

WUP

Web Unified Process

Author: Sander Nagtegaal / Central

Version	Author	Date	Description
1.0	Sander Nagtegaal	23-11-2000	De geboorte van Web XP/WUP
1.1	Sander Nagtegaal	26-1-2001	Wijzigingen mbt stories/project/whiteboard/in het kort.
1.2	Sander Nagtegaal	8-2-2001	Wijzigingen mbt prototyping, FO, TO
1.3	Sander Nagtegaal	17-4-2001	Implementatie WAE
1.4	Sander Nagtegaal	20-4-2001	Toevoegen UML diagrams
1.5	Sander Nagtegaal	26-4-2001	English version
1.6	Sander Nagtegaal	27-5-2001	Added project phases
2.0	Sander Nagtegaal	22-7-2001	Release 2.0 / design parts etc.

Index

Introduction	4
1 The outlines of WUP	5
1.1 The aim of the method.....	5
1.2 The roots	5
1.3 The specifications	6
2 The WUP team.....	9
2.1 Who's doing what	10
2.2 The functional decisionmaker	11
2.3 The project manager.....	11
2.4 The developers.....	12
2.4.1 Technical team	12
2.4.2 Design team	12
2.4.3 Interaction designer	13
2.4.4 Text writers	13
3 The tools of WUP	14
3.1 Domain Classes	14
3.1.1 The determination of the classes	14
3.1.2 An example of a Domain Class Diagram	15
3.2 Requirement description.....	15
3.2.1 Use cases	15
3.2.2 Use case packages	16
3.2.3 Stories	16
3.2.4 Story estimation.....	17
3.2.5 Speed of realisation.....	17
3.2.6 Priority	17
3.2.7 Notation of use case and stories.....	18
3.2.8 Use case diagrams	18
3.3 Sequence diagrams.....	19
3.4 System requirements	20
3.5 Design requirements	20
3.5.1 Navigation map.....	20
3.5.2 Design brief.....	21
3.5.3 Design template.....	21
3.5.4 Design Comps.....	21
3.5.5 Interaction design and prototyping	21
3.5.6 Design production planning.....	22
3.6 Risk	22
4 The project	22
4.1 Project flow	22
4.2 Conceptual Phase	24
4.3 Analysis Phase.....	25
4.3.1 Functional design	25

4.3.2	Initial design.....	25
4.3.3	Interaction design and prototyping	26
4.3.4	Technical design	27
4.3.5	Planning increments	27
4.4	Development Phase	28
4.4.1	Stand-up meetings and the whiteboard.....	29
4.4.2	Database- and stored procedure scripts	30
4.4.3	Releases.....	30
4.4.4	WUP Programming.....	31
4.4.5	Testing.....	32
5	Suggestions	32
6	Reporting project progress	33
7	Conclusions	33
8	Acknowledgements.....	33
9	Documents that you will probably need.....	34
10	Resources	34
10.1	Books and white papers	34
10.2	Sites	35
11	References	35

Introduction

Web application development grows rapidly towards regular IT. However, the main difference will stay: the web is all about communication, and human communication depends heavily on the use of emotion. As we all know, the strength of traditional IT development does not lay in creative design...

This dependency on emotion and creativity directly leads to a trade-off that bothers every web development company. How on earth can you mix the free spirit of the web with the effectiveness that modern application development asks for?

Web functionality can hardly be planned for, deadlines are neglected, budgets are overestimated and results do not fit the needs of a client: this sounds like an average Internet project. However, the technical development method described in this document could prevent those problems, by using the creative and flexible nature of the web and the structural capabilities of more traditional development.

WUP (Web Unified Process) is a method for building web applications in a professional environment. The method uses UML (Unified Modeling Language) for its notation.

The WUP method guides developers and projectmanagers into a more structural and flexible way of building web apps that suit the needs of the client. It differs from traditional processes in being much more practical, simple and directly designed from a web point of view.

This document starts with determining the outlines of the method, after which the details are described.

1 The outlines of WUP

WUP (Web Unified Process) is a method for building web applications. The method uses UML (Unified Modeling Language) and where applicable the UML WAE (Web Application Extension) for its notation.

The method tries to establish an iterative project rhythm, in which certain task scenarios are performed repetitively according to a plan. The development of an application is segmented into time periods called increments.

A description of the problem domain is the start of each project. The behaviour of the future system is driven by UML use cases.

1.1 The aim of the method

In general, a development method is meant to guide the activities of a team of developers. Especially in projects that are relatively complicated, things tend to be forgotten or underestimated. A structural method can guide the less experienced members of the team and even the more experienced developers.

As a result, every member of the team has a clear view of the order of tasks he has to perform, which means that everybody knows when who is going to do what and why he is going to do that.

The process has the main responsibility to establish a rhythm, which helps to build project inertia. In practice, this inertia will keep up the flow or development speed of the project even in hard times. The method also offers criteria for monitoring and measuring the project's process.

The WUP tries to provide a flexible approach for building web applications, without all the hassle of traditional methods. The process tries to prevent the creativity from being structured to death.

Most of all, WUP is practical. If you own a small web development company and want to get started with a project right away, just read the WUP manual and use the template documents.

1.2 The roots

The described process is partly based on iterative, incremental methods like Dynamic Systems Development Method (DSDM) and Extreme Programming (XP). These processes are widely used in usual IT projects, but for building web applications a more flexible approach is recommended.

The Rational Unified Process and the ICONIX Unified Process are also resources on which WUP is based. These methods are driven by UML, although not especially designed for web app development.

It's a matter of fine-tuning: a method can't be equally useful for regular IT and for web projects at the same time. Therefore, the WUP uses parts of the four methods to create a web-compliant way of working. Apart from this combination of methods, the WUP method is also based on extensive experience in the web development practice.

So, WUP does not pretend to be a completely new development method, but is merely meant to be a frame in which the web-compliant parts of existing methods and UML are put together. This results in a structural development environment in which both coders and designers can be creative.

When using WUP, please feel free to adjust the process to the specific needs of your company. No development method will exactly suit your own process, but it can provide some guidance.

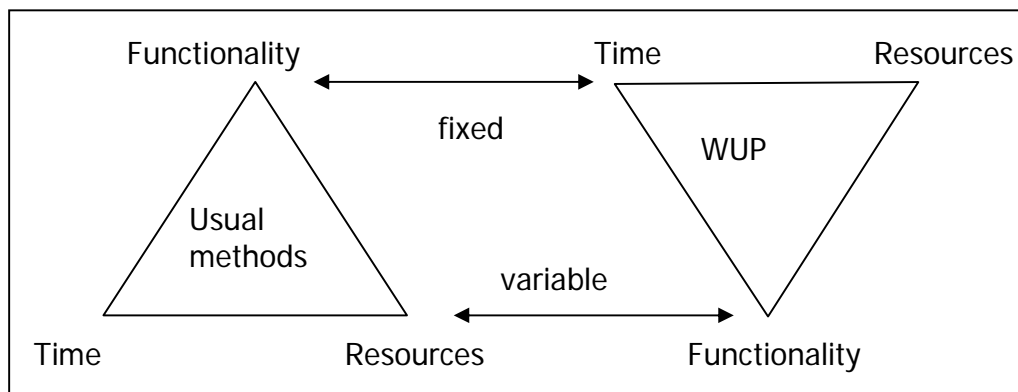
1.3 The specifications

The method is meant to serve the larger web projects which are depending heavily on technical logic. The process relies on a strong architectural foundation.

This kind of projects is preferably object oriented (OO) and usually 3 tier. In a 3 tier architecture the software is strictly divided into three parts: the presentation tier, the business logic tier and the data tier. The separation of these parts leads to a system that is more scalable and easier to maintain.

The WUP approach supports development in small teams best. The developer team should consist of a projectmanager, two or more developers and at least one functional decisionmaker. If the number of persons in a team exceeds, let's say, twelve, it would be advisable to create more roles than the usual WUP role definition accounts for.

This team aims at delivering some functionality within a fixed time period, which satisfies the needs of the customer. The deadline is fixed, but on the fly the optimal functionality can be redefined. The project can be functionally steered, depending on the progress of the project and the needs of the customer.



This way of dealing with time, resources and functionality is called *timeboxing*. Timeboxing means effectively that time and money are fixed, but that functionality varies.

Every phase of the project is dealt with as a time box. There are usually four phases in an average web project: the conceptual phase, the analysis phase, the development phase and the post-deployment phase.

In the conceptual phase, a business concept is created. A concept describes the functionality of the future system on a marketing level, usually by marketing people, in no more than a couple of pages in writing. In the analysis phase the developers and the customer try to fill in the gaps in functionality of the concept, to create an initial functional design, a style definition, an interaction design and an initial technical design. In the development phase people actually start coding, testing and releasing. The final product is supposed to be delivered at the end of this phase. The post-deployment phase can include beta testing, performance tuning, maintenance and user training.

Programming is usually more succesful if done in steps, with each step focussed on a limited readily achievable objective. Therefore, the development process runs iteratively. The steps or iterations are planned using the feedback on business value and priority of the requirements that the functional decisionmaker supplies. There will be small releases for every independent part of the system, to give an oppportunity for that feedback.

A very complex system requires an incremental approach. An increment is a period after which a workable version of the system is released. After finishing an increment, the functional decisionmaker and the customer can decide whether and how to continue. Within an increment the development always runs iteratively.

The start of a project is characterized by the determination of the problem domain. The entities of the problem domain are defined and used to construct a Domain Class Model according to the UML standard. From this model the object classes and data model are derived.

The external behaviour that the system should exhibit is described in *use cases*. Use cases are small units of functionality, describing the interaction between an actor outside the system and the black-box system itself. Use cases are easy to read, for both customer and developers.

The functional decisionmaker defines the use cases. After that, the developers and the functional decisionmaker determine the actions inside the system that will lead to the completion of the use case. These actions are called *stories*. The stories are prioritized and estimated in complexity. The individual developers sign in on the realization of a specific use case and its stories.

The link between use cases and stories is described in more detail using sequence diagrams. These can be modeled using the WAE (Web Application Extension) for UML¹.

Use cases and stories supply an easy tool for projectmanager and functional decisionmaker to monitor and measure the progress of the project. Since the time needed to realize stories is estimated, it is very easy to estimate the total amount of days needed to finish the project. The duration of a project can be calculated using the team members' individual capabilities or development speed.

Apart from use cases, the system must conform to technical system requirements. These constraints are typically expressed as a statement that begins with: "The system shall..".

2 The WUP team

The team should consist of a project manager, two or more developers and at least one functional decisionmaker. The developers can be technical developers, designers or text writers.

These are the people that make the analysis, development and most of the post-deployment phase of the project work. The conceptual phase, however, can be taken care of by marketing people, although the developer team might well be capable of doing this also and definitely needs to be included in the process.

If there are more than two developers of a discipline, it is recommended to designate one of those developers to be the team leader of that discipline. This person is preferably a senior developer.

The functional decisionmaker is, or functions as, the customer. He is responsible for determining the business requirements and has to take all the decisions that have something to do with the business rules.

The requirements of a project are described in the Functional Design. One team member is designated to be responsible for this Functional Design, preferably the functional decisionmaker, or at least someone who continuously keeps in touch with him.

Everyone has to perform a specific role, but it's important to stress that they're all in the same team and share the same goal.

2.1 Who's doing what

The team members have specific tasks. These tasks are listed below. They are defined based on the key assumption that business people make business decisions, and technical people make technical decisions. The project managers balance power between developers and business people.

Who	How many	What
The functional decisionmaker	1	Determines the requirements
		Prioritizes the requirements (what to do first)
		Defines functional tests
Who	How many	What
The developer	At least 1, preferably more	Analyses requirements
		Designs
		Programs, designs, writes
		Tests
		Integrates the system
		Estimates the difficulty of use cases and their stories
		Estimates the pace of development and delivery of the product
Who	How many	What
The project manager	1	Brings functional decisionmaker and developers together
		Remove obstacles
		Monitors and measures the progress of the project
		Reports

The team members have specific tasks as well as rights.

The functional decisionmaker and project manager have the right:

- to have an overall plan (to know what can be accomplished, when and at what cost);
- to change their minds about requirements and priorities (within a timebox);
- to be informed of schedule changes.

The developers have the right:

- to know what is needed, with clear view on priority;
- to ask for and receive help from senior developers and functional decisionmaker;
- to make and update their own estimates.

2.2 The functional decisionmaker

The role of functional decisionmaker is defined based on the key assumption that business people make business decisions, and technical people make technical decisions.

The functional decisionmaker is the person who defines the business priorities and requirements. He must be someone who knows the problem domain or stays in close contact with someone who does.

Preferably, the functional decisionmaker is a representative of the customer. If the size of the project justifies this, see to get a permanent customer representative on-site.

If this isn't possible, try this:

- Try to get someone to represent the customer locally;
- Try to get the customer on-site at meetings;
- Go and visit the customer often;
- Release code frequently to the customer;
- Expect to have misunderstandings!

In practice, it can be handy to have a functional designer have the role of functional decisionmaker. For projects that do not have a direct customer, this is the way to go. For projects that do, it is key that the functional designer has a permanent communication possibility with the real functional decisionmaker at the customer's place.

2.3 The project manager

On a WUP project, the project management role is very important, but it's very focused on management per se.

Most importantly, the success of the project manager depends on removing anything from the team's path that doesn't contribute to the objective of delivering a good web app on time.

Looking out for things that are slowing down the team, expediting purchases, organizing meetings, reporting results; the project manager is merely the person who facilitates the team's process. He's also the one who deals with the financials and all that stuff that a developer doesn't care about.

It is very important for a WUP project manager to know what not to do. When it comes to the process of technical planning, designing, testing, coding, releasing: managers don't do these things directly. The project manager causes these things to be done, coordinate their doing, and report the results.

These tasks are time-consuming as it is, and means that a project manager can't be the functional decisionmaker (which might well be a fulltime job). Apart from the time issue, a project manager has completely different motivations than the customer. The project manager is focused on effectiveness of the process, rather than on business priorities. The business requirements and priority have to be determined by the expert, not by a busy manager.

2.4 The developers

The size of the team of developers obviously depends on the scope of the project. If a project is large enough, a team could consist of the following members.

2.4.1 Technical team

A technical team should consist of coders and at least one technical senior, who will act as the technical team's leader.

The technical constraints and the detailed system design are documented in the Technical Design, preferably initially constructed by the senior. All developing team members are responsible for this Technical Design during the development phase. However, one person is designated to keep an eye on the developers' efforts to keep the Technical Design up to date during the project. This person is preferably the technical team leader, but not necessarily.

Another technical team member checks the database and stored procedure scripts every week, according to a specific procedure that is described later in this document.

2.4.2 Design team

A design team should consist of designers and at least one design senior, who will act as the design team's leader.

The design process will be described later in this document.

2.4.3 Interaction designer

Functional design and technical design do not satisfy the needs of the developers. What we miss here is an interaction design, which describes an intuitive page flow and where to find which specific part of the functionality.

To supply this, we need an interaction designer. This is a person who is perfectly able to read the functional design, and can talk with the technical team to verify whether his ideas are technically possible. He must be experienced in using web interfaces.

However, he can't be the same person that made the functional design: a functional designer perfectly knows the functionality: any interface will do, as far as he is concerned. This implies that the functional designer can't design an intuitive interface anymore. The same goes for technical persons: they know how systems work, which can't be said for the end user of the system. Therefore, an interaction designer must not be too technical.

2.4.4 Text writers

Depending on the scope of the project and the agreements with the customer, text writers are needed to supply content.

Text writers for the Internet are heavily underestimated: in the end, content is what will attract the end user of the web system.

However, it is common practice that the customer itself takes care of the content. Larger companies obviously want their own people to fill and update public web sites.

3 The tools of WUP

Before digging deeper into the process itself, let's define the WUP toolkit first.

3.1 Domain Classes

An entity is a functional unit that acts in the problem domain for which the web app is being build. In object orientation, the entities are translated in classes of the system. The Domain Class Diagram is the UML model in which these classes are shown.

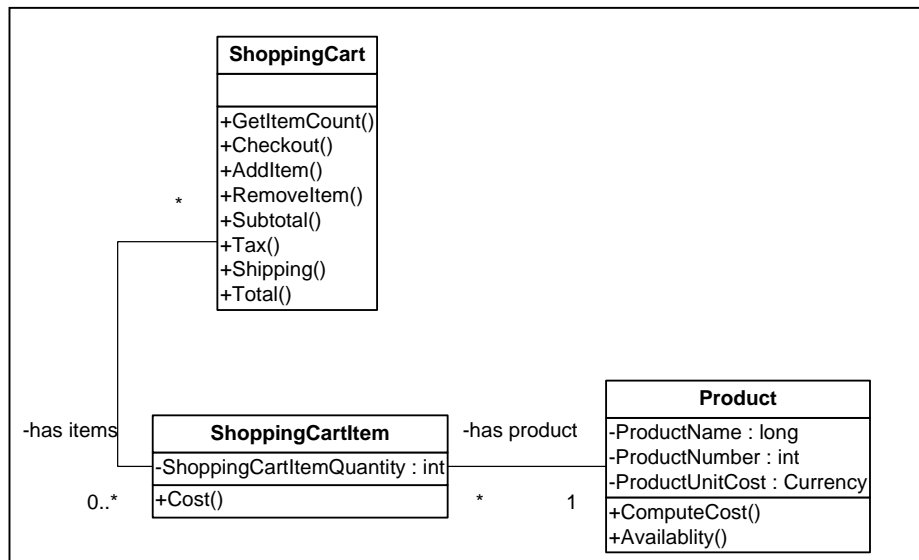
3.1.1 The determination of the classes

A class model is determined through a series of stepsⁱⁱ. This is done in brainstormsessions. Preferably, the whole team joins these sessions. If this is not possible, at least the functional decisionmaker and a couple of technical developers should be present.

- Identify all possible class candidates from the problem domain and use cases (if present);
- select the classes from the list of candidates;
- make a model dictionary;
- identify associations;
- identify attributes;
- identify operations;
- generalise using inheritance (Be careful! This is not supported by all programming languages);
- make constraints;
- iterate along these steps.

3.1.2 An example of a Domain Class Diagram

The result of the described session could lead to the following diagram. This simple example is taken from the design of an online sales system.



3.2 Requirement description

The description of requirements of the systems can be divided in three parts: the use cases, their stories and the system requirements.

3.2.1 Use cases

The business requirements that describe the outside functionality of the system are defined in UML use cases. Use cases are short descriptions of the behavior of the system, from the point of view of the user. A use case describes explicitly the interaction between user or actor and the system, where the system is considered a black box.

A use case has a name, preconditions, a description, exceptions, postconditions and a priority.

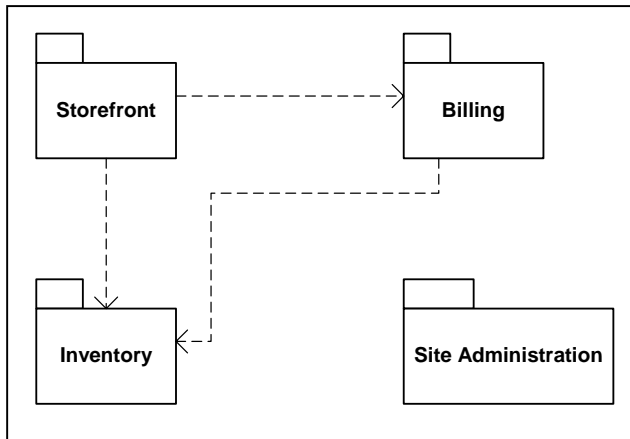
The functional decisionmaker defines the use cases. He determines their business value and therefore their priority.

The use case list can be altered to a testlist by reverting the use cases to questions. By asking these questions, the abilities of the system to satisfy the requirements can be checked easily.

3.2.2 Use case packages

The complete system functionality is described in functional parts or packages of use cases. One of these could be the client interface, or an administration module.

A package can be shown by means of a UML Use Case Package Diagram, as is done for an online sales system in the diagram below.



3.2.3 Stories

In a use case, the system is considered a black box. This means that internal logic is not described or even mentioned.

In WUP, a use case has one or more stories. Stories describe the steps to be taken within the system to realize a use case. Stories are not part of UML, but are part of WUP because it is a very direct way of describing what the system has to do.

The functional decisionmaker and the developers define the stories of each use case together, as opposed to the use cases themselves, which are defined by the functional decisionmaker. A story is a unit that can be implemented, and can be considered a direct task for the developers.

A story can be technical, but can also be meant to be dealt with by a designer or a writer. The functional decisionmaker should not be bothered with this distinction. The developers themselves sign up for realising a use case or individual stories.

3.2.4 Story estimation

The developers estimate the complexity of each story and therefore the complexity of each use case. Each developer signs in to realize a couple of stories.

These estimates are expressed in PPD. A PPD means a Perfect Project Day, or a day of development for a senior developer without external complications.

A story has, in general, a realisation period with a maximum of 2 or 3 PPD. If the number of PPD's is higher than that, the team should consider to split the story.

3.2.5 Speed of realisation

The speed of realisation for different developer levels can be estimated like this:

Senior	1	PPD/day
Medior	0.8	PPD/day
Junior	0.6	PPD/day
Trainee	0.4	PPD/day

Using these assumptions, the duration of the project can be estimated considering the resources of the team.

It's as simple as this: for each developing team member, multiply the number of days he or she is present each week by their specific daily speed. This leads to the number of PPD's each week per person. Sum up these numbers for the whole team; this leads to the team's week production. Divide the total project amount of PPD's (all use cases summed up) by the teams week performance. This results in the duration of the project in weeks.

If this duration doesn't fit the deadline, use cases with low priority should be discarded from the to-do list. This procedure should be repeated until the list of use cases satisfies the deadline of the project.

For more information about this, take a look at the document [duration estimation](#)ⁱⁱⁱ by the same author.

3.2.6 Priority

The functional decisionmaker has to determine the priority of every use case and every story.

The priority indicates which use cases are to be realized and which are not in the case of a limited number of PPD's. To deploy before the deadline is more important than details in the functionality!

Priority P is quantified with an A, B or C.

P = A This is core functionality. The system cannot exist without it.

P = B The system could work without this piece of functionality, but it's still relatively important.

P = C If there are PPD's left before the deadline exceeds, they could be spent on functionality with priority C, but it's not a disaster if the functionality isn't built.

Within a use case, the stories can accordingly be prioritized. Usually, the total priority of a use case reflects the average priority of its stories.

3.2.7 Notation of use case and stories

A use case with its stories could be found in a WUP Functional Design in the following notation.

Use case 1			
<i>Use case name</i>			
Actor			
Extends	<i>Is this use case an extension of another use case?</i>		
Preconditions			
Description			
Exceptions			
Postconditions			
Overall Priority P			
Total PPD			
Nr.	Story	P	PPD
1.			
2.			
3.			

The left-open fields must be filled in.

3.2.8 Use case diagrams

WUP does not use UML use case diagrams by default, since one of its first priorities is to supply a functional design that can be adjusted in the blink of an eye. Obviously, it takes more time to adjust pictures than plain text.

However, if a functional design has over, let's say, 50 use cases, it is advisable to strategically add use case diagrams to get an overall view of certain parts of the process. The developers will thank you.

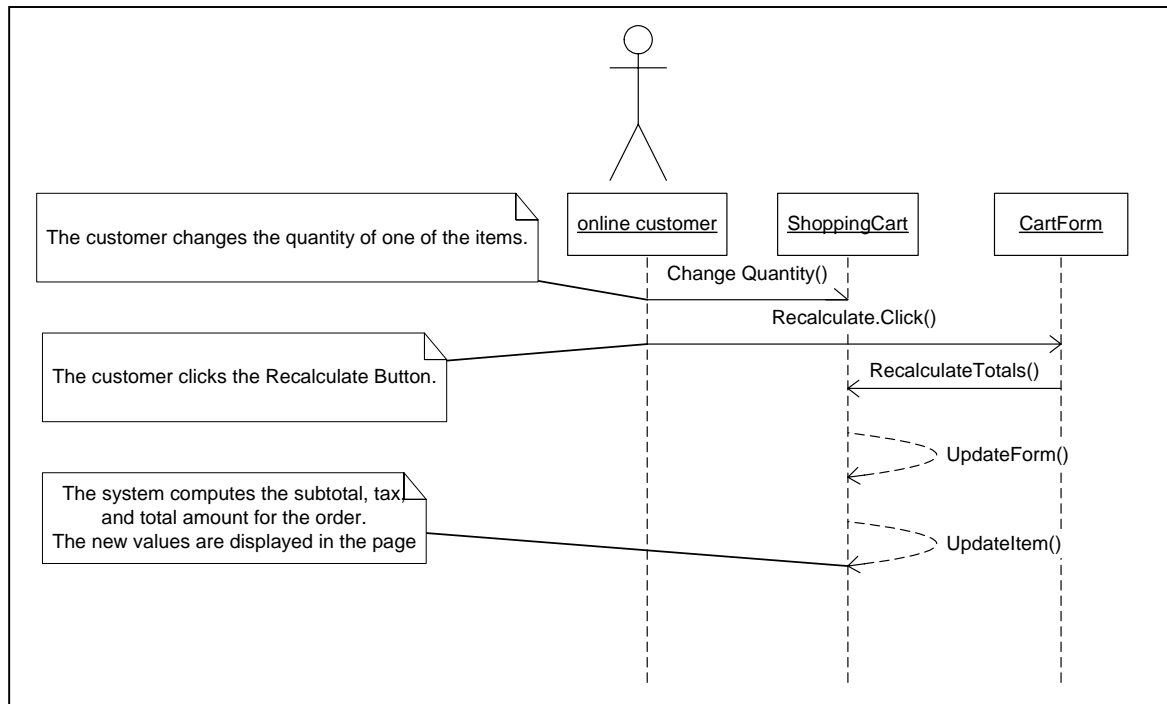
3.3 Sequence diagrams

A sequence diagram shows the interaction of the actor and the entities of the problem domain within a use case through time. This interaction is already implicitly defined in a use case and its stories, but the sequence diagram supplies a different view on each use case. The diagram relates the classes with the use case and the stories.

Apart from the problem domain classes, a sequence diagram uses web system classes if necessary. These system classes are usually defined in stories that tell from the interaction between business logic and presentation, for example. The business logic classes usually literally represent the problem domain classes, while presentation classes have limited system-related functions. These system classes represent objects like 'server page' , 'form' and 'client page'.

Within sequence diagrams the WAE (Web Application Extension) for UML^{iv} is used to define these web-related system classes. This extension uses metamodel classes or stereotypes that stand for the objects in a web architecture.

An example of a sequence diagram is shown of a use case that describes a part of an online sales system.



3.4 System requirements

System requirements are technical requirements that describe the rules to which the system must comply. These system requirements are not related to any action of an actor and are thus described in the Technical Design.

System requirements can be described in textual format, and are about things like choice of platform, performance and so on.

3.5 Design requirements

In WUP, there is no difference between designers and programmers: they're all developers in the WUP team. This, of course, implies that designers will have to read the technical design at least partially, as well as the functional design. Their feedback will only increase the overall quality of the application. However, apart from a functional and technical design, designers need their own set of tools and means of documentation.

3.5.1 Navigation map

A navigation map is a view of the web application showing the specific presentation level HTML pages in a hierarchical tree diagram. This map naturally evolves from the use case model.

Navigation maps can be made for different user levels or even actor specific if necessary. For example, the administrator module of an application has a totally different navigation map than the regular users of the system.

The navigation map can be found in the functional design.

3.5.2 Design brief

The design brief describes high-level user interface guidelines. The design brief is created by a designer and the customer directly. Typically, the creative brief defines:

- the mood of the site;
- whether the site will use frames;
- any color limitations the site will have;
- if applicable: a graphics standards guide.

Technical stuff like the kind of browsers that will be used and the connection speed of most of the users will be in the technical design.

3.5.3 Design template

From the Functional Design, the main web elements can be extracted that have to be designed. Web elements are the set of standard graphical components that return in almost every page. These elements and the design brief provide the information from which the designers make the design template.

The design template consists of one or more web pages that contain the main web elements that will be used. The design template is constructed with the sole purpose to have the customer make a decision on the creative design direction.

3.5.4 Design Comps

Design comps consist of mock-ups of what the site might look like, based on the design template. These mock-ups are typically pictures, with a browser frame around it. The comps supply the interaction designer with a structural style guideline, and the customer with happiness (trust me).

3.5.5 Interaction design and prototyping

From the use cases, the navigation map and the design comps, an interaction design is made. These are preferably black and white pictures that show the GUI functionality of each and every page. Colors would blur the customer's view on what's important here: the construction of an intuitive user interface.

If the scope of the project justifies it, HTML prototypes follow from the interaction design. These are plain HTML pages without any design that reflect the functionality of the interaction design, but which have a simulated

transaction handling (i.e. no real business logic, just a simulation). This gives even more insight in the GUI.

3.5.6 Design production planning

Fixed estimates can be made for the creation of a navigation map, the design brief, design template, comps, interaction design and prototypes.

Planning ahead for design production in a structured way is only possible after the completion of the initial phase in design. From that moment on, designers (or maybe HTML template builders, depending on your capacity) know exactly what to make. This means that for every page that has to be constructed, the number of PPD's is estimated by the designer that signs up for the page.

3.6 Risk

It is very important to address risk in a system before even starting to develop. There are a few types of risk in every project that can be classified.

Requirements risk - Disaster scenarios have to be outlined in an early stage by the functional decisionmaker. He knows best what could be the worst thing to happen to his business. Special use cases that describe these disasters have the highest priority.

Technological risk – Does the technology that is chosen or required fit the functional requirements? Can it deliver the functions that the users need?

Skills risk – Are the available team members able to build the required application? If not, then don't start, but demand more or other team members.

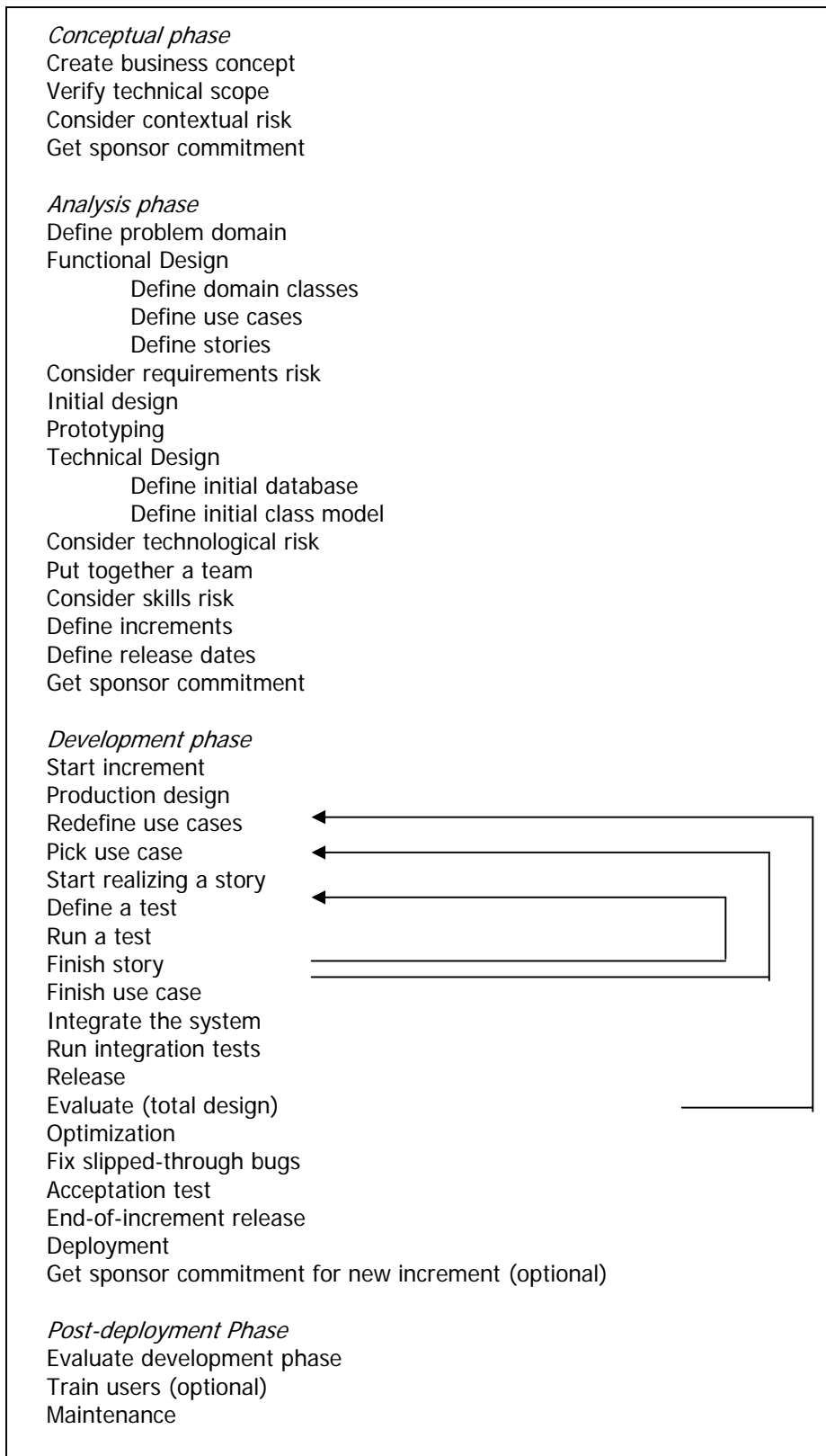
Contextual risk – Are there any political forces that could possibly get in the way of your project, legal consequences or people with hidden agendas? Think about this kind of risk, and only document it if you don't enlarge the risk by doing so!

4 The project

Now that everyone is familiar with the definitions used, the flow of the project can be described.

4.1 Project flow

The phases of a WUP project are shown below, including what happens in these phases.



When changing phases, the customer should be able to call off the continuation of the project. Each of the phases can be considered a time box and each of them can be planned and estimated in cost separately. During the phases, the customer should be closely involved and able to steer the project at all times.

4.2 Conceptual Phase

WUP doesn't define much of how to work in this phase, since its philosophy is to let everybody do in which he's best. However, there are a couple of things that should be mentioned.

The conceptual phase does not require a specific team. Depending on the company, business scope or customer, the people that create a concept are chosen.

Usually marketeers are involved. Their advantage is, that they can create concepts without bothering about technical stuff; but their advantage is also their disadvantage. A lot of web projects are spoiled because business marketeers and application development hardly mix.

To prevent from this, it is important to address the technical scope of a concept in an early stage. This means that at least a member of the technical department should be incorporated in the conceptual process, to verify whether it is technically possible to build an application that fits the concept. Don't sign or promise anything until this is verified!

In the conceptual phase, it is as important as in the later phases to remember that the customer for whom the concept is created knows his business best. So when thinking of a business concept, communication with the customer is key.

The other thing is: don't try to speak of buttons, navigation or any web stuff like that in a business concept. It's supposed to be a concept, not an application design.

Wait with the nifty details until the Analysis Phase starts, and let developers define them: they know best how to do that. If you don't stick to the big picture, for sure the customer will get falsely informed, because the details will change in the process anyway. Usually, the best business concepts can be described in just a couple of written pages.

Obviously, the end of the conceptual phase requires the commitment of a project sponsor.

4.3 Analysis Phase

The first thing to do in the analysis phase, is to determine the problem domain in detail. The functional decisionmaker and the customer sit together and write down a description of the problem domain. This description shouldn't be longer than one page.

An initial problem domain class diagram can be constructed with the customer, since he knows his own business best.

From the problem domain and the concept, a list of functional requirements is derived that describes what the application should do to fit the concept within the domain. These requirements are described using use cases and stories, which are prioritized.

This prioritization should lead to the decision whether to deliver the system in increments or not, depending on the wishes of the customer. Wait with the exact dates and detailed functionality of these increments after the functional design is constructed.

Now is the time to put together an initial team of developers, that will help constructing the functional, technical, interaction and creative design. These are the bare necessities for a developer to start of with the next phase: the development phase itself.

4.3.1 Functional design

From the requirements list and the initial class diagram, a functional design is derived, preferably by the functional decisionmaker (that is, if he is not the customer). This can be done using the template [FD document](#)^v by the same author.

This is an initial FD, from which an initial project scheme will be made. In this FD problem domain class, use case package, use case and sequence diagrams are present.

After completion of the initial FD, plan a meeting with the developers to discuss the contents. Let them estimate the complexity of each use case and story. This will supply some insight in project duration.

Verify the correctness of the FD with the customer directly.

4.3.2 Initial design

Since the user interface is really important, the look & feel of the application should be determined in an early stage of the project. Parallel to the construction of use cases, the initial user interface guidelines should be determined.

These guidelines have a lot to do with the emotion that the users of the system should experience. Therefore, it is best to send one of the designers to speak with the customer directly. It is one thing to have a functional decisionmaker in the team, but to translate emotion is hardly possible. This designer returns with enough information to complete the Design Brief.

By the way, the technical boundaries of the system associated with design (for example, which browsers do the users support) should not necessarily be determined by this designer, but should already be in the technical design.

The use of the Design Brief should result into one or a couple of pages that represent the feeling that the customer wants his application to have. These pages together are called the Design Template. In this template, the most important design elements are shown.

It is recommended that that just one type of design template is made, which should be adjusted according to the customers' wishes. If more than one design is offered, the customer mostly applies for a combination anyway. This is a lot of extra work for the designers and gives a nasty competitive edge to the creative process.

Finally, a couple of pages of the future system are simulated, that are used as a guideline by the interaction designer to construct his interaction design. These mock-ups of the future web GUI are called design comps.

4.3.3 Interaction design and prototyping

The use cases (not necessarily the stories) and sequence diagrams are handed over to a creative person in the team who is or acts as an interaction designer. He uses the use cases to construct an outline of the navigation of the future system, the navigation map. Most of the pages on this map are already defined functionally in the use cases or in the sequence diagrams, especially if the WAE for UML has been used.

