Software project management

Table of Contents

Context

Software projects are, from a project management perspective, an amount of work that has to be completed with sufficient quality in a given time interval, just like any other project. Notable differences that set software projects apart from other projects are the relatively low demands for management of resources, other than the people doing the work, and the relatively high amount of changes to the expected result of the project during project execution.

Humanity knows one basic way to make any amount of stuff manageable: divide et impera. The secret to all management is to divide the stuff into less large chunks, manage each chunk in itself, and manage the interaction between the chunks. Repeat recursively. The part that requires experience and insight here is to divide the amount in such a way that the interaction between the chunks is minimal, and not too difficult to manage itself.

- definining the project (analysis and design of objectives and events, defining the products of the project, ...)
- facilitating the work and enabling the people actually doing the work (allocation of resources, organizing the work, issue solving, people management, ...)
- managing the people (acquiring human resources, directing activities, assigning tasks, delegating responsibilities, coaching, ...)
- controlling the quality of the work (analyzing the results based on the facts achieved, quality management, issues management, defect prevention, ...)
- planning the work (estimating resource needs, organizing the work, controlling project execution, issues management, controlling project execution, tracking and reporting progress, identifying, managing & controlling changes, changing the plan, ...)
- assessing and controlling risk (forecasting future trends in the project, assessing and controling divergences from the plan, ...)
- project closure and project debrief
- communicating to stakeholders

(see the list of project management activities at Wikipedia)

Project risks are internal and external

list of work chunks

list changes

project management triangle cost / scope / schedule

http://en.wikipedia.org/wiki/Project_management

Manage risk with a plan

The main goal of this document is to address how to enable for the goal of a project to be reached, within the limits of available resources (time, materials, people, money, location, ...), in the context of the project, with diminishing uncertainty and increasing certainty of success. http://en.wikipedia.org/wiki/Cone_of_Uncertainty

The best way known to man to do this, is to imagine subgoals, recursively, in a (partial) order of dependencies, with the resources they each need, necessary to reach the final goal. Such an image we call a *plan*, and can be visualised in a directed acyclic graph, where when this is work, it is a work breakdown structure but then way of breakind down is work RISK

If this plan would be perfect, and it would be clear to everybody concerned that it is perfect, there would be no uncertainty about reaching the goal of the project by executing this plan, and thus no risk. In practice however, this plan is never perfect, and cannot be proven perfect, and from this uncertainty arises, and hence any project has a risk of exceeding the given limits or failing altogether. The plan is imperfect, and its degree of perfection is unknown, because:

- the list of subgoals is not guaranteed to be complete;

- the imagined resources needed for each subgoal and the final goal are only estimates; and it is even difficult or impossible to formulate a degree of accuracy for the estimates;
- execution of subgoals may, and probably sometimes will, not be perfect;
- choosen resources might not deliver as envisaged or may not have the imagined quality;
- the availability of resources for the project may, and probably will, change during execution of the image;
- the context of the project may, and probably will, change during execution of the image;
- and finally, because even the final goal of the project may, and often will, change during execution of the image, and even turns out not to be fully concretely defined.

Planning is one of the major tools used in project management to do this. It deals with subdividing goals in subgoals recursively, estimating the resources needed for each subgoal, allocating resources for each subgoal, and timing the execution of each subgoal with respect for interdependencies.

Success can't be measured

Project success is not a black or white concept. Although a project can fail completely (the goal is never reached and the project is stopped), it is not often that we can declare a complete success (the goal is reached, within the set resource limitations), simply because the goal is most often not fully concretely defined and has changed during the project. It is rare that the goal is a concrete checklist that can be used as a reference for absolute success.

And even then, between these two extremes, there are degrees of success. A project that satisfies the client, is delivered on time, within budget, but with a slightly different set of functionality, is surely a successfull project. A project that satisfies the client, is delivered on time, with a slight budget overrun, but more functionality than envisaged, cannot be called a failure. Even a project of which the budget was cut severly during execution, but that still delivers a minimal functionality on time is not a failure.

And whether or not the initial plan was executed to the letter or not at all is irrelevant. The plan is an enabling tool, and not part of the goal, or the measure of success.

Mitigating risk by lowering uncertainty beforehand

Project management is aimed at mitigating the risk of failure, heighten the chance for success, lessen the uncertainty about success, and heighten the degree of success.

One way to try to do this is to limit the uncertainties and risks at the start by being more formal. History shows however that this does really results in better projects. A reasonable amount of formality does help though, but the main gist is: there is risk in your project due to uncertainty, and you can't wish it away.

The formal way

We can *define* the goal of the project as the list of subgoals we created, and freeze it. We thus disregard any change in context, and anything we learn during the project. In most projects, this does not lead to a satisfied client however. The client got what he ordered, but not what he wants.

The same applies to changes in the context of the project. We can agree beforehand to not take into account changes in the context of the project during project execution. The client got what he ordered again, but not what he needs.

We can also try to make our estimates about resource needs more accurate. Historically, several theories and approaches have tried this, function point analysis being one of the more successful attempts. In practice, not one of these approaches has lead to significantly less uncertainty in projects though. We can try to at least know the degree of accuracy of the estimates, using statistics. But that requires historical data, and although we gather historical data in the project as it progresses, we do not have that data before the project execution starts. It is also difficult to use data from earlier projects, since most often the parameters differ enough between projects to make the data non-representative. This can possibly be solved with a very large set of data, but that is not someting we can do ourselves, and it is not unlikely that the differences between projects prove so great that the accuracy we find will be indeed very low.

We can try to get a more predictable quality level in the execution of subgoals. Note that this is not necessarily the same as a higher level of quality. It is foremost a more *predictable* level that lowers the degree of uncertainty (and thus, risk) associated with the project, not a higher level of quality. On the other hand, a low level of quality of execution will make the final result more complex, and from that complexity more uncertainty arises. In this approach, software developers are seen almost as automata that follow a standard approach, and there is no room for extra-ordinary performance or quality. Practice learns that this is a much more expensive approach though, since in general this does lower the level of quality to a lowest common denominator, and mitigates the risk associated with lower quality with more testing and more fix - test cycles.

We can try to lower the uncertainty about and risk associated with the resources we envisage to use in the project, by extensively studying and testing them, and proving their fittness, before we start the project. In practice however, often any full prove of fittness is the actual use of the resource in the exact or very similar context it would be used in the project, and the cost of the study and test of fittness is the same as or higher then the use of the resource in the project. By doing this, we will effectively lower the risk associated with the project, but the risk, uncertainty and associated cost has not disappeared. We have merely moved it out of the project, where it is less visible. Certainly the time resource comes under stress with this approach, since now a lot of work has to be done prior to the start of project execution. Furthermore, a lot of the work needed in the project for which the resource is needed still has to be done.

We can try to lower the uncertainty and risk associated with the project by lessening the uncertainty and risk about the availibility of resources during the execution of the project at the time they are needed. We can procure necessary resources beforehand, or make legally binding reservations, are SLA's, or get JIT guarantees. All this however comes with a cost. And whatever we do, there still are nature, death, accidents, and other random acts of god.

Of course all the possible endeavours listed above do indeed mitigate uncertainty and thus risk. However, this always comes with a cost, in client satisfaction, money and time. Many more traditional approaches to project management focus on these mitigating factors, and if you furthermore have a perfect project plan at the start of the project, we are set up for a perfect project. The history of project management over the last 100 years has shown however that in practice this does not happen, and that his is not even a good approximation of reality. Furthermore, the extra cost in client satisfaction, money and time, make for projects that are often prohibitively expensive (for the client, the contractor, or both) from the start, so that they are never approved. When they do get executed, they finish often objectively close to a complete success, but feel more like a complete failure.

Reality and necessary formality

It is important to define the goal of the project as completely as possible, and have a list of subgoals as exhaustive as possible. But that list only defines the *initial scope*. The project setup should take into account changes in the goal and the list of subgoals. It is then necessary to manage the changes and the impact of those changes carefully and deliberately. Any reasonable effort to ensure the availibility of resources when they are needed should be made. But you can have a fully binding legal contract, be completely in your right, be entitled to a giant fine, and still have a project that cannot be completed. This is of course

different for business critical projects, but in practice most often the reasonable effort stops at making an arrangement with people and asking for a commitment, and reminding them of it as the time to deliver draws nearer. When you have done that, we must assume that non-delivery is an accident.

Changes in the context of the project cannot be controlled from within the project. Just like problems with the availibility of resources, they should be considered acts of God, which you cannot plan for. You can only take into account that there is always a risk of it happening, and that this might even lead to project failure.

Estimates, in practice, are made by senior people, based on their extensive experience (at the cost of prior mistakes). In our experience, in a project that is large enough, errors on estimates cancel each other out more or less, with a resulting uncertainty of about 20%. Making estimates is a craft. What we can do, during execution of a project that is large enough, is use the initial estimates and actual usage during the earlier phases of the project as statistical data to determine the accuracy of the initial estimates for the entire project. A factor can be calculated (factor = initial estimate for subgoals completed / actual usage for subgoals completed) that can than be applied at the initial estimate for the entire project or the remaining subgoals (current estimate for subgoals remaining = initial estimate for subgoals remaining / factor). With a homogenuous spread of different kinds of subgoals, this factor becomes more and more accurate as the project progresses, which lessens uncertainty and risk for the remainder of the project. When the resource is the time spent by people to complete subgoals, this is called the *project speed*.

Trying to make the quality of execution more predictable is ok, as long as the level predicted is high :-). Again, this is, in our opinion, part of the craft of software development. We are convinced that higher quality makes, in the end, for a cheaper total cost of ownership. The level of quality that we envisage is reached by people management, by choosing the right people, training, experience, and most importantly, coaching. A standard approach to things helps a lot, but the people executing the project must not only apply the standard, but understand it. The standard should only cover the most common 80% of the work. The people doing the work must be able to see when the standard does not apply, and be able to handle the remaining 20% themselves. In general, we believe quality of subgoal execution should be high enough to make testing a formality. The discovery of a problem during testing should be a rare event, a reason for surprise, a social gathering, and some banter. We observe by the way that a low level of quality is often rather the result of bad communication of the subgoals and of behavior. Fix - test cycles become a costly way for the people executing the subgoal to find out what it is exactly that is expected of them. Better communication about the subgoals and coaching of behavior should not be replaced by more fix - test cycles.

Of course we should choose the resources for the project wisely and with confidence. However, most resources we choose for a project are resources of which the functionality and quality is known, and resources that are new and unknown are rare. For new and unknown resources, getting to know them should be a subgoal *within* the project however, and there should be sufficient confidence about the fittness of the resource. This choice does entail a risk, concerning fittness and quality, but it does not make sense to plan for this eventuality. If the choosen resource is not a fitt, or is not of the required quality, what will you do? Try an alternative is the only option. But you have choosen this particular resource above its alternatives at a given moment, as the most appropriate to your knowledge at that time. You were wrong, it turns out, but you only find that out while employing the resource in your project. It wasn't obvious that the resource would not satisfy the needs of the project, and it wasn't obvious for the alternative either, or you would have choosen the alternative to start with, with less uncertainty and risk, no? Surely it is important to put the resource to the test as early as possible in the project, with a minimal cost (this is called a *spike*). But most often doing this within the project twice is less costly then setting up a make-believe context outside of the project to choose the best between the alternatives with certainty (if that is at all possible). Note that, for the project, we are not interested in the best alternative, but in an appropriate alternative that enables us to reach the project goals. When experienced people choose an alternative, chances that the choosen resource is not up to its task are small. The extra cost of evaluation of alternatives and prove of fittness outside the project is most often much larger than the accidental extra cost in money, time, and

procurement of another alternative during the cost. That a choosen resource does not perform as expected should be treated, again, as an act of God.

If we note al these possible accidents and acts of God, and our dependence on the craftmanship of the people involved, it should be clear that uncertainty and risk is part of any project in practice, and whatever we do, we cannot make ik go away. It follows that we need ways of dealing with these risks and uncertainties.

Now suppose we would have a good way of dealing with risks and uncertainties, would it still be necessary then to go to great lengths to diminish the uncertainty and associated risk before the project starts? As we have seen, this comes with a considerable cost. If the total cost of the project that starts with greater uncertainty, but managed with this good way of dealing with it, is lower than the total cost of the project that starts with less uncertainty, which of the two is the better approach?

Planning is a continuous activity

Since we have established that a plan is in practice never perfect, and all parameters of the project can, and often will, change during project execution, it follows that even the best possible plan at the start of a project, even the ideal plan at the start of a project, doesn't keep that quality as time passes.

Even if the context, resource availibility, and the final goal miraculously do not change, and even if the list of subgoals was complete, and the estimates prove to be head on, and the execution is flawless, and resources are flawless, still the plan is no longer the best possible or ideal. Because, as the project progresses, more and more of the subgoals are actually reached, and thus the uncertainty about them and the risk associated with them has become zero. You did not know this beforehand. This means that the total uncertainty about the plan and the risk associated with the project is now less, and this probably means that there are better plans for the rest of the project.

Because you did not know about the quality of the plan at the start of the project, the best possible or ideal plan at that time had in it ways to deal with the uncertainty you had then about those subgoals and the risk those subgoals contributed to the overall risk then. Those buffers are now no longer needed. Furthermore, because of the success of the subgoals that are completed already, and because further subgoals are probably related and similar, your uncertainty about those subgoals and the risk associated with them probably lessened (and certainly your estimate of the uncertainty and risk has), and there is now a better plan for the rest of the project (e.g., with less resources and lower contingency reservations).

In short, it is even so in theory that *a project plan needs to be adapted continuously during project execution*.

Of course, in practice the context and resource availibility do change, execution and resources are not flawless, and estimates differ significantly from real consumption. And as we progress through the project, we learn more about the project, and confronted with reality and partial results we discover missing subgoals, or better subgoal divisions, and flaws or incompleteness in the goal, or alternative solutions that contractor and client both believe are better. The intermediate results itself change the context and the goal. So, a plan is just that. It is a path we set out for a longer period of time, that guides our actions now. And after this moment, we need to revise the plan with the new knowledge we gained, to guide our actions in the next now. Planning is a continuous activity throughout the project, and not something you do once before the project starts.

Good enough for now

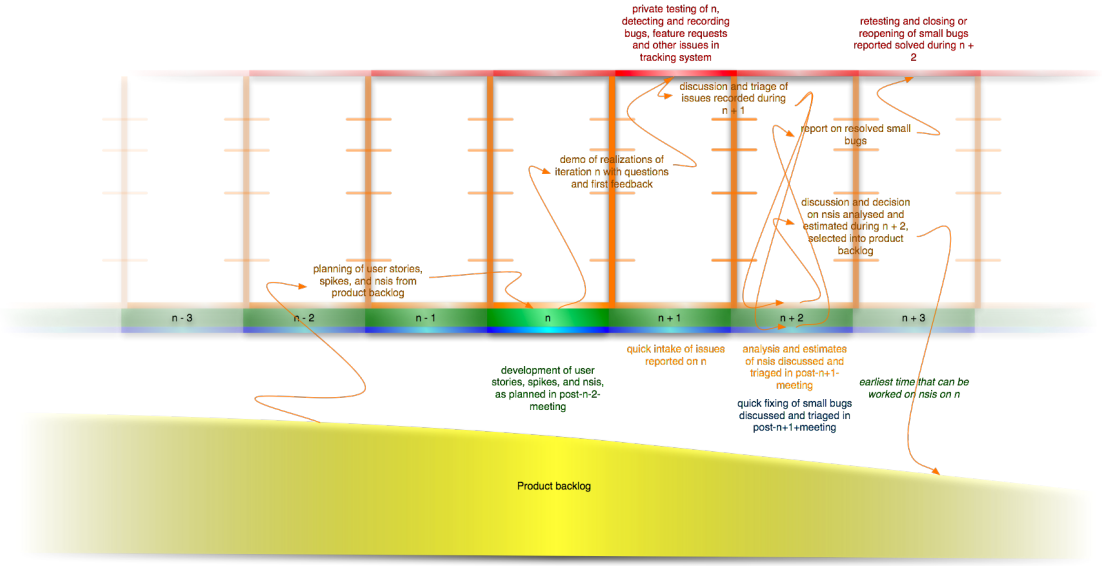
Now, if we know that even an ideal plan at the start of the project needs to be continuously adapted during project execution, why would we invest in trying to make the initial project plan (or any adaptation in between) as ideal as possible? This effort makes no sense.

Together, it turns out that any project needs

Iteration n

First we will discuss the flow of the project when we are up-to-speed, around the work to be done in an iteration n. At the start and the end of the project, things are a bit different. These border situations will be discussed later.

Figure 1. The software project management process in steady mode



Work flow

Planning iteration n

At the meeting at the end of iteration n-2, the *user stories*, *spikes* and *other planning blocks* from the *product backlog* are planned between client and contractor for iteration n and n+1.

