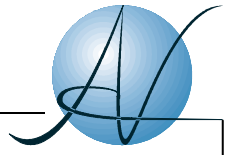

CONQUEST 2006

Planning Software Development

TAKEN AS A MANUFACTURING CONTROL PROBLEM

BERLIN 29.09.2006

DI. Andreas Nehfort

Abstract

"All models are wrong, but some are useful!"

Project planning and management is the commonly preferred approach (model) for planning and controlling software development. In this paper I want to point out that the project approach is not always useful for planning software development! In many cases the management of software development has more in common with managing a flexible small lot production than with a project.

Consequently, in those cases the project paradigm makes planning and especially re-planning unnecessarily complicated, tricky and cumbersome. A paradigm shift (from project management to manufacturing control) will replace the project task as primary planning variable by the resources. This can reduce your planning complexity and increase your planning success dramatically.

*Agile planning approaches work with this paradigm shift. This paper will also illustrate **how** and especially **why** agile planning works.*

1. MOTIVATION

During my consulting work I have dealt with lots of IT-projects. In the past five years many IT companies have invested a lot of money to improve their project management skills.

In spite of this, more and more projects fail in the management of change. Project managers struggle with an increasing number of minor or major changes. These projects are characterised by time-consuming Jour-Fixe meetings and frequent pre-planning with slipping schedules and due dates, which results in unreliable plans.

Some of these projects called me to bail them out. Based on my consulting work, I have established a set of techniques and rules to handle a high rate of change requests. Reflecting my work I realised that these techniques sounded familiar to me. They reminded me of my work running a manufacturing control station for assembling printed circuit boards years ago. These similarities were not accidental - from my point of view many of these software development "projects" had more in common with managing a flexible small lot production than with a project.

Today, more and more IT projects are of this kind. Requirements often have become moving targets. Consequently, in these cases a paradigm shift (from project planning to manufacturing control) can reduce their planning complexity dramatically. Applying manufacturing control techniques and some simple rules can help them take control over their problems quickly.

2. THE PROBLEM - ILLUSTRATED WITH AN IMAGINARY PROJECT

Note: similarities to real world software development are intended – not accidental!

2.1 The starting point

UsefulSoftware Inc. is developing Rel. 3.1 of their leading product UseMe. During the planning phase in January the product manager and the manager of the SW-development department have agreed to implement 10 new features. Initially the product manager had at least 12 to 14 new features in mind, however he had to accept that with the available team (8 software engineers) they would implement "only" 10 features if he wanted Rel. 3.1 to be finished until the end of September to be able to present Rel. 3.1 at the large exhibition early in October.

2.2 The project evolves

The implementation of the 10 features was planned – the plan was ambitious but not impossible. At this point the change (hi)story began:

- Feb. 15th: An urgent bug fix (and patch) for Rel. 3.0 is required. The analysis shows that the problem is tricky and the solution costly.
- March 2nd: Mrs. Meyer – who is leading the UseMe implementation project at TheBigOilCompany – needs a new feature until April 10th to fulfil the contract.
- March 20th: The product management has a change request: a new feature is required for Rel. 3.1 to respond to a competitor's product announcement (therefore feature 3 could be postponed - if absolutely necessary)
- In April the developers find out that feature 6 cannot be implemented as planned (further investigations are required to find an adequate solution).
- And so on … each month one or two additional major requests and lots of minor ones, all of which have been handled by change management …

2.3 The result

At the end of September the development team has implemented

- 4 of the initially 10 planned new features
- plus 2 features proposed by the product manager during development.
- plus 3 features needed by customer implementation projects
- plus 2 patches for Rel. 3.0: the urgent patch from March and a "service pack" solving different problems in July

The output is 11 features/patches instead of 10. But only 4 of them reflect the initial plan for Rel. 3.1. The overall performance seems not too bad, but the result does not satisfy – nobody is happy with Rel. 3.1. The cost overrun amounts to approximately 25%.

The project from the perspective of the development team

The development team has been under pressure from Feb. 15th to Sep. 30th; working overtime was a matter of course; 60 working hours per week were no exception.

In weekly status meetings, most of the time was spent on telling each other WHAT has NOT been done, and WHY it could not be done – despite the plans and agreements. The rest of the time was spent on trouble-shooting to handle new requests (tasks) and new priorities.

The plans have been (r)evolved frequently; many decisions had to be delegated to the change control board. It was frustrating that most features and tasks were not according to plan.

The project from the perspective of the management

The product manager and the head of the UseMe business division were not satisfied with the poor development performance: Only 4 of the 10 initially planned features had been implemented (let me remind you that the product management wanted 12 to 14 new features – and 10 was a compromise …). Unfortunately, the announcement of all the 10 features has started in February (the customers at the large exhibition in October asked for the missing features … those were hard days for the marketing team).

The 4 features needed by customer implementation projects and the 2 patches are not worth mentioning because these were daily work requirements we simply had to deliver - if we were not able to manage them we could as well close the company and go home!

The release 3.1 from the customer's perspective

Most of the customers were disappointed about UseMe Rel.3.1. Many Users of Rel. 3.0 or Rel. 2.x waited for some of the features which had been announced but were not implemented in Rel. 3.1. They had planned to upgrade to Rel. 3.1 in the last quarter of the year. They felt insecure, because this was not the first time the product did not comply with the announcement.

2.4 Your opinion of our imaginary project?

What do you think about this imaginary project?

- Does this story sound totally strange to you - overstated and crazy?

Or do you see similarities to real world software development projects in your surroundings?

- Have you ever heard about such a project?
- Does the story sound familiar to you?
- Have you even worked in such a project?

3. THE ANALYSIS

From my point of view our imaginary project does not have much in common with what we call a "project"! - What is a project? The "PMI - Project Management Body of Knowledge" – PMBOK [1] defines "project" as follows:

> *"A project is a temporary endeavor undertaken to create a unique product or service." "Temporary means that every project has a definite beginning & a definite end." "The end is reached when the project objectives have been achieved, or … the project is terminated." … "Unique means that the product or service is different in some distinguishing way from all other products or services."*
>
> *"For many organizations projects are a means to respond to those requests that cannot be addressed within the organization's normal operational limits."* PMBOK 2000

Is our endeavor temporary? Does it have a definite end?

- With respect to the release date (End of September) it has a definite end!
- With respect to the "project objectives" (→ initially planned feature list) it has not!

Does our endeavour create a unique product? Does it have a "project objective" in the PMBOK's sense? Let's have a look at what the PMBOK says about the "project objective":

> *"The objectives of projects and operations are fundamentally different. The objective of a project is to attain the objective and close the project. The objective of an ongoing nonprojectized operation is normally to sustain the business. Projects are fundamentally different because the project ceases when its declared objectives have been attained, while nonproject undertakings adopt a new set of objectives and continue to work."*

I think that makes it clear: The development of Rel. 3.1 is no typical project work! It has the characteristics of an "ongoing nonprojectized operation":

- The main objective of our endeavour called "project" was to produce a new release until the end of September. The "project objectives" regarding the content (the set of features) was highly unstable: Initially, there were 12-14 features, then 10 features defined as project scope, and in the end we had 4 of the initially planned features and some others which had arisen during development.
- What is our project team going to do at the end of September? Close the project? No - they will tackle the next set of features to implement Rel. 3.2 – which means they will "adopt a new set of objectives and continue to work".

These are the characteristics of non-project work! The main tasks are

- Implementing features
- Bundling features to a product release

Software development under such circumstances may be regarded as a manufacturing problem:

- A single feature or request can be taken as production order
- The set of features can be taken as order backlog

Agile development works with such metaphors and provides adequate planning techniques. Mary Poppendieck [2] has published "Assembly-line production techniques apply to software, too" in 2001 – see also [4] and [5].

Note: Contrary to the development of rel. 3.1 the UseMe implementation project at TheBigOilCompany is a project. What will the project team do at the end of the project? They will close the project and start a another implementation project with totally different stake holders, business goals, agreements, and so on … the achievements for TheBigOil-Company will not affect our new customer – each customer site implementation is a "unique product or service".

4. DIFFERENT PLANNING APPROACHES TO PROJECTS AND PRODUCTION

4.1 The project planning approach

Classic project planning is based on the project objectives and the corresponding project scope (product or service). The problem / project-task represents the amount of work. For the first cut we treat the problem (amount of work) as constant → it is considered as the independent planning variable. Then, we try to determine (estimate and plan) the corresponding effort (resources) and time schedule needed to do the work → resources and time are the dependent planning variables.

If the planning result is not satisfying, we have to adjust resources and time to meet the project objectives. If we cannot meet the project objectives by adjusting resources and time, we have to reconcile work, effort and time – in that sequence - to re-dimension the project until the plan seems to be feasible and satisfying.

If the project scope changes later on or relevant deviations from the plan become obvious, we have to re-plan the project (→ change management & re-planning).

If the project scope is a moving target - like in our imaginary project – re-planning based on the "amount of work" as independent planning variable is a cumbersome task.

Note: In software projects relevant deviations of the estimated and the actual effort are quite common. This means that even when the project scope is quite stable, the amount of work may be higher than estimated and may cause relevant deviations from the plan.

In projects, the method to achieve a satisfying performance is to design and implement an appropriate project organization. If the project scope is a moving target, this method will be of limited effect. You cannot redesign your project organization on a frequent basis, but you may try to establish a flexible project organization. However, for more flexibility you typically have to put up with decreased productivity …

4.2 The production planning approach

Production planning for a flexible small lot production is not based on the "production goal" or the order backlog – it is based on the available resources. The amount of resources available per time period is considered as the independent planning variable. In many cases we can treat the available resources as constant – at least for short-term planning. What can we plan? We can allocate orders and the time to the available resources - these are the dependent planning variables.

In the order backlog the production orders are prioritized with respect to the requested delivery date and classified with respect to the resources they need.

Based on this priority and resources requirements, the production orders are dispatched to the available resources.

If the planning result is not satisfying, we have to reconcile priorities and resource allocation (dispatching), and if that does not help, we have to arrange for a new delivery date.

Dispatching can be oriented on different optimization goals. Typical antagonists are:

- Optimizing throughput (maximum productivity)
- Optimizing throughput time (short-term delivery)

It is a matter of fact regarding production planning that you cannot optimize in both dimensions simultaneously without making compromises. Another optimization strategy is:

- Optimizing from bottleneck resources

Optimizing througput

Optimizing throughput is focused on minimizing non-productive times (down-times, set-up times, re-tool-times) by an appropriate sequence of production orders and an appropriate parts supply. Changing order sequence will typically lead to increasing set-up-times and therefore reduced throughput. Hence, optimizing throughput means limited flexibility. You will achieve the maximum throughput on a single product assembly line, which typically is very limited in flexibility.

Optimizing throughput time

Optimizing throughput time is focused on minimizing the time between order and delivery (short-term delivery). Minimizing throughput time implies short-term changing of priorities and production sequence. This requires flexible workers and machinery, a just-in-time parts supply, workload with reserves for rush-orders and so on. It is obvious that these measures will reduce throughput compared to the concepts above.

Optimizing from bottleneck resources

Many production planning problems result in bottleneck resources, which cannot be increased on a short-term basis. Hence, the overall output is limited by the bottleneck. In this case, production planning usually starts with planning the bottleneck resources to optimize their throughput, e.g. by minimizing their set-up-times. Then the rest of the work to

achieve an optimized production order for the bottleneck resources is planned. This implies sub-optimal work-loads for the non-bottleneck resources. The supply-chain before the bottleneck must be focused on causing no downtimes at the bottleneck due to supply problems, and the assembly line behind the bottleneck typically does not work at full capacity.

4.3 The conclusions

Should the characteristics of your software development "project" look like a "production problem", you can benefit from treating it as a production planning problem.

Rephrasing the planning problem

Re-planning by changing the dependent planning variables is always simpler than re- planning by changing the independent planning variables! All planning techniques focus on adjusting the dependent planning variables to achieve the goal, with respect to the value of the independent planning variable. So the control loop is closer compared to adapting the independent planning variable!

Once you have accepted that the resource (in man-days) is your most reliable planning parameter, it is just a small step towards the execution of the paradigm shift: to replace the "project objectives" (=the work to be done) by the "resources" as independent planning variable.

Focusing on the resources as independent planning parameter suddenly makes you realize that your planning problem is a different one:

- First (focusing on the "project objectives" = the work to be done): How can we manage the (whole) project task due to limited resources and schedules? Often this problem turns out to be an impossible task[1].
- Afterwards (focusing on the resource limits as primary planning parameter): How much of the required or desired results can we produce at which delivery date – and which features fit best to achieve the project objectives.

