Planning Game

Planning is an emotional minefield. Of course Development would like to program faster. Of course the project manager would like to be able to say exactly how fast Development can go. Of course Business would like to be able to say exactly what they want. Of course Business would rather not change its mind. When any of the participants in planning begin acting these wishes (or rather in accordance with the fears that lie behind each wish), then planning doesn't work well.

The Planning Game: Create a little emotional distance from planning by treating it as a game (hence the name). The game has a goal, playing pieces, players, and rules for allowable moves.

Pieces: The basic playing piece is the UserStory. Each Story is written on an IndexCard. Stories have a value and a cost, although this is a little tricky because the value of some Stories depends on the presence or absence of other Stories (see StoryDependenciesInXp), and the values and costs change over time.

Goal: The goal of the game is to put the greatest possible value of stories into production over the life of the game.

Players: The players are Business and Development.

Moves:

Write Story: Business can write a new Story at any time. For purpose of the Planning Game, writing a Story just means assigning it a value (in practice, it has to have enough information for Development to assign it a cost).

Estimate Story: Development takes every story and assigns it a cost of 1, 2, or 3 weeks of IdealProgrammingTime (c.f. ExtremeTimeSpans) If the estimate is higher, Business splits the story. (This may result in the story being implemented over more than one Iteration.) If the estimate is lower, Business merges it with another story. (Sometimes we just batch small stories willy-nilly until they add up to at least a week, for estimation purposes. Don't try that at home.) We use a LoadFactor (see ProjectVelocity) of 3 real weeks per ideal week to convert the final schedule to real time.

Make Commitment: Business and Development work together to decide what stories constitute the next release and when it will be ready to put into production. There are two ways to drive the commitment, Story Driven and Date Driven.

Story Driven Commitment: Business starts putting the Stories for the next release on the table. As each Story is introduced, Development calculates and announces the release date. This move stops when Business is satisfied that the Stories on the table make sense as the next release.

Date Driven Commitment: Business picks a release date. Development calculates and announces the cumulative cost of Stories they can accomplish between now and the date. Business picks Stories whose cost adds up to that number.

Value and Risk First: Development orders the Stories in a commitment so:

```
    0 A fully working but sketchy system is completed immediately
(like in a couple of weeks)
    0 More valuable Stories are moved earlier in the schedule
(BusinessValueFirst)
    0 Riskier Stories are moved earlier in the schedule
(WorstThingsFirst)
```

Overcommitment Recovery: Development had predicted they could do 150 units of stories between now and the deadline. Based on measuring ProjectVelocity, they find and immediately announce that they can only do 100. Business selects the 100 units of Stories to retain, deferring the other Stories to a future release. (Or highly unlikely: Business decides to defer the deadline to get the extra 50 units done.)

Change Value: Business changes the value of a Story. In response, Development may change the order of Stories not yet completed.

Introduce New Story: Business writes a new Story. Development estimates it. Business defers Stories in the current Commitment whose cumulative cost is the cost of the new Story. Development re-evaluates Value and Risk First.

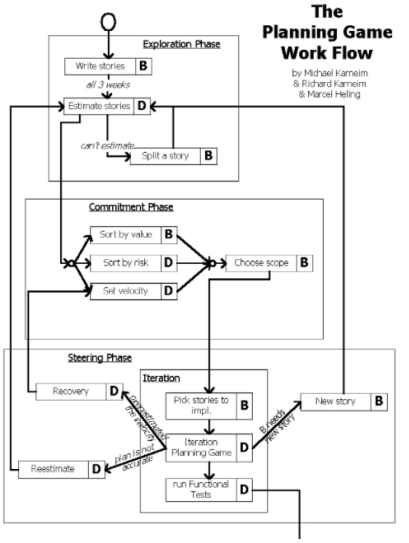
Split Story: Business splits a Story into two or more. Business assigns a value to each, and Development assigns a cost to each. Typically this is done because resources do not permit the whole story to be done soon enough.

Spike: Business can divert Project resources to do a throwaway Spike to fight a fire or prove a concept. If this Spike is anything more than a temporary fix, Business makes a UserStory to account for it. That Story is scheduled according to Value And Risk First. Regular spikes, especially fire-fighting ones, will affect the LoadFactor.

Re-estimate: Development estimates the remaining stories in the Commitment again. This can spark an OvercommitmentRecovery.

A PlanningGame flow chart:

(Web site gone. The diagram is captured at www.archive.org [see below].)



B = Business
D = Development

How do you reconcile the PlanningGame with the project that is re-writing an existing system, or systems? The incremental deployment (and incremental abandonment of the old system) that seems to be an integral part of XP can be difficult when existing data needs to be converted from the old system and integrated with existing data in the new system. -- SteveSawyer

In which case, the conversion itself and the provision of interfaces onto the (new|old) data that behaves exactly as if it was the (old|new) data (before|after) the replacement of each code segment should be set as stories themselves. -- MattTrout

I think you must mean *tasks* and not *stories*. The customer writing the stories is unlikely to understand all the technical aspects of the old system and data migration. This understanding is difficult to acquire when a poorly maintained and documented system is to be replaced. In these cases incremental abandonment and deployment is easier to do at the organizational level, rather than the software level. (see also ExtremeDeployment) -- ChrisSteinbachsddsd

The old system might have a turn-off date; up until that date keep on re-writing functionality, deploying it with conversions from the old data, and make sure that the new code is used. For a CRM solution, for example, a good starting place might be some user security plus a search facility to find a contact's unique ID, which can then be used to look the contact up in the old system. From there go to where the money is - often high value and risk in one functional area. It gets messier when the data has to flow both ways, but should be ok so long as the customer expectations are managed and the users cooperate - they should so long as they can see a solid, beneficial system (and preferably a pretty GUI with good HumanComputerInteraction).

Should the PlanningGame ever have an entire iteration set aside for refactoring? See RefactoringIteration.

PlanningGame...

...is considered incompatible with the fixed-price contract. See CostingExtremeProgramming OptionalScopeContracts. *Yes, but see the **Succeeding with "Agile Fixed Price" projects** papