. Benefield - Scrum Primer Version 1.2

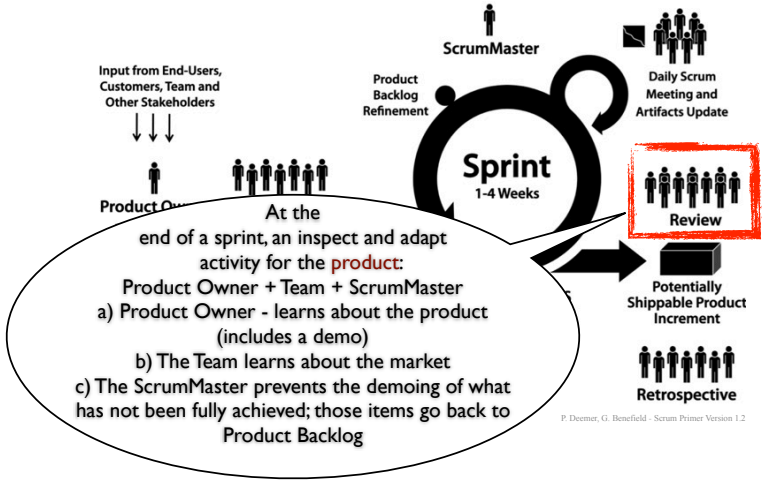
Sprint - Other elements

5-10% of the sprint, the Team should refine the Product Backlog with the Product Owner (e.g., a workshop near the end of the sprint): requirements analysis, (re)estimations, items splitting, details, etc.



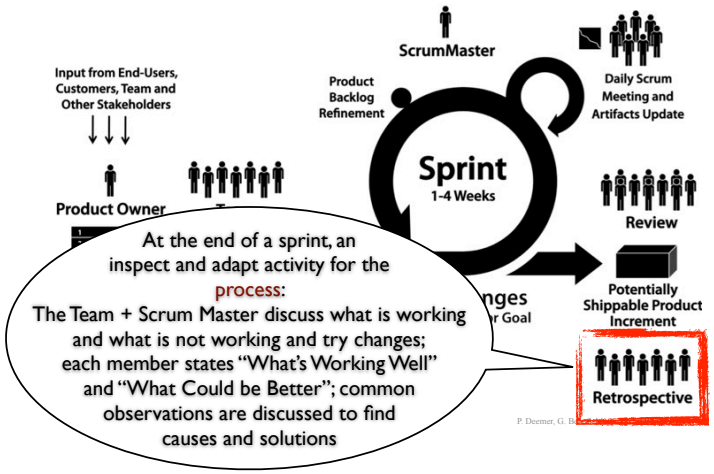
P. Deemer, G. Benefield - Scrum Primer Version 1.2

Sprint - Other elements



P. Deemer, G. Benefield - Scrum Primer Version 1.2

Sprint - Other elements



Other elements

The product must be shippable at the end of each sprint

In practice

- **An actual release will contain more items (i.e., Release Backlog Items) developed in several sprints**
- **The Release Backlog must be maintained in a similar manner like the Sprint Backlog, together with a Release Burndown Chart**
- **A final release sprint might be required before a release**
  - **Perfect vision of shippable product after each sprint is hard to achieve**
  - **Will contain final integration testing, etc.**
  - **Continuos refactoring, integration and effective testing in each sprint should reduce the necessity of such an iteration**