Rephrasing the project order

If we want to focus on the resources as primary planning parameter, we have to say good-bye to our perception of a project as a kind of contract in which the requirements are the subject matter, as all articles of a contract are of the same importance. If you do not fulfill them all you break the contract.

Once you have agreed to this contract, the ordering party (client) consequently has no motivation to prioritize the features or requirements. You have agreed to fulfill them all – typically for a fixed price or budget and a fixed schedule. So why should they resign from single requirements? They must really have a good reason to do so!

Planning with a focus on the resource limits as primary planning parameter needs a prioritized list of features or requirements allowing you to insert new items, to upgrade or downgrade the priority of items or to delete items.

The agile manifesto [7] reflects this need with the following values:

- "Customer collaboration over contract negotiation"
- "Responding to change over following a plan"

[1] Mostly we don't call it "impossible task", and we try hard to succeed – but in fact, often it is an impossible task – and sometimes we know it from the beginning …

5. RECOMMENDED RULES AND TECHNIQUES

The following rules focuson achieving 2 rewarding goals:

- **Increasing planning reliability** by planning a feasible workload, which takes into account the additional workload caused by short-term and ad-hoc requests.
- **Increasing productivity** by minimizing down-times due to interruptions caused by requests, which have to be handled (analyzed, estimated, …) and the succeeding re-planning. You may remember: "In weekly status meetings most of the time was spent on telling each other WHAT has NOT been done, and WHY it could NOT be done – despite the plans and agreements". Too much resources have been blocked by request handling and trouble shooting.

5.1 Resource partitioning due to different planning horizons

In our imaginary "project" we have 4 planning horizons:

- **Long-term:** future releases - coming after the release under development. In our imaginary project, the long-term planning horizon is represented by all the features which have been postponed to Rel. 3.2 and higher. The long-term planning horizon is typically about one year or longer.
- **Mid-term:** the features for the release under development at the beginning of the planning phase. In our imaginary project, the mid-term planning horizon is represented by the 10 features planned in January for Rel. 3.1 (we know – only 4 of them have been implemented in Rel. 3.1 …) Typically, the mid-term planning horizon is about 9 to 12 months.
- **Short-term:** the features for the release under development which arise during development. In our imaginary project, the short-term planning horizon is represented by change requests – you may remember: the 2 additional features proposed by the product manager and the 4 features urgently needed by customer implementation projects – and some other requests which have been postponed … Typically, the mid-term planning horizon is about few weeks to months.
- **Ad-hoc:** the urgent fire fighting and trouble shooting actions which arise during development. In our imaginary project, the ad-hoc planning horizon is represented by the urgent request for a patch for Rel.3.0 (in March) and other tasks we have not mentioned in Chapter 2. Typically, the ad-hoc planning horizon is about a few days to several weeks.

At the beginning of our imaginary project only the mid-term planning horizon has been taken into account – so any request disarranged the plan.

It seems to be avisable to plan the resources for three planning horizons: for mid-term and short-term requested features and for ad-hoc requested fire fighting and trouble shooting actions.

However, we only know the mid-term requests. They are typically specified when we set up the plan, and we know that there will be some short-term requests and some ad-hoc action. The rest of these requests is unknown: We do not know their number, the time when they will be made, their kind and the effort to be made to allocate resources.

Extreme programming [3,4] and SCRUM [5] try to solve that problem by short-term iterations (about a month). The intention is to plan in detail for the next iteration (the next 4 weeks) and to freeze the set of features for this iteration. For the next iteration all the requests accumulated in the last period may be included in the new list of features which may have new priorities.

Short iterations cannot prevent us from the necessity to look into the mid-term future (6 - 12 months ahead) to get an answer to the question how many items on our feature list we can we tick off by September.

With some practical experience (and some data from the past) we can find a reasonable answer when we split our resources in the following way, e.g.:

- 50% for mid-term features
- 35% for short-term requests

- 15% for ad-hoc actions

Note: If we assume that for ad-hoc trouble shooting the estimated 15% will be needed in any casethe 35% reserved for short-term requests must also compensate our possible estimation errors at the mid-term features.

If we surprisingly do not spend resources booked for short-term and ad-hoc requests as planned, we can gradually deblock these resources for implementing the next features from the feature list. This strategy leads to the "50% rule" and the "1/3 rd rule"

Note: These rules are intended as a way out in a situation as described in our imaginary project. They are definitely not meant as ultimate solutionbut work quite well to improve planning reliability rapidly. Hence they can be a starting point for finding an own agile planning strategy.

5.2 Prioritizing rules - The 50% rule

The 50% rule is intended for following situations:

The effort estimates are not reliable (typically over-optimistic), and we expect some ad-hoc request and few short-term requests, or the effort estimates are not so bad, and we expect a couple of short-term or ad-hoc request.

The 50% rule works as follows:

- 50% of the resources shall be allocated to priority-1-featues (must-have features)
- 50% of the rest (25% overall) may be allocated to priority-2-featues (should-have features).
- The rest (25% overall) is booked for, estimation errors, other problems and requests.

The best case result will typically be:

All priority-1 and priority-2 features will be implemented in time and budget (if the interferences caused by requests and estimation errors and problems add up to less than 1/3 of the planned effort for prio-1 and prio-2 features)

The worst case result will typically be:

At least the priority-1 features will be implemented in time and budget (if the interferences caused by requests and estimation errors and problems add up to less than 100% of the planned effort for prio-1 features).

The effect:

Even in the worst case scenario we need not worry about the prio-1 features! They will be available on schedule! In most cases, you will get all prio-1 features and some of the prio-2 features. As a reasonable compensation for the prio-2 features which have not been implemented the most urgent requests and troubles could be dealt with.

5.3 Prioritizing rules - The 1/3 rd rule

The 1/3rd rule is intended for following situations:

The effort estimates are not reliable (typically over-optimistic), and we expect a couple of short-term and ad-hoc requests, or the effort estimates are not so bad, and we expect many short-term or ad-hoc requests.

The 1/3rd rule works as follows:

- 1/3rd of the resources shall be allocated to priority-1-features (must-have features)
- 1/3rd of the resources may be allocated to priority-2-features (should-have features).
- The rest (1/3rd overall) is provided for estimation errors and other problems and requests.

The best case result will typically be:

All priority-1 and priority-2 features will be implemented in time and budget. (If the interferences caused by requests, estimation errors and problems accumulate to less than 50% of the planned effort for prio-1 and prio-2 features)

The worst case result will typically be:

At least the priority-1 features will be implemented in time and budget. (If the interferences caused by requests and estimation errors and problems accumulate to less than 200% of the planned effort for prio-1 features).

The effect:

Even in the worst case scenario we need not worry about the prio-1 features! They will be available on schedule. In most cases you will get all prio-1 features and some of the prio-2 features. As reasonable compensation for the prio-2 features which have not been implemented, the most urgent requests and troubles could be dealt with.

Both rules are based on the consideration that you cannot ad-hoc improve estimation accuracy and the way an organization handles their requirements or requests. But you can ad-hoc change your strategyon your promises for the next planning period – if you really want to.

5.4 Project controlling techniques

Some controlling techniques which will help to get the process under control:

- Measuring the requirements volatility (breakage factor – see Chapter 6.1) helps us to estimate the percent-age of work caused by short-term and ad-hoc requests
- A feature completed trend line helps us to measure the progress of work (the project velocity)
- The earned value analysis [1] helps us to focus on completing work packages or features instead of starting new work and leave already started work packages open, because the value will not be earned until the work package has not been finished completely! It may also help to identify a systematic estimation error (overoptimistic estimates) when your data are not detailed enough to make a target/actual effort comparison for single work packages.

6. ADOPTED CONCEPTS AND TECHNIQUES

6.1 The breakage-factor from CoCoMo II

CoCoMo II (Constructive Cost Modelling) [6] is a cost estimation model developed by Barry Boehm and his team at the USC-CSE (University of Southern California - Center of Software Engineering).

Breakage reflects the requirements volatility in a project. The breakage-factor represents the percentage of features which will be changed, added or deleted during the project. In simple terms, it is the percentage of work we have to do more than is represented by the work breakdown structure based on a specification or feature list.

Breakage is the part of work we cannot estimate based on our specification or feature list, because it is not known at this time.

