JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 2, March - April 2006

# Planning before plans

**John D. McGregor**, Clemson University and Luminary Software LLC, U.S.A.

**Abstract**

A plan is a blueprint for solving a problem. It summarizes the decisions that have been made during the planning of that solution. In this month's issue of Strategic Software Engineering, I want to talk about the strategic importance of planning in making plans.

## 1 INTRODUCTION

The television schedule is full of so called reality shows where otherwise reasonably sensible people do ridiculous things to win prizes. In software development we see the same phenomenon when otherwise sensible professionals write plans, without doing any planning, just to meet a deadline. I have encountered this many times but it never fails to amaze me. This is a direct result of deliverable-driven processes. These processes reward creating the document because it is measurable and tangible as opposed to rewarding the thought that goes into planning which is much less tangible.

Answers.com says a plan is: "A scheme, program, or method worked out beforehand for the accomplishment of an objective". "Worked out" is my focus this month. A plan is worked out by specific actions and decisions that go before the actual writing down of specific actions. When a plan is written before the method for accomplishing a task is "worked out," decisions are made implicitly without sufficient analysis of the trade-offs that are almost always required for an effective plan.

Effective is the key word. I don't have some theoretical objection to plans made without appropriate planning. They just aren't as effective. How can we know this? By autopsying failures. Often by tracing back we find that plans that fail do so because of factors that were present and knowable prior to the plan being executed. That is a faulty plan and one that could have been effective if there had been sufficient planning.

Planning is defined to be "an act of formulating a program for a definite course of action." There are specific actions that occur during planning that lead to a better plan than those produced by off-the-cuff writing. I will get into those actions later. For now, it is sufficient to say that writing a plan should be preceded by formulating a course of action.

Developing a software intensive product is a complex endeavor that requires much planning and a number of plans. Management plans include resource allocations, development schedules and release plans. Technical plans include architectures, test plans, configuration management plans and quality assurance plans.

What is needed is a very efficient planning process. Some agile methods do this by having very short term plans such as planning for a 24 hour period during the daily Scrum or for a 30 day Sprint [Schwaber 02]. Their hypothesis is that plans of longer duration have to be revised anyway because of the constant churn in a development project so shorter plans involve less waste of time. Indeed I have observed projects that were in a constant state of replanning due to plans being out of date by the time they were completed. However, even projects using agile development techniques usually must have some kind of longer range plans that show customers the feasibility of delivery dates of a completed product.

I do planning but often I don't produce a plan document. Is that consistent with my position of planning before plans. Absolutely. I only need to communicate the result of my planning as appropriate and that may occur in many ways other than a written document. The essential elements, to me, are well reasoned decisions that have buy-in from all the appropriate stakeholders and a well thought out sequence of actions. The act of planning communicates these decisions and actions.

In the next sections I want to first consider how to get the most out of planning and then follow that with a discussion of constructing plans.

## 2 PLANNING

Planning is often tedious and seldom quick. Planning involves deciding the specifics concerning how some high level task will be performed. Planning occurs prior to the actual action so that we have thought through the entire sequence of actions prior to carrying out any of the actions. Boehm and Turner characterize agile development as planning driven rather than plan driven [Boehm 03]. Agile teams spend little time communicating "the plan," but conduct planning sessions on a regular schedule.

Planning has its own rewards. Honest, it does. For one, you actually have a chance of achieving your goal if you have planned how to achieve it. For another, projects that use planning can often eliminate some mistakes before they happen or at least before they become expensive to fix. For example, beginning test planning in the very early stages of a project often finds many of the requirements defects. This saves a large percentage of the resources that would be required to fix the defects later.

Planning is the most strategic activity in which we engage because the very nature of planning causes us to look forward and to consider options and goals. Planning before doing gives us the luxury of considering the implications of each action which is much harder to do if we are caught up in ongoing operations. An effective planning process contributes directly to the success of sofware development by eliminating wasteful actions that don't contribute directly.

## Planning Process

The act of planning requires longitudinal thinking and reveals logical linkages. These processes help illuminate conceptual gaps and even some incorrect assumptions so that the resulting plan will be an accurate description of what has to be accomplished and how it can be accomplished.

Consider the following high-level planning activities:

**Establish the high-level goal** – The goal may be to create a deliverable or accomplish some intermediate step toward completion of the deliverable. This activity should be revisited periodically so that planners can remind themselves of the goal. This keeps the planning focused and on track.

**Scan the environment** – This step identifies the forces that will facilitate achievement of the goal and those forces that will impede efforts to achieve the goal. This step provides the essential context and constraints that make the resulting plan realistic.

**Analyze the high-level goal** – The high-level goal must be converted into an operational goal or set of goals. To me this is the most often miss-handled step. The planners try to derive a concrete set of activities for a high-level goal but the goal is not sufficiently specific to allow measurement. The goal(s) in a good plan should be specific enough to be measured and sufficiently broad to be strategically significant accomplishments.

**Develop strategies and tactics** – For each operational goal, a strategy needs to be adopted and tactics identified. I have previously discussed the use of Porter's Five Forces strategy development approach in the context of developing a production strategy for a product line [McGregor 04]. This technique guides the thought processes by providing five standard forces that the strategy must resolve.

**Develop schedules and assign responsibilities** – Each activity in the selected tactics is assigned and the schedule for its execution is recorded. The success of the Toyota Production System (TPS) points to the need for very specific assignments. Each activity should be the responsibility of a specific person with other specific persons assigned in backup and escalation roles.

This is a fairly standard set of steps. How we might make them more efficient? Standardization and reuse. Many of the goals in a development project are routine and occur in every development project and, in fact, several times in the same project. Having a standard plan format leads to a standardized planning process.

## Decisions, Decisions, Decisions

The important aspect of planning that is often overlooked in a deliverable-driven process is that a plan records important decisions about a project or product. Seldom can these decisions be made by one person, and where they can be, the person assigned to write the plan seldom has the knowledge or authority to make them. This is not a matter of empowering engineers, it is a matter of getting agreement among the set of engineers who will be affected by the decision.

Decision making in software intensive system development is usually a cooperative effort involving many complex trade-offs. Any one person will have a single viewpoint and if only that person has input into planning, the plan will only address issues from that perspective. For example, if a process engineer is asked to write the development plan, the process descriptions will be overlong, and perhaps overblown, while important technical constraints may be vaguely referred to, if addressed at all.

Ordinarily I would never advocate have a meeting of any kind. However, a well-run planning session in which people are given materials ahead of time along with a specific goal is often an efficient way to gather input from all the stakeholders. The Product Line Systems Program at the Software Engineering Institute (SEI) has shown that focused sessions such as their Quality Attribute Workshop [Barbacci 03] or Production Planning Workshop can be very effective in gathering the information, establishing the context and making decisions that shape the eventual plan. Each of these provide read-ahead materials, a structured agenda, and managed interactions.

## 3 PLANS

The act of capturing decisions, made during planning, in writing forces the planner to be more precise and concrete in the details. I have witnessed many incidents in which faults in programs or discrepancies in documentation have been traced back to a developer’s vague understanding of a task, which was never put in writing. Agile development methods often counter this risk by having daily meetings in which such inconsistencies might be caught.

Plans often follow a standard format. There are a number of IEEE standards for specific types of plans. Some of them are shown in Table 1. (Notice that IEEE discriminates between planning and plans in software quality assurance.) Gary Chastek and I have provided a standard format for a product line’s production plan [Chastek 02].

Standard forms support automation. The automation can range from simple parameters to a document, such as variables in a Word document, to build scripts that assemble selected pieces based on choices made at variation points in a product line architecture. Automation for plans reduces the effort for constructing the plan considerably.

**Table 1 - IEEE plans**

| |
|---|
| IEEE Std **730-1998**, IEEE Standard for Software Quality Assurance Plans |
| IEEE Std **730.1-1995**, IEEE Guide for Software Quality Assurance Planning |
| IEEE Std **828-1998**, IEEE Standard for Software Configuration Management Plans |
| IEEE Std **1058-1998**, IEEE Standard for Software Project Management Plans |
| IEEE Std **1059-1993**, IEEE Guide for Software Verification and Validation Plans |

Figure 1 shows the standard outline of a production plan as defined by Chastek and McGregor [Chastek 02]. We can build small purpose-built tools that guide the completion of the standard plan. Figure 2 shows the front screen of a wizard that guides the production planner through completing the plan. The up front investment in the tool is amortized over a number of plans that follow the same standard format.

1. Introduction
   Production context
   Audience
   Qualifications
2. Strategic view of product development
   Assumptions
   Qualities
   Products possible from available assets
   Production strategy
3. Overview of available core assets
   Basic inputs and dependencies
   Variations
4. Detailed production process
5. Tailoring production plan to product-specific production plan
   Product production
6. Management information
   Schedule

Production Resources
Bill of materials
Product-specific details
Metrics

Figure 1 - Product Plan outline

## 4 EXAMPLE

Recently I have been working with the SEI on production planning. One of the advantages of a software product line is that reuse goes well beyond reuse of code. By using commonality and variability analysis decisions, constraints, and plans can be reused. Not only is this efficient, it ensures a higher degree of consistency among products and development efforts.