The interaction designer then composes pictures of each and every page. He defines where the input fields are, in which order, where the buttons are, the logo's and everything else I can't think of right now. These pictures, preferably digitized but under schedule pressure sometimes even hand-drawn, are called the Interaction Design. The Interaction Design is a key feature, and not just in WUP. During this process, the interaction designer keeps an eye on the style guidelines as defined in the comps by the creative designers too.

The next step is to create functional prototypes for each of those pages. This process results in a series of semi-interactive HTML pages that supply guidance

for the developers that have to build the presentation part of the application. Take careful note of the implications of building these prototypes on the schedule: if the schedule is tight, use the 'paper' interaction design and forget the prototypes.

Both the navigation map and the interaction design are a powerful tool to show the customer what his application will look like. Let the functional decisionmaker comment on them and adjust them if necessary.

4.3.4 Technical design

Along with the initial design, the technical design can be started (for example using the [TD template document](#)^{vi}). This can be done by anyone from the technical developers. However, this person should have participated in the meetings that were about the functional design.

The technical design is relatively simple: the class model is directly defined from the problem domain class diagram and the rudimentary functionality of the system. The same goes for the data model.

It is no use to define in more detail. The technical design will be an initial guideline, but as the project rolls it will serve more and more as documentation instead of a guideline. In practice, solutions for problems can be figured out on the job by those who are best at this: the developers. So, during development the technical design will follow the developers, not the other way round.

One person of the technical developers should be given responsibility for keeping an eye on the TD staying up to date during the project. Every developer is responsible for the changes in TD: every time he changes something that affects the TD, he should change the TD.

4.3.5 Planning increments

After completion of FD, TD and initial design, a detailed project planning can be constructed.

The functional decisionmaker defines at the beginning of an increment which use cases should be build, based on business value, complexity and the speed of realisation of the developers.

An increment is planned using the following set of actions:

- Determine which use cases and stories you need to end up with a successful product;
- Let the designated seniors estimate the PPD's for each story and use case;
- Prioritise use cases and stories;
- Estimate the speed of realisation for each team member;
- Define an increment release date;
- Pick the use cases that fit in the development period according to the team's capacity and the deadline, starting with the ones with highest priority;
- Do some risk analysis based on the chosen functionality (what can go wrong?).

Do not forget that the first release period requires more time than the rest of them, because a couple of initial things have to be taken care of. Think about the initial classes, database and of course the technical development environments, software and so on.

Furthermore, internal release cycles can be planned for. Internal releases are the releases in which the system is deployed on the test environment on a regular basis. At the moment of deployment, the development stops. This is the start of a short integration and functional test period in which the whole team takes part, thus preventing ending up with a load of bugs at the end of the increment. It is recommended to do such a test release at least every two weeks, or even every week when you are on a tight schedule.

4.4 Development Phase

An increment starts with a couple of kick-off meetings with the whole team. The seniors know what the functionality is about, since they estimated the PPD's, but the rest of the team must be filled in about the details. The functional decisionmaker presents the use cases for the increment, the technical seniors educate the rest of the team about the procedures if they're not yet familiar with them.

It is advisable to have the team members picking the use cases that they would like to build themselves. In doing so, the team members will have to build the things that they like to build, and they are individually responsible for their choice. Obviously, there will be some use cases left that have to be assigned.

So, every developer signs up for a couple of use cases. This results in use cases being designated technical, design or content use cases, or any mix of these components.

Since every team member is present at these meetings, it is clear for everybody what the product will look like at the end of the increment.

At this point in the process, we're ready for some coding!

4.4.1 Stand-up meetings and the whiteboard

During development, the project manager and the functional decisionmaker organise an informal meeting, every morning at the work place. During these stand-up meetings the use cases and stories with highest priority are written on a whiteboard, preferably by the functional decisionmaker.

While the whiteboard is being filled for the day, the whole team is present and can react to changes in the requirement schedule immediately. The functional decisionmaker and the project manager can update their functional inventory; that is, since all developers are present, it will become clear what's built and what's not built from yesterday's requirements.

The whiteboard must be visible at all times for the whole team during development. This might sound rigid, though the whiteboard has a central function in the process.

Next to the specified daily use cases the name of the responsible developer is written. When he has finished the specific use case, the developer simply points that out on the whiteboard and moves on to the next use case.

The daily updated whiteboard makes sure that even the smallest changes in requirements are picked up directly by the whole team. If anything is not clear, the functional decisionmaker can explain the difficult parts directly.

When monitoring the team's progress, it is inevitable that use cases are switched from one developer to another at a certain point in the project. When names are changing for a use case, this will become clear in the stand-up meeting and pointed out on the whiteboard too.

It is key to keep communication direct at all times: so please let the whole team work together in one room, so that everybody can see the whiteboard!

4.4.2 Database- and stored procedure scripts

Although this is not directly an essential WUP feature, it is highly recommended to use database- and stored procedure scripts instead of fiddling around in database client software. During the development phase, these scripts have to be updated continuously.

If something goes wrong with the project database, you won't have to rely on the accuracy of your system administrator to restore your database. Secondly, scripts are usually a necessity during deployment.

During development, every change in database or stored procedures has to be applied using the scripts instead of client software. This is the only way to ensure that developers keep the scripts up to date.

Automatically generated scripts are usually a mess, so you can't rely on them, since the script has to be readable for any new developer joining the team. Manually changing the scripts demands a higher level of knowing what you're doing anyway. The comment lines are a necessity too, to keep track of every change, since every developer is responsible for the scripts but also has to work with the resulting database.

It is advisable to assign one of the technical developers to be the script controller. He is not responsible for the contents, since every individual developer is responsible for his own changes, but he is Big Brother in database world.

Once a week the scripts are used to recover the database in a separate environment. A script from this recovered database is generated and compared with the generated script from the development database (it's easily done using versioning software, usually). These scripts should be identical, and if they're not, there's a problem. If they are identical, tag a backup of the manual version for the current date and keep on developing.

4.4.3 Releases

During development, releases have to be done for every independent part of the system as often as possible. This could be just internal releases, for testing purposes, but it is advisable to have the customer looking over your shoulder too.

To facilitate these releases, the entire team (and this means entire as in including project manager, functional decisionmaker and even the designers) has to use a versioning software package. Within this kind of software versions of code and design can be labelled for a specific release number. If this is done

consequently, the revision history of the whole project can be reconstructed whenever necessary.

Every release has a test procedure. As many tests as possible have to be automated instead of manually performed. For details on possible test methods, read the [test manual](#)^{vii} and the [test template document](#)^{viii} by the same author.