These terms will be defined in detail later. For know, it is enough to understand that *planning blocks* are the units of planning, with a size between 2 and 6 mandays. The *product backlog* is the list of all planning blocks that together make up the work to reach the final goal of the project, as agreed between client and contractor at this time. *Issue* is used as the most general term, encompassing both planning blocks, bugs, or units of work.

The planning for iteration n is done at the meeting at the end of iteration n-2 because it is too late to plan for iteration n-1: that iteration has already started, and we expect it to take some time for the project manager to prepare for the execution of decisions of this meeting. After the nominal work of iteration n-2 ended, development already started on the work scheduled for iteration n-1, while the demo and the meeting was prepared. At the meeting, a proposal, which is in essence the original plan, is proposed by the contractor. All the larger issues have an estimate, and the available capacity for iteration n is known. We decide what to do in n and n+1, and maybe already change the plans for n+2 and further.

Executing the work planned for iteration n

The work planned for iteration n is then executed during iteration n. The construction team plans this work internally, in a most efficient manner for the work load planned for the iteration as a whole. This means, that it is not necessary (although it is safer) to do this work in the order of priority, or in exactly the planning blocks as defined in the planning. The net effect must be that the total work planned for the

iteration is done and ready for demo at the end of the iteration work cycle, and the path to that goal is an internal issue to the team. On the other hand, the measure of success is the completion of planning blocks as a whole. A planning block that is completed even 99%, and cannot be demoed, does not count in the evaluation of the iteration. It is thus highly advisable to structure work internally according to the defined planning blocks, to consolidate work done as soon as possible.

Demo of the realizations of iteration n

At the meeting at the end of iteration n, the result of the work of iteration n is presented and demoed. The client asks questions and gets answers, and gives his initial feedback. First of all, the contractor reports whether the work was completed as planned, and only demoes user stories that are completely finished.

Client testing of the realizations of iteration n

During iteration n+1, the client tests the software that has been completed, i.e., everything that has been constructed and demoed in the iterations 1 .. n. All kinds of issues, bugs, remarks, new ideas, new feature requests, and anything else is logged by the client in the issue tracking system. The classification and priority given by the client to these new issues is indicative.

Quick intake of issues recorded on realizations of iteration n

During iteration n+1, the contractor does a quick intake of the issues reported during that iteration on the result of the work of iterations 1 .. n. Some things can be dealt with immediately (such as closing issues that are the result of misunderstandings, answering to quesions, etc.). Remarks are added, questions on recorded issues can be asked via the issue tracking system. This quick intake is necessary in preparation for the meeting at the end of iteration n+1. During any iteration, the contractor can add issues too, for any reason, but also in response to issues created by the client (e.g., a subtask to study an issue added by the client).

Discussion and triage of issues recorded on realizations of iteration n

At the meeting at the end of iteration n+1, the issues recorded during iteration n+1 are discussed and triaged (i.e., the type of the issue and its priority is set) in mutual agreement between the client and the contractor. For the purpose of work scheduling, the important classification is either whether the issue describes a *bug*, or something else. Bugs are put on the *bugs list*. Anything else is placed on the *research list*.

Bugs recorded on realizations of iteration n

Quick fixing of bugs recorded on realizations of iteration n

Capacity is reserved during iterations to deal with bugs. Issues in the bugs list are handled within that capacity, and normally this means that the issues that are put on the bugs list during the meeting at the end of iteration n+1 are handled within that capacity during iteration n+2. These are normally small bugs recorded on subjects constructed during iteration n (which was planned in the meeting at the end of iteration n-2). These issues are kept apart from the others, since these are issues for which the actual work load is relatively low in comparison to the overhead of administering the issue (discussion, research, analysis, estimate, reporting on the estimate, discussion, and decision) and it is certain that the work needs to be done. There is no discussion or decision necessary.

Reporting on the fixing of small bugs recorded on realizations of iteration n

At the meeting at the end of iteration n+2, the contractor reports in a concise way which bugs have been resolved.

Retesting and closing of small bugs recorded on realizations of iteration n

These bugs are retested by the client during iteration n+3, and closed, or posibly reopened. In the latter case, they are reported again during the meeting at the end of iteration n+3, and discussed and triaged with the other issues.

Non-bug issues

Analyzing and making estimates on non-bug issues recorded on realizations of iteration n

Capacity is reserved during iterations to prepare discussion and decision during the next meeting, at the end of iteration n+2, on issues that are not bugs. These should not be tackled without further decision since either they represent a work load that will have a significant impact on planning (and they would certainly put the goals of iteration n+2 into jepoardy), or they are not part of the work client and contractor agreed upon. The latter is extremely important, since adding anything to the expected results of the project, even if it is initially seen as having a very small workload, is known to start an avalanche of feature creep. On the one hand, the contractor will not agree to do this work within the agreed scope of a fixed price project. On the other hand, feature creep is known to be the absolutely major reason why projects are delivered later than planned, go over budget, or simply fail.

This is not to say that changes to the original scope of the project are not allowed. But, it is absolutely necessary that they are made deliberately, with insight in the impact that they will have on the overal timing of the project and the price, in mutual agreement between the client and the contractor. This requires a clear analysis of the impact of the issue, and an estimate of the work load, and in our experience. To be able to make the estimate, for larger issues, it may be necessary to list subtasks in a finer granularity. Furthermore, it is not a bad idea to give the client some time to ponder how necessary the issue is for completion of the current version of the project. Things that seem absolutely indispensable now, often turn out much less important after the dust has settled, in the context of other equally indispensable issues, and with a price card in time and money attached.

When the client analyses the non-bug issues within the capacity reserved for this during iteration n+2, he might find that fixing the issue takes less time than the overhead of administering the issue in this way. If the issue certain to be work that needs to be done anyway, i.e., it is a bug, the client can decide to move the issue to the bugs list and fix it immediately on its own accord. If it is not absolutely certain that the work needs to be done, the work should not be done, and a decision in mutual agreement between the client and the contractor should be waited on. This goes even for very small issues, like changing a label. In our experience, it are often precisely these very small issues where clients change their mind a lot, and where merry-go-rounds start, which have a sizeable impact on the overal cost. Only work that is sure and a conscious decision of the client should be executed.

For issues recorded on realizations of iteration n that are exceptionally clear as recorded, and extremely simple, it is possible that a complete triage, estimate and decision can be done during the quick intake of issues during iteration n+1 and the initial discussion and triage of issues recorded on realizations of iteration n at the meeting at the end of iteration n+1. Although this probably will be exceptional, if client and contractor agree, issues are allowed to follow this expedited flow.

Discussion and decision on non-bugs recorded on realizations of iteration n

During the meeting at the end of iteration n+2 the client reports on the impact and cost of each of the non-bugs. The contractor decides, with this information in mind, whether this issue will not be withheld, is moved to a later major version, or is kept within this version. The non-bugs that are kept within this version now become part of the product backlog of this version, next to the other issues that are already there. For the new issues on the product backlog, the client decides on the priority (must have, should have, could have).

As such, the new issues are part of the product backlog from which issues can be planned later in the meeting for iteration n+4.

For any issue on the product backlog, the client can change the priority at any time before the the issue is planned in an iteration n. Once it is planned in an iteration n (and we are past the meeting at the end of iteration n-2), the priority can no longer be changed.

Planning and reality

At the meeting at the end of iteration n, the result of the work of iteration n is presented and demoed. Most of the time, the actual results will not be exactly what was planned for the iteration.

Planning blocks

When progress was slower than expected, the work that was not done has been moved to n+1 automatically, and most likely the developers are already working on it. This means that the work planned at the end of iteration n-1 for iteration n+1 is too much, since the spill-over from n is added to that workload. It is thus to be expected that iteration n+1 will not meet its target either, and that there will be a spill-over of work from iteration n+1 into iteration n+2. Since n+2 (and n+3) is being planned at the end of this meeting, that planning will take the spill-over into account.

When progress was faster than expected, the development team has already started on, and maybe already completed, planning blocks defined for iteration n+1 during iteration n. This means that the work planned at the end of iteration n-1 for iteration n+1 is too little. It is thus to be expected that during iteration n+1 work that will be defined at the end of this meeting for iteration n+2 will be done too. Again, the planning of iteration n+2 will take the difference into account.

It does not make sense to change the planning for iteration n+1, since work on n+1 has already started, and the internal work planning of the construction team of the iteration n+1 would be severely impacted and possibly need to be redone when the work package changes. Also, it is not advisable to extend the iteration. Not only does this make long-time planning difficult, and does it mess up peoples schedules, but most importantly because it changes the yard stick which with progress is measured. This makes it impossible to compare the work done in the iterations, which makes progress reports obscure, and thus heigthens uncertainty, which, in turn, damages the confidence in the project of as well the client as the contractor.

Bugs

For the same reasons, the capacity that is reserved for fixing bugs is fixed. The capacity needed for fixing bugs should be low if the quality of the construction work itself is sufficiently high. Since the work for any iteration n+2 is planned at the end of iteration n, and the work that needs to be done during n+2 fixing bugs is decided only at the meeting at the end of iteration n+1, it is much too late to change that capacity.

If there is more work than the capacity allows, it should be done within the capacity reserved for fixing bugs during iteration n+3. Since the work load for iteration n+3 is decided during the meeting at the end of iteration n+1, the planning for that iteration could reserve more capacity for fixing bugs.

If there is less work in fixing bugs than the capacity allows in iteration n+2, it cannot be used for fixing more bugs, since new bugs will only be added to the bugs list during the meeting at the end of this iteration. This extra capacity should be used first to complete the user planning blocks defined for iteration n+2, and then to work on planning blocks defined for n+3.

Analyzing and estimating new non-bugs

Again, the same applies to the capacity reserved for analyzing and making estimates for the non-bugs.

Estimates and reality

Estimates of the amount of work needed for a planning block or any other issue are merely that: estimates. It is to be *expected* that the actual time spent on an issue differs from the estimate. In general, with enough subgoals, and estimates made by people with enough experience, the differences between estimates and actual time spent should average out.

In the context of all of the above, it is advisable to not plan too much for an iteration. It is better to leave a little margin in an iteration. If the planned work is indeed completed before the end of the iteration, work can already start at the work planned for the next iteration. If is far better for the trust between the client and the contractor to be able to report on completion of all the work planned for an iteration, and to state that the work on the next iteration already started, than the other way around.

The actual time used for each subgoal is measured during the project progress. Thus as the project progresses, we can see how much time the completed subgoals actually took in sum, and how much time was estimated for those subgoals initially in sum. A factor can be calculated (factor = initial estimate for subgoals completed / actual usage for subgoals completed) that can than be applied at the initial estimate for the entire project or the remaining subgoals (current estimate for subgoals remaining = initial estimate for subgoals remaining / factor). With a homogenuous spread of different kinds of subgoals, this factor becomes more and more accurate as the project progresses, which lessens uncertainty and risk for the remainder of the project. This is called the *project speed*.

In this context, we need to mention that the estimates on work blocks are made in *ideal mandays*. This is not a true calendar day, and we cannot simply map the estimates in ideal mandays on a calendar and see where project will be completed. First of all, during a day, people do other things than the core work they are hired for. They need some discussion and synchronisation, need to eat and drink, and satisfy other bodily functions. Furthermore, it happens that people are called away from the project for some reason now and again. Every time a person switches work, we need to take into account some overhead time needed to retrieve the train of thought and get up to speed. Further, when mapping ideal days unto a calendar, we need to take into account holidays and possible sick leave. When transforming an estimate in ideal mandays into a promise for a delivery date, it is better to leave some room for contingencies. Nobody gets hurt if the contractor delivers early. Delivering late however has a severe impact on other teams most of the time. In general, when estimates are expressed in hours, we map hours to days using 80% of a work day. Days are mapped to months using 16 work days per month for multi-month projects. For shorter projects, it is better to use the actual calendar of people involved in the project and actually map the work load.

Agenda for the post-iteration meetings

Below the agenda for the post-iteration meeting after iteration n is consolidated:

Table 1. Agenda for post-iteration-n meeting

Subject	Duration
Welcome	5'
Discussion and triage of issues recorded during iteration n (on realizations of iterations 1 .. n-1)	30'
Reporting on the fixing of bugs (recorded during iteration n-1 on realizations of iteration 1 .. n-2)	5'
Demo of the realizations of iteration n	45'
Break	15'
Discussion and decision on planning blocks analyzed and estimated during iteration n (recorded during iteration n-1 on realizations of iteration 1 .. n-2)	50'
Planning of iteration n+2 and n+3	30'
Total	**(3h) 180'**

After the meeting at the end of iteration n, there is an internal meeting of the contractor team, where feedback is passed on.

Other meetings

Note that, apart from the post-iteration meetings, there might be additional meetings and contacts between client and contractor. The post-iteration meetings are reserved for formal reporting and planning, and other points should not be addressed during this meeting.

It is however very likely that there are other points to be discussed in more detail. It is to be expected, e.g., that there is a need during project execution to flesh out analysis and design, even requirements and screen mock-ups. Either because during execution it becomes clear that there are inconsistencies or there is ambiguity in the original agreements, or because it was planned from the get go to only deal with these details during project execution. Such work, of which meetings are a part, is then defined as any other work, in other planning blocks, put on the product back log, and planned like any other planning block.

It is also important to refrain during these other meetings from the topics reserved for the post-iteration meetings. In such meetings, there should be no demo's, there should be no reporting, and certainly there should be made no decisions regarding project scope or planning in these other meetings.

Verba volent, scripta manent

In the post-iteration meetings agreements and arrangements about the project future are decided on and commited to by all parties concerned. It is of the utmost importants that these decisions and commitments are clear to everybody, and are remembered exactly, so that they can be checked when they are due.

All agreements and arrangements should be clear and be defined in terms of what is to be delivered by when and by whom. To gain clarity and to keep the exact agreement on record for future reference, it is important to formulate the agreements, arrangements and decisions explicitly in a meeting report. This applies to the post-iteration meetings, but to other meetings too.

To expedite the meetings, it is a good idea for the project manager to prepare the meeting report in advance based on the agenda, and flesh it out during the meeting in mutual consensus about phrasing. It helps to use a projector to do this. This applies to the post-iteration meetings, but to other meetings too.

When an issue management system is used, the actual post-iteration meeting report can be pretty concise. Since the subjects of each agenda point are defined clearly in function of an issue management system

(see below), and decisions are recorded immediately, live, in the issue management system, there is not much to be added. If the report is kept in an electronic format, like a CMS or Wiki, the report can link to the lists in question in the issue tracking system.

For the discussion and triage of issues recorded during iteration n, a link should be provided in the report to the list in the issue management system of newly recorded issues discussed at the meeting. The decisions are recorded immediately in the issue tracking system, so do not need to be repeated in the report. Points of discussion, remarks, questions and their answers, and other comments made during the meeting need to be recorded. Most of these should be recorded as comments on the actual issues. More general comments can be added to the meeting report.

Reporting on the fixing of bugs (recorded during iteration n-1 on realizations of iteration is a one-way communication. The report should contain a link to the list of bugs reported fixed in the meeting in the issue management system.

For the demo of the realizations of iteration n, a link to the list in the issue planning system of planning blocks that are demoed in the meeting should be added to the report. Any remarks, comments, questions and answeres, should be noted in the report, or possible. It does not often make sense to still add comments to the issues itself, since these are closed, and will probably not be looked at again. It is however possible that at this time, live during the meeting, new issues are created to log remarks or comments. These are then among the first newly recorded issues of the next iteration. The creation of these new issues should be recorded in the report, and they should be linked to.

For the discussion and decision on planning blocks analyzed and estimated during iteration n, again a link to the list in the issue management system of the planning blocks discussed in the meeting should be available in the meeting report. The decisions are recorded immediately in the issue tracking system, so do not need to be repeated in the report. Points of discussion, remarks, questions and their answers, and other comments made during the meeting need to be recorded. Most of these should be recorded as comments on the actual issues. More general comments can be added to the meeting report.

The result of the planning of iteration n+2 and n+3 are actual changes in the issue management system. After the meeting, the planning for iteration n+2 is fixed. During this meeting topic, the entire project future is looked at, and often planning blocks in more future iterations are shuffled too already. Since most issue management systems do remember what happened to issues during their lifetime, but can not easily list which issues where shuffled on a given day, or why, each move and decision is probably best noted in the meeting report. The meeting report should mention explicitly which are the iteration n+2 and n+3 that are being looked at, that the planning of iteration n+2 was decided on in mutual agreement and was fixed, and what the exact dates are of their start and end, and what the appointments are for their post-iteration meetings.

Futhermore, each meeting report should mention explicitly when it took place, and who was attending, and who was invited, but could not attend.

Shifted iterations

Note that, during the post-iteration meetings, the contractor construction team is busy already on the work defined for the next iteration. The client team will only start testing after the meeting. The iterations are thus shifted in time for the client and the contractor. Also, the project manager needs some time to prepare the post-iteration meetings. In our experience, it makes sense to reserve 2 work calendar days for this preparation for the project manager. The total shift is thus 3 work calendar days.

As an example, suppose it is decided that post-iteration meetings will be held at Fridays. For the client, an iteration thus starts on Monday morning. For the contractor the previous iteration ended Tuesday evening (leaving Wednesday and Thursday for the project manager to prepare, for the meeting on Friday). For the contractor an iteration thus starts on Wednesday.

This is of importance for the definition of the actual product back log, bug list and research list. If the meeting at the end of iteration n is held at day D_n, it means that the topic of discussion during the post-iteration meetings are the issues recorded, bugs fixed, realizations completed, and planning blocks analyzed and estimated until the evening of D_n-3, from the morning of D_{n-1}-2. Issues recorded, bugs fixed, realizations completed, and planning blocks analyzed and estimated on D_n-2, D_n-1 and D_n are not discussed at the meeting at the end of iteration n, but only at the meeting at the end of iteration n+1.

Size of iterations

The main intend of the approach to project management described in this text is to deal with uncertainty and the resulting risk without stifling the project. It is important for both client and contractor to be able to be and remain confident in a positive outcome of the project during the project execution. We tackle this by formally planning execution, formally measuring progress, by frequent reporting and handing the client the power to decide continuously on the direction of the project, given the reality of the past and an estimate of the future.

We know that for clients a software construction project is a big thing. We are talking a lot of money, and a big impact on the day-to-day business of the client. This is a serious stress factor for the client. We also know that confidence and trust melts away rather quickly over time. When the client has confidence in a positive outcome today, based on the information and decisions made during a post-iteration meeting yesterday, his confidence level wll be less tomorrow, and diminish quickly. The client craves for a new update, to be reassured.

Any information is better than no information. Even a negative report on progress will raise confidence in the project. It is foremost vagueness and uncertainty that hurts confidence and heigthens the sense of risk, not so much deviations of the original plan.

Therefor, the time between post-iteration meetings should not be too large, and be constant and predictable. In our experience, and be measured in calendar time, and not in mandays or workload. Iterations of 4 weeks are in our experience the absolute maximum.

On the other hand, each iteration introduces overhead in building intermediate results, preparing the meetings and doing the meetings, which has more or less a fixed cost associated with it. Thus, very many iterations in a project will make it more costly. Furthermore, if iterations are too small, there is not enough time for the construction teams to come up to speed.

Therefor, an iteration should not be too small. In our experience, it should be at least 2 work weeks long.

Thus, in conclusion, iterations should be between 2 and 4 calendar weeks long. During holiday seasons, iterations can be made longer.

How much work can be done during that calendar time depends on the resources allocated to the project by the client. The size of the clients team is another issue of debate, and not the topic of this text. Furthermore, the total duration of the project is an issue too. In general, this text is aimed at client teams of 2 to 10 people, with a total duration of less than a calendar year. For larger projects, much of this approach can be kept, but other techniques are necessary on top of that. For very small projects, as we will see later, the approach needs to be adapted too.

Classification of issues, issue attributes, workflow, and list definitions

To estimate and plan, the final goal of a project will be subdivided in subgoals. How to do this best, is a dark art. In our approach, this differs for the intention of the subdivision.

Tasks

Certainly in a fixed price project, the contractor wants to make sure that the estimate on which the proposed price is based, is as accurate as possible. The best way know to us to do this, is to subdivide the final goal in subgoals of a technical nature recursively. The subgoals are expressed as chunks of work.

Tasks

To get a realistic estimate, we feel that we have to do this down to the granularity of more or less one manday. Some of the subgoals might be estimated to take up to 2 days, some will be expressed in hours.

The list of work chunks must be complete, and contain all possible work covered by the fixed price proposal (thus, not only development work, but also testing, project setup, project management, administration, and so on). Some estimates are based on factors of other estimates (e.g., project management is a 20% surplus on the sum of all the other estimates).

These subgoals make up the *initial project backlog*, and is what will be executed when the client agrees to the proposal.

These subgoals might be classified further, but for the process of the project, further classification is irrelevant: it is the work that was initially agreed upon between client and contractor. Further classification might be introduced for other analysis of the project. We just call such subgoals *tasks*, and remember as an attribute that they are *initial tasks*. They are aranged in a hierarchy if needed (*parent task*, *child task*). These initial tasks van only be created in an issue tracking system by the contractor, and have an estimate before the project starts.

Task dependencies

Tasks exist in a dependency structure: some tasks must be completed before other tasks can be executed. E.g., a class must be defined before a property can be added to that class.

Planning blocks

Because tasks are of a technical nature however, they are close to meaningless for the client. Thus, they cannot be used in a sensible discussion between the client and the contractor, such as the planning of a future iteration. Furthermore, the size of these tasks is too small to plan in iterations (you would get dozens of small subgoals in one iteration), and moreover, it would be difficult for developers to understand the larger picture of they would just execute random subgoals in the project. We need a larger unit of work for planning and communication with the client.

User stories

For this reason, *user stories* are introduced. User stories are little stories about the use of the final product by the client, in his terms, about part of the functionality.

In the approach described here, a user story is realized by tasks that are defined and estimated beforehand. Apart from the definition of the user story, we thus also express which of the tasks realize the user story. From that follows the estimate of the user story automically, as the sum of the estimates of the tasks it gathers.

We aim at user stories that express a work load of at least 2 days and maximum 6 days. This means a team of 2 developers has to organize its internal working for 1 to 3 days, and that there is synchronization with the main project line at least every 3 days. This size also makes for sensible planning of an iteration, which

is about 30 to 60 days. This means that the granularity of the user stories is a guiding parameter, and that sometimes larger user stories are split in different user stories to get this granularity. E.g., the user story where a user fills out a specific form and that data is stored persistently, with all user interface elements and feedback, and the handling of all possible execution paths and errors, might be split in a user story that tackles the flow of the form in itself, empty, a user story that adds a number of fields to the form, with all details, and a third user story that adds the final fields with all details. During 1 iteration, the subgoal of the first user story might be planned and reached, while the 2 later subgoals are only executed in the next or a later iteration. During this definition process, it is also possible that technical tasks are split, because 1 initial task needs to be split over 2 user stories. Of course, this should not change the total estimate. As such, the list of these planning blocks is just another view of the *product back log*.

In this approach user stories most often do not have an estimate of their own, as their work load is expressed as a list of technical tasks. However, in another approach, it is possible to make an initial estimate entirely based on user stories, without the preliminary definition of the technical tasks. This is certainly quicker, because the creation of the exhaustive list of technical tasks is done on the basis of a technical analysis, and doing the analysis and creating the exhaustive list of technical tasks takes time (and has a cost). We believe however that the resulting estimate is less acurate, since we have the experience that a lot of the complexity of a software project only becomes visible when we look at the technical detail. On the other hand, if the project is of a kind that is well known, this is not such a big issue. Still, an analysis and definition of a task list needs to be done before work on the user story can start, because the list of tasks is part of the way in which work is communicated and assigned, and progress is measured, during an iteration by the project manager, and an analysis is needed for the developers to know what to do, and to keep the result techically consistent. In an approach that starts with an estimate based on user stories, detailing the analysis and task list for a user story is then part of the normal flow, and is executed as a planned task itself in earlier iterations.

In any case, a user story must always have a result that is visible to the client. it should never be merely technical result. The client measures progress of the project in the functionality he gained during an iteration that he can see, touch and feel. Invisible work can never give the client the feeling of progress to is final goal, and will never add to the trust in the project needed, will never diminish the uncertainty the client has about the project. It is foremost for this reason that the total work of project should be divided and planned in function of user stories.

Other kinds of planning blocks

Yet user stories are not the only aggregation structure of tasks for planning and communication. Apart from technical tasks that are immediately and directly relevant for user stories, which should be part of the user story aggregate, although their effect is inivisble to the client in isolation, each project also has technical work that is needed for the project as a whole, but that cannot be attributed a particular user story. Examples are project setup, database definition, etcetera. Because this work needs to be executed in the iterations too, and thus needs to be planned, we want to aggregate such purely technical tasks in larger units too. This larger units are called *spikes*. And as shown in the previous paragraph, there might be analysis work to be done to. And there are probably more categories of *planning blocks*. Actually, apart from user stories, which are of extreme importance in the communication between client and contractor, the difference in the work is irrelavant for the process.

Planning blocks are aggregates of tasks, used when planning iterations, of a size between 2 and 6 mandays. User stories are the planning blocks that express together the total functionality of the final goal of the project. Because user stories are so important, and spikes are critical, we will classify planning blocks as *user stories*, *spikes*, or *other planning blocks*.

Like with tasks, it is important to remember which planning blocks are part of the initial agreement between the client and the contractor. Like with tasks, the user stories that are part of the *initial product backlog* are labeled with an attributed as an *initial planning block*.

Planning block dependencies

From the above, it should be clear that planning blocks can have a dependency relationship: a project setup planning block needs to be completed before other construction planning blocks can be executed. A form skeleton user story needs to be completed before a user story that fleshes out the form can be executed.

Non-initial work blocks

Both for tasks and for planning blocks, it is to be expected that additional work is defined during project execution. The process described above is build around this. This extra work is recorded as extra issues during iterations, and discussed in post-teration meetings.

Because this work is outside the initial scope of the project, we need to deal with it with the utmost care. Therefor, these extra issues are triaged in mutual agreement between the client and contractor continuously in the post-iteration meetings.

Bugs

Bugs are deviations in the tested software from the the functionality that was agreed upon to be completed in meeting at the end of previous iterations. There is no question that bugs must be fixed, and there is no question that this is at the expense of the contractor.

Bugs are a further kind of planning block. It might be interesting for the work organisation of the client to create additional tasks in the bug as aggregate, and estimate them.

Extra non-bugs

Other issues represent extra work that was not part of the initial product back log, which was not included in the initial estimate. The distinction is not relevant for the process described above, which is esepcially geared to changes in the final goal of the project. It is of the utmost importance however to make a distinction between initial goals and changes in the goals, to control the budget, end date and feature creep.

Extra work at the request of the client

During project execution, it is to be expected that the client will see the need or possbility for extra functionality, in details or in extra user stories. This is expected and supported by the process, yet must be handled deliberately and fully conscient of the impact on the project as a whole. It is clear that work associated with these extra user stories is outside the initial scope of the project.

The impact of these extra user stories or other non-bug planning blocks will be analyzed and estimated by the contractor, and a decision on the inclusion in the product backlog will only be made by the client after an analysis and estimate has been made. During the analysis and estimate, the contractor will probably create additional tasks, and possibly spikes or other planning blocks on which the extra user story is dependent.

Such extra user stories might also be created at the initiative of the contractor, but it is always the decision of the client to add them to the product backlog.

Oops, we missed that

Finally, it is well possible that work is recognized during project execution, in the form of user stories, spikes or other planning blocks, or as tasks, that was understood to be part of the initial scope anyway, but not recorded in the initial product backlog.

This is the responsibility of the party that created the initial backlog. It is also reported during the end-of-iteration meeting.

Extra or missing

Whoever created the extra issues, it is decided during the meeting at the end of iteration n, as part of the triage, in mutual agreement between the client and the contractor, whether the extra issue is extra or missing. We have, along this axis, thus 3 types of issues: *initial*, *extra*, or *missing*.

Scheduling and priority

Planning blocks are scheduled during the meeting at the end of iteration n-2 for iteration n and n+1. It is only at this time that the decision is final and the client is commited to execution of the work. Before this decision is made, stakeholders can move planning blocks to other iterations than originally planned, and even remove them from the project altogether.

To aid in this scheduling, the project backlog is divided in iterations from the start of the project on, as a proposal. Also, a next minor version and a next major version of the product is already defined in the issue tracking system, and planning items are attributed with a *priority*.

In our experience, a priority system must be very simple and have some absolute meaning. Thus, most often, we propose to use the MSCW (Moscow) scale, which stands for Must, Should, Could and Won't. User stories that are classified as *must* are an absolute conditio sine qua none in the eyes of the client, and spikes that are classified as *must* are an absolute conditio sine qua none in the eyes of the contractor. Issues that are classified as *should* are issues that should be included in the final result, and would hurt the satisfaction of the client with the final result if they are not included, but are not a conditio sine qua none. Issues that are classified as *could*, are issues that are nice to have, but that the client can live without. We do no have a real priority won't. Issues that are decided not to be included in the current project, are either closed, or moved into the next minor version or next major version, that is already foreseen as described above.

This classification helps the client to decide which planning blocks to tackle first. Apart from dependencies, it is highly advisable to plan the *must* issues first, even if they are *extra*, before we commit work to *should* or *could* issues.

Prioritization of issues is done initially, and continuously during the end-of-iteration meetings, during the discussion and triage of issues, the discussion and decision on planning blocks that are analyzed and estimated, and the planning of the iterations. Priorities of planning blocks can and will change as the project progresses, resulting in changes in where there execution is scheduled, in this version or a later version, and in this version, in an earlier or later iteration.

The prioritization of tasks that make up planning blocks is secondary. They inherit the priority of the planning block they are aggregated in.

Resolutions

Finally, we need to contemplate the end of an issue. This is called the issue *resolution*.

Tasks and planning blocks and bugs will nominally be closed when the work they represent is *done*.

Issues can however also be decided during triage to be created mistakenly, as a *duplicate* of an issue that already exists, or as something that really is *not an issue*.

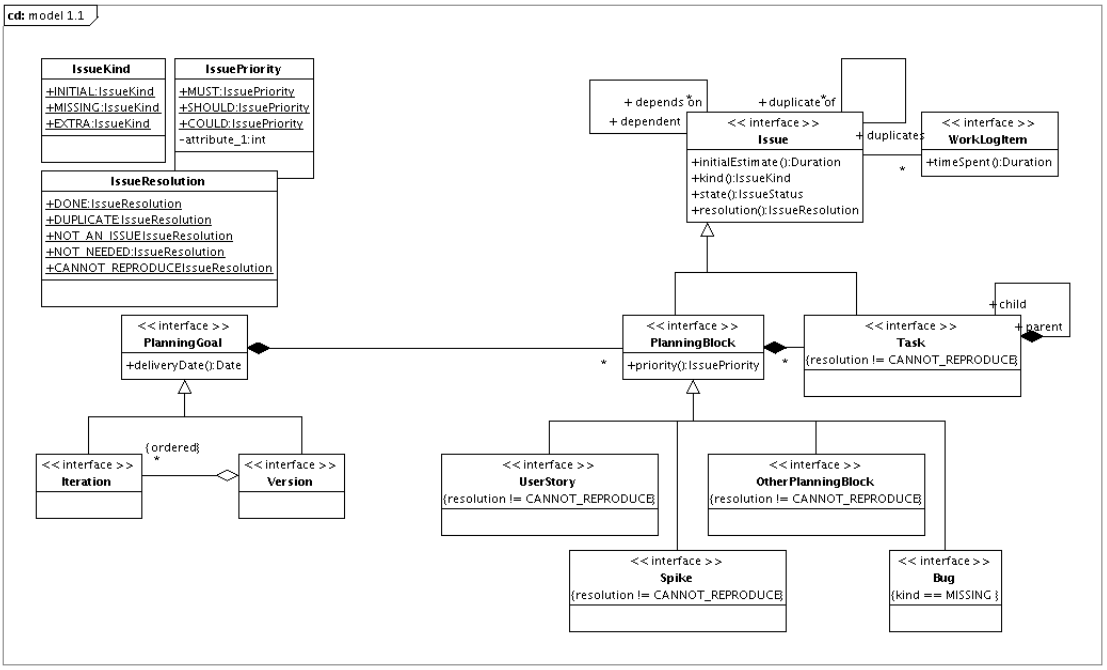
Furthermore, at any time, tasks and planning blocks might turn out not to be needed (*not needed*). This goes even for initial planning blocks and tasks.

Finally, during bug fixing, it is possible that the reported defect does not or no longer exist. The bug is then marked resolved with resolution *cannot reproduce*. The client than still needs to validate this and close the bug with this status, or reopen it with more comments.

Classification of issues

The following diagram summarizes the kinds of subgoals used in this approach:

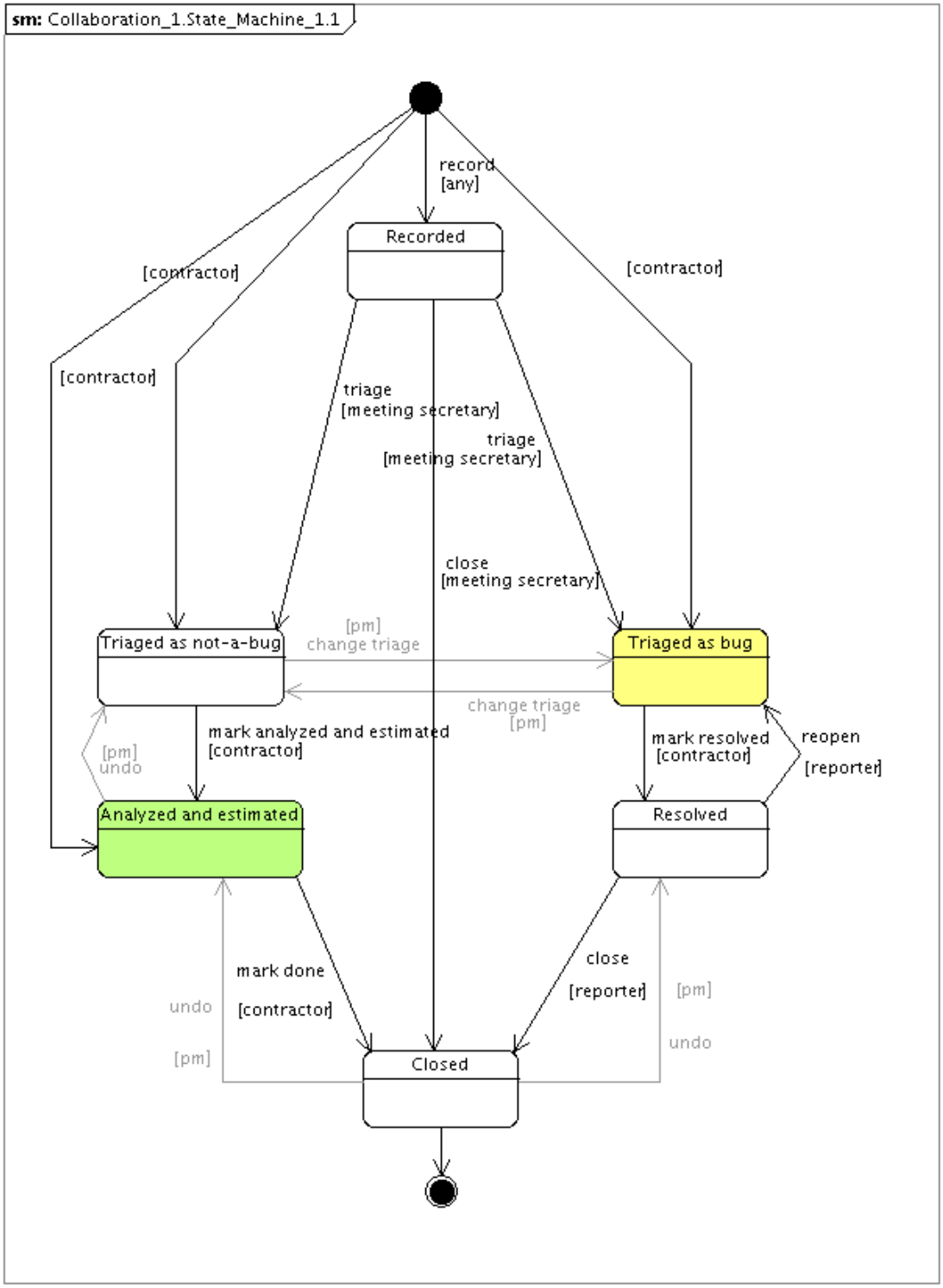
Figure 2. Classification of issues



Workflow of issues

The workflow of an issue is another representation of the process described in full above.

Figure 3. Issue state machine



Lists of subgoals

Finally, we can summarize here a definition of the 3 lists of subgoals used in the process.

- the *bug list* is the list of all issues of type bug that are not resolved for which triage has confirmed the classification
- the *research list* is the list of all issues that are not bugs and not resolved, for which triage has confirmed the classification, that are not associated yet with any version of the product; the result of the discussion and decision on planning blocks that are analyzed and estimated must be for these issues to either associated the issue with the current version, the next minor or the next major version, or be marked closed as *not needed* (or possibly, a *duplicate*)
- the *product backlog* of a version of the product is the list of all issues that are not bugs and not resolved, that are associated with the that version of the product; the product backlog has issues that are scheduled in iterations, and unscheduled issues

Since the iterations are shifted over 3 working calendar days, as discussed higher, it is important to define exactly in time what makes up the list of subgoals that are the subject of the post-iteration meetings and the iteration work.

- The list of issues to be discussed in the meeting at the end of iteration n at day D_n is the list of all issues created and bugs reopened, between 00:00h at D_{n-1}-2 and 23:59h at D_n-3.
- The list of bugs reported fixed at the end of iteration n at day D_n, which is the same as the list of bugs to be validated during iteration n+1 by the client, is the list of all bugs marked resolved between 00:00h at D_{n-1}-2 and 23:59h at D_n-3.
- The realizations demoed at the end of iteration n at day D_n, which is the same as the list of user stories to be tested during iteration n+1 by the client, is the list of all planning blocks marked done between 00:00h at D_{n-1}-2 and 23:59h at D_n-3.
- The list of planning blocks to be discussed at the end of iteration n at day D_n with regard to prioritization and their inclusion in the product backlog or not, is the list of all issues that are not associated yet with any version of the product, that are marked analyzed between between 00:00h at D_{n-1}-2 and 23:59h at D_n-3.

The product backlog, from which planning blocks can be scheduled in iterations in a post-iteration meeting, is the entire list as described above at the time of the discussion, without time limitations.

The bug list, i.e., the list of bugs that should be fixed at any given time, is not bounded in time, if we state that an issue can only be of type bug after it has been triaged.

The research list, i.e., the list of issues that should be analyzed and estimated at any given time, is not bounded in time, if we state that an issue can only be on the research list if it has been triaged.

Implementation in issue management systems

The above classification. attributes, workflow and list definitions are ideal representations. The process described in this document requires tool support for efficient management, and usually we find this in a more advanced issue tracking system such as Atlassian JIRA or Microsoft Team Foundation Server.

Depending on the tools at hand, the actual realization of the issue classification, attributes, workflows and list definition will differ from the ideal representation. It makes more sense to apply some usage rules ourselfs than to develop a custom solution that does exactly what we describe in this text.

The start and end of the project

At the start and end of the project, the nominal process as described for an iteration n cannot be followed completely.

The start of the project

At the start of the project, we cannot exactly follow the process as it is described for an iteration n in the middle of the project.

Before the project start-up

Before the projects starts, the contractor builds the initial product back log, with initial tasks, initial user stories, initial spikes and other initial planning blocks. The contractor distributes the planning blocks over iterations of the agreed upon size, and plans these iterations on the calendar, taking into account foreseable holidays, etcetera, and he allocates the resources needed.

This initial planning and schedule is only a proposal, but it will make the planning discussions during the end-of-iteration meetings more efficient.

Project start-up meeting

In a first meeting at the project start, commitments are made in mutual agreement about the planning for iteration 1, 2, and 3. This is the only topic on the start-up meeting.

Iteration 1

During iteration 1, there is no capcacity reserved for bug-fixing or analysis and estimates of new issues.

It is possible that during iteration 1 already issues (non-bugs) are recorded. In the meeting at the end of iteration 1, there is a discussion and triage of issues recorded during iteration 1, a demo of the realizations of iteration 1, and the planning of iteration 3 and 4.

Normally, the first iteration is twice as long as the other iterations. This is done because there is always some initial work, like setting up the project, and some initial spikes, that have work results that are not visible to the client, and it does not make sense to report finalization of those, since they cannot be demoed or tested by the client. Furthermore, often the project starts of with limited resources for this phase, because resource allocation takes some time, and also because these first planning blocks are more difficult to parallelize. Less work is thus done in the first iteration per unit of time.

Iteration 2

During iteration 2, there is no capcacity reserved for bug-fixing, but there is a limited capacity reserved for analysis and estimates of new issues.

The issues recorded during iteration 2 might contain bugs reported on the results of iteration 1.

In the meeting at the end of iteration 2, there is a discussion and triage of issues recorded during iteration 2, a demo of the realizations of iteration 2, there is now also a discussion and decision on planning blocks analyzed and estimated during iteration 2, and the planning of iteration 4 and 5. This is the full meeting agenda, except for the reporting on the fixing of bugs.

Up to speed

During iteration 3, there is capacity reserved for bug fixing and for analysis and estimates of new issues.

The meeting at the end of iteration 3 is the first meeting with the full agenda.

In other words, iteration 3 is the first iteration that is as described as a general iteration n in the middle of the project.

The end of the project

Also, at the end of the project we cannote exactly follow the process as it is described for an iteration n in the middle of the project. It is important for the contractor to know well in advance that the project will end, first and foremost to be able to effectively plan the people that are involved in the project: after this project, they will work on something else. The project manager of their next project needs to have a commitment about their starting time and time to prepare too. It is important for the client to know wel in advance that the project will end: after delivery of the final project result, the next steps, like acceptance, end user training, marketing, implementation, etcetera, will need to be executed. The person responsible for these tasks needs time to prepare and make arrangements too. This also means that, once we do announce the end of the project, it is a commitment to a lot of people, and the date cannot change anymore.

Iteration k

Suppose that you notice, during the planning at the end of iteration k, that the entire backlog is planned during iteration k+3. Unless very many planning blocks get cancelled or moved to a later version during this meeting, k+2 will still be planned full. Otherwise, you would have noticed this at the end of iteration k-1). When this happens, the decision is automatic that the end of iteration k+1 will be the moment of *feature freeze*: no more new planning blocks will be allowed on the product back log after that.

It thus does not make sense to analyze and estimate issues after iteration k+1. Whatever the outcome, they will not be added to the product backlog anymore. It still makes sense to record new issues during the final iterations, but they are not analyzed or estimated, not discussed or planned. They are issues for a next version.

Iteration k+1 and feature freeze

Iteration k+1 is an iteration like the previous one, except that, from now on, any new issues that are not triaged as bugs are put automatically in a next version, and will not be considered for inclusion in the product backlog of this version anymore.

During iteration k however, new issues that are not bugs might still have been added. These where recognized during the triage in the meeting at the end of iteration k, and put on the research list. Iteration k+1, which was planned at the end of iteration k-1, still has capacity reserved for analysis and estimates. This is still executed as planned.

The meeting at the end of iteration k+1 is follows the nominal agenda. New issues are discussed and triaged, the contractor reports on fixed bugs, there is a demo of the realizations by the contractor, and there is a discussion on issues analyzed and estimated during iteration k+1. This means that some planning blocks might indeed be added still to the product backlog, *but this is the last time*. Issues put on the research list at the start of the meeting are put on the research list of a next version, without further analysis and estimate.

During the planning, the final decision is made whether or not iteration k+3 will be the last one, or whether one or more extra iterations k+4, k+5, ... will be added. Before the discussion on issues analyzed and estimated during iteration k+1, there is work left over on the product backlog for iteration k+3. This could be very little, or almost fill out a complete iteration. With the issues added to the product backlog during the discussion on issues analyzed and estimated during iteration k+1, there might be too much work to complete in one iteration k+3. Depending on how much work is left, it is decided to either call iteration k +3 the last one, or k+4, or even later. If would be surprising however, and a testament to less involvement by the client in the earlier iterations, if the discussion on newly analyzed and estimated planning blocks necessitates the addition of many iterations only now.

The final iteration might be decided to be shortened (it does not make sense to wait another week if everyting is done). This final iteration we will call iteration N, where N = k+j, with j >= 3.

This decision fixes the delivery date of the final result of the project, and this decision is final. The end of the post-iteration meeting marks the *feature freeze*.

Note that from this moment on, it is possible to start a new project already, that works on the next version of the product under development. When concerned about merging code, it is better to wait until the project is completed, but certainly analysis and estimates, and the building of the initial product backlog for the next version can already start.

Iterations k+i = (k+2 .. N), where N = k+j (j >= 3)

For clarity of thought, we will call these iterations k+i, with i = 2 .. j, where j >= 3. It would be a surprise if j would not be 3 or 4, meaning that we either have an iteration k+2 and k+3 to do, or an iteration k+2, k+3 and k+4. Viewed from the back, iteration k, where the end is recognized, is thus most likely iteration N-3 or N-4 (after the post-iteration meeting, there are 3 or 4 iterations to go). Iteration k+1, at the end of which feature freeze is declared, and the delivery date is fixed, is most likely iteration N-2 or N-3 (after the post-iteration meeting, there are 2 or 3 iterations to go).

During the final iterations k+i, the work planned is executed by the contractor, and the client tests the result of previous iterations, and validates fixed bugs. During these iterations however, there is no capacity reserved anymore for analysis and estimates of new issues. Capacity for fixing bugs remains reserved, and can possible be heigthened.

At the meeting at the end of the final iteration k+i, during discussion and triage of newly recorded issues, they are either declared a bug, closed immediately, or merely scheduled for a next version. The contractor reports on fixed bugs, there is a demo of the realizations of the iteration k+i by the contractor. Since there was no analysis and estimates done during iteration k+i, there is no discussion of issues on a research list. Finally, The planning for iteration k+i is fixed (if i <= j) and the planning for k+i+1 is made (if i <= j-1).

For clarity, lets walk trough the last iteration explicitly:

Iteration N

During the final iteration N, there is no capacity reserved for analysis and estimates. This iteration might be decided in mutual agreement to be shorter than the previous iterations.

At the meeting at the end of iteration N, during discussion and triage of newly recorded issues, they are either declared a bug, closed immediately, or merely scheduled for a next version. There is a report on fixed bugs, and there is a demo of the realizations of iteration N. Since there was no analysis and estimates done during iteration N, there is no discussion of issues on a research list. There is no planning topic on the agenda, since there is no iteration N+2.

At the end of this meeting, the contractor declares the final goal of the project delivered.

After-iterations

At the end of iteration N, there are however

- possibly still bugs recorded during this iteration N, or left over from earlier iterations, on the bug list;
- the bugs fixed during iteration N still need to be validated by the client, which could result in reopening the bug;
- and the results of iteration N still have to be tested, which might result in new bug reports.

Therefor, there are a number of after-iterations, which are only intended to fix, resolve and close the final bugs. There is thus only capacity reserved for fixing bugs, and for nothing else during these after-iterations.

At the meeting at the end of these after-iterations, only new issues and reopened issues are discussed and triaged, and the contractor reports on the fixing of bugs.

These after-iterations are typically with less capacity on the side of the contractor, and more ad hoc. The first after-iterations are ususally shorter than previous iterations, and the last are on-demand of the client. The after iterations stop when the client accepts the project result or when a predefined number of iterations is reached or time frame has passed.

Limitations of the approach

Minimum project

From the above, it becomes clear that at least 2 iterations are needed for the project to get up to speed at the start, and at least 3 are needed for the project to wind down at the end. In the extreme case, there are not enough planning blocks on the initial project backlog to fill a 3th iteration completely, but there are enough issues to fill up 2). To use this approach as described, the minimum project size is thus 3 iterations, independent of the resources available to the project.

For really small projects, the resources should be limited to spread out the work over at least 3 iterations, and the iteration time should be minimized. In practice, this means that a client team of 2 persons, with 3 2-week iterations is the absolute minimum. With a first iteration that is usually twice as long, and the final iteration on average half the size of the other iterations, we get 4 + 2 + 1 = 7 work weeks with 2 people, or 70 work days as the minimum size of a project for which the approach as described is sensible. With the rules described above, mapping 80% of an ideal manday on a work day, and adding 20% overhead for project management, the minimum size of a project is thus is (70 * 80%) * 100% / 120% = 47 ideal mandays estimated construction time.

For smaller projects, much of the same approach can be used, but this falls outside the scope of this text. When, e.g., iteration time would be limited to 1 week, counting 30% management overhead, this leads to 2 + 1 + 0,5 = 3,5 work weeks with 2 people, or (35 * 80%) * 100% / 130% = 22 ideal mandays estimated construction time. With less structure, 2 people can do that in about 2,5 work weeks, without much overhead. That period is short enough to not invest in costly measures to reassure the client about the progress.

Taking into account earlier considerations, given that the main goal of this approach is to reassure the client and contractor about the positive completion of the project, and that confidence can, in our experience, not be stretched out much beyond a calender month (20 work days with 2 people), it is advisable not to use this approach with projects that are less than about 50 ideal mandays estimated construction time.

Maximum project

Taking into account earlier considerations, stating that the total maximum duration of a project with this approach is a calendar year, and that the maximum size of the client team is about 10 people, we can find out what the maximum size of a project is for which the approach described in this text is applicable.

Given that a year consists of 12 months, for which we dare only to plan 16 work days per month, we take into account 200 work days per calender year. This amounts, with a team of 10, to 2000 work days. With the rules described above, mapping 80% of an ideal manday on a work day, and adding 20% overhead for project management, the maximum size of a project is thus is (2000 * 80%) * 100% / 120% = 1333 ideal mandays estimated construction time.

For larger projects, much of this approach can be kept, but other techniques are necessary on top of that. This falls outside the scope of this text.

Because of the amount of results a 10-person team can produce in a given time period, it is advisable to keep iterations short in such a project. Taking into account holidays, this would result in a project of about 20 2-week iterations, of 100 work days each, making up for the total of 200 work days in a calendar year.