As part of my work on the Pedagogical Product Line for the SEI, I produced a prototype for automating the creation of the production plan. Figure 2 shows the front end of this system. The back end uses XVCL to encode decision making logic [Zhang 04]. This logic selects from among a set of predefined "chunks" that capture a particular decision. While my approach assumes a closed set of decision outcomes, it is possible to allow a more open ended system that allows the planner to enter their own outcome.

XVCL provides a set of tags that support manipulation of text, whether the source is a plan document or a piece of source code, The important feature for our discussion of planning is that XVCL provides logic operators that allow the encoding of decision making logic. Figure 3 shows a simple example of a selection statement. Each "option" is a separate frame to process. The example shown corresponds to the first set of choices shown in the interface shown in Figure 2. The complete example can be seen in [McGregor 05].

The consideration that goes into developing this logic goes beyond the thought necessary to write a plan in English. Incompleteness is easier to spot because logical paths can be traced and the plan generation can be formally tested.

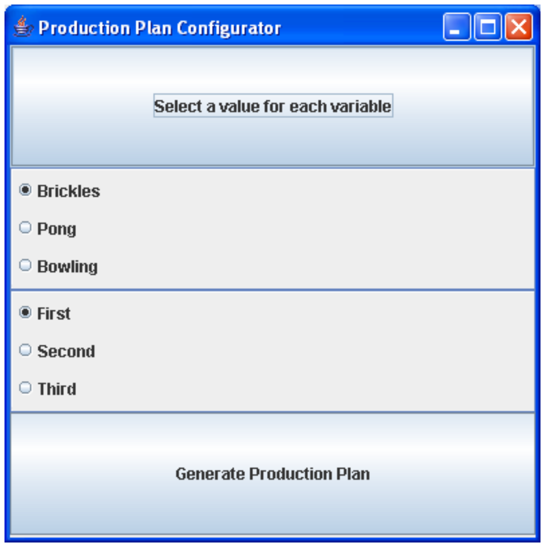
Production Plan Configurator
Select a value for each variable
Brickles
Pong
Bowling
First
Second
Third
Generate Production Plan

Figure 2 - Automated Production Planner

```
<x-frame name="chooseBricklesFeatureDiagramFrag.xvcl">
<select option="GAMENAME">
<option value="Brickles" comp-operator="=">
<adapt x-frame="BricklesFeatureDiagramFrag.xvcl">
<option value="Pong" comp-operator="=">
<adapt x-frame="PongFeatureDiagramFrag.xvcl">
<option value="Bowling" comp-operator="=">
<adapt x-frame="BowlingFeatureDiagramFrag.xvcl">
</adapt>
</option>
</select>
</x-frame>
```

Figure 3 - XVCL fragment

## 5 SUMMARY

As you may guess from reading this article I have had some clients lately who have written plans without planning first. The result is first a sense of accomplishment that a task is finished followed by perplexity when the plan fails. I once wrote a column titled “Let’s Don’t and Say We Did.” Saying we have a plan when it has not been thought through and vetted with the appropriate stakeholders may satisfy some deliverable in a contract or process but it seldom results in good plans or good products. Well conceived, well executed plans can make a strategic difference for your organization. Automating certain aspects of planning provides the same benefits as generating code and other technical assets.

## REFERENCES

[Barbacci 03] Mario Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood. Quality Attribute Workshops, Third Edition, CMU/SEI-2003-TR-016.

[Boehm 03] Barry Boehm and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley, 2003.

[Chastek 02] Gary Chastek and John D. McGregor. Guidelines for Developing a Product Line Production Plan, CMU/SEI-2002-TR-006, 2002.

[McGregor 04] John D. Mcgregor. Factors in Engineering Strategically Significant Software Development Methods, Second Workshop on Method Engineering for Object-oriented and Component-based Development, October 2004.

[McGregor 05] John D. McGregor. Preparing for Automatic Derivation of Products in a Software Product Line, Software Engineering Institute, CMU/SEI-2005-TR-017, 2005.

[Scwaber 02] Ken Schwaber and Mike Beedle. Agile Software Development with Scrum, Prentice Hall, 2002.

[Zhang 04] Hongyu Zhang and Stan Jarzabek. XVCL: a mechanism for handling variants in software product lines. Science of Computer Programming, 53(3), 381 – 407, 2004.

## About the author

**Dr. John D. McGregor** is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.