4.4.4 WUP Programming

The next set of rules should be taken into account in a WUP project.

Collective code ownership

All code is accessible for every team member. 'Your own code' does not exist! This makes all code the responsibility of every team member, and works as a control mechanism.

Code conventions and version administration

To make the rule of collective code ownership work, every programmer must apply the rules of the company's code conventions, must document his code thoroughly and must use the versioning system.

Simple design

Code is simple when it reflects the physical reality (that is, when the code uses logical steps to get to a solution; and remember that logic is more important than technical efficiency!).

Important is that clear, complete names are used for variables and that the code is logically structured, all according to the code conventions. Also, the source cannot contain double code!

Refactoring

Software entropy demands software to gradually become a total mess as new functionality is added. To prevent from this old code should be rewritten and enhanced constantly, even if it already works. This is called refactoring. If you don't redesign the program constantly, functional additions will become more and more complex and thus cost more time and money!

Development infrastructure

The application should be developed, tested and deployed during the project within an infrastructure that is clear to every team member and the customer too. Everyone must know where to find which part of the application at any time in the project. This demands a standard infrastructure, like the one described in [infrastructure specifications](#)^{ix}.

Pair programming

If the size of the project is large enough, two programmers could work together behind one screen. This prevents code from becoming messy, keeps the programmers focused and last but not least, it's more fun! The RSI monster doesn't like pair programming, either.

In the best case, programmers should switch pairs at least once a day.

4.4.5 Testing

All kinds of testing methods have to be performed continuously during development and releases. Especially for testing the business logic layer, automated test procedures have to be written. Don't wait with this until you're ready for testing, but start with the test classes right away.

Functional testing procedures have to be determined by the functional decisionmaker, based on real-life business scenarios. In practice, these scenarios are the use cases in reverse mode: stated as a question.

GUI (Graphical User Interface) testing does not have to be performed automatically, but can be done manually, preferably using some sort of checklists and the interaction design. According to the rules of structural programming, there is no business logic in the presentation layer. This means in practice, that the GUI bugs are relatively easy to fix. It is important, however, to verify that your GUI reflects the interaction design.

4.5 Post-deployment phase

The post-deployment phase can include beta testing, performance tuning, maintenance and user training. All this depends heavily on the agreements that are made with the customer. Therefore, there are no standard procedures in WUP that describe what to do.

A couple of things have to be kept in mind, though. Be certain to define in detail which responsibilities the customer expects you to fulfill. Think about upgrades, maintenance, quality guarantees and so on. It is recommended to have direct access to the production server when you have to do stuff like this. Another useful rule is, that when the customer adjusts anything in the application himself (that is, or his developers), nothing can be guaranteed any longer.

5 Suggestions

The next set of suggestions makes using WUP a lot easier..

- Don't try to design the whole system in advance;
- Try to think about scalability well in advance though;
- During implementation, please keep on designing (creative, functional, technical);

- Don't try to keep the functional requirements fixed;
- Let the functional decisionmaker choose the order in which use cases are implemented;
- Don't produce huge documents in analysis and development phase, save that for the post-deployment phase for documentation and evaluation purposes;
- focus on communication;
- produce only documents that are actually used;
- take note carefully of the reports;
- let every team member contribute in the creative process;
- don't build for tomorrow: building for today is hard as it is and maybe tomorrow doesn't fit your expectations;
- don't build for yesterday but for the current standard, i.e. don't let yourself get restricted by old versions of browsers that won't exist anymore before you can say Netscape, unless the customer demands it.

6 Reporting project progress

The project manager has to keep track of the project's progress. Every one or two weeks he should send out a status report to all members of the team, the customer and maybe some sponsors too. The schedule has to be fine-tuned using these reports.

A status report would at least contain the following topics: the team itself, evaluation of the past period, a prognosis for the upcoming period, the use cases that have been built versus the ones that haven't and of course the PPD estimates versus the spent time (in PPD's and true days).

You could use the [template report](#)^x to do this.

At the end of an increment or after a mayor release a separate evaluation has to be sent out to everyone involved. This evaluation should dig deeper into the requirements that have been realised versus the ones that haven't.

7 Conclusions

The practical development method WUP could well support teams of web builders to enhance their speed and quality of web application development.

Customers, project managers and developers will all benefit from using WUP.

8 Acknowledgements

I would like to thank the following people for their contribution to the development of WUP:

Zoran Kovacevic, Dick Appel, Fried Broekhof, Ton Goossens and David Ireland.

C-factory still rules the flavour!

9 Documents that you will probably need

Functional Design:	/Templates/Template_FD_WUP.doc
Technical Design:	/Templates/Template_TD_WUP.doc
Duration:	/Templates/Template_duration_WUP.doc
Reporting:	/Templates/Template_report_WUP.doc
Test Checklist:	/Templates/Template_Test_Checklist.doc
Test manual:	/Manuals/Testing.doc
Infrastructure description:	/Manuals/Infrastructure.doc

10 Resources

WUP is, apart from sheer practice, based on the following resources.

10.1 Books and white papers

Building Web Applications with UML

Conallen, Jim

Addison-Wesley, 2000

Building web solutions with the Rational Unified Progress: Unifying the creative design and the software engineering process

Ward, Stan / Kroll, Per

Rational Software white paper

Extreme programming installed

Jeffries, Ronald E. / Anderson, Ann / Hendrickson, Chet

Addison-Wesley, 2001

Planning extreme programming

Beck, Kent / Fowler, Martin

Addison-Wesley, 2001

Praktisch UML

Kleppen, Anneke / Warmer, Jos

Addison-Wesley, 1999

UML Distilled

Fowler, Martin
Addison-Wesley, 1997

10.2 Sites

<http://www.dsdm.org>
<http://www.junit.org/>
<http://www.xprogramming.com/>
<http://www.extremeprogramming.org/>
<http://www.xpdeveloper.com/>
<http://www.rational.com/>
<http://www.iconix.com/>

11 References

ⁱ Building Web Applications with UML
Conallen, Jim
Page 221

ⁱⁱ Praktisch UML
Kleppen, Anneke / Warmer, Jos
Page 46

ⁱⁱⁱ Template_duration_WUP.doc

^{iv} Building Web Applications with UML
Conallen, Jim
Page 221

^v Template_FO_WUP.doc

^{vi} Template_TD_WUP.doc

^{vii} Testing.doc

^{viii} Template_technical_Test_Checklist.doc

^{ix} Infrastructure.doc

^x Template_duration_WUP.doc