6.2 The planning game from XP or SCRUM

The planning game as described in XP (eXtreme Programming [3,4]) or SCRUM [5] helps us to prioritize features and select as set of features for the next iteration, and it is based on the following principle: “The customer is responsible for business decisions, the supplier is responsible for technical decisions”.

6.3 The SCRUM sprint

The SCRUM sprint is a technique to balance flexibility and productivity. The development is divided into short iterations (30 days - called sprint). The sprint planning meeting defines the sprint backlog (= set of features to implement in this sprint).

During the sprint, the sprint backlog is executed –and no external changes are allowed.

This helps the team to work at top speed without interruptions due to new requests or changes. The scrum sprint is a technique focused to minimize non-productive times (down-times, set-up times, re-tool-times).

6.4 Conclusions and Recommendations

If the characteristics of your software development "project" look like a "production problem", you can benefit from treating it as a production planning problem:

- Rephrase the planning problem to treat the resources as primary planning variable.
- Treat the resulting output as dependent planning variable –a factor which has to be planned and managed during the project instead of being defined at the beginning.

You will see - you will get a different planning and management problem:

- Plan and manage the features which shall be realised as the variable factor.
- Identify the "must-have-features" and check whether you can provide them according to your needs; if not, you should better not try to carry out the project – it might be an impossible task.
- Help the stakeholders to define a feature list and to prioritize the features.
- Increase the planning reliability by splitting the resources for mid-term, short-term and ad-hoc requests. If you do not have any experience values from the past, the 50% rule and the 1/3rd-rule can help you to start making plans which are robust enough to withstand estimation errors and additional requests.
- Strictly prioritized features will make you flexible and will help you to maximise your earned value. As long as you have not started with the detailed specification and the implementation you can still change priorities with minimal losses. Otherwise, if you have started with the detailed specification and the implementation you should complete it with high priority. If you cancel or postpone the work on already started work packages, your losses (stranded investments) will typically exceed your savings.
- Strictly prioritized features and robust plans will help you to increase productivity due to a reduced effort for re-planning and motivated staff.

Based on my experiences with some clients which maintain and enhance existing IT solutions or develop IT products, I would like to point out that this is not only work for difficult project settings – it is a solution, if the prioritizing of features and the splitting of resources can be turned into practice! It can help to reduce planning complexity dramatically and to increase planning success in situations where planning reliability is unknown. If you want to apply an agile process, this paper can help you to understand the purpose of some of its rules.

References

[1] PMBOK® guide 2000 Ed.: A guide to Project Management Body Of Knowledge
PMI Project Management Institute Inc. © 2000
Note: The actual edition is the PMBOK® guide 2004 Ed.

[2] MaryPoppendieck: Lean Programming Part 1 and Part 2;
SD Magazine, May & June 2001; www.sdmagazine.com

[3] Kent Beck: Extreme Programming Explained
Addison Wesley © 2000, ISBN: 0-201-61641-6

[4] Kent Beck, Martin Fowler: Planning Extreme Programming
Addison Wesley © 2001, ISBN: 0-201-71091-9

[5] Ken Schwaber: Agile Project Management with SCRUM
Microsoft Press © 2004

[6] Barry W. Boehm et al.: Software Cost Estimation with Cocomo II
2000; Prentice-Hall; ISBN 0-13-026692-2
see also → http://sunset.usc.edu/research/cocomosuite/suite_main.html

[7] The Agile Alliance (Kent Beck et al.): Manifesto for Agile Software Development; 2001 → http://www.agilemanifesto.org

Authors' biography - DI. Andreas Nehfort:

- Consultant for IT & Software Process Improvement (SPI); Nehfort IT-Consulting → www.nehfort.at Focus: Software Process Models and Software Process Improvement – Agile Processes, Assessment Based Process Improvement → CMMI / SPiCE, Project Management & IT Quality Management.
- Intacs Certified ISO 15504 Assessor → SPICE: Software Process Improvement and Capability dEtermination
- since 1988 Trainer for IT Project Management, Software Engineering & SW Quality Assurance
- since1986 self-employed → Nehfort IT-Consulting → www.nehfort.at.
- since 1982 Project Manager, Requirements Analyst & Consultant for Software Engineering & Software Processes
- 1978: start of my professional career as Software-Developer;
- 1975 – 1979: Study of Technical Mathematics at the Technical University of Vienna – certificate Dipl.Ing. (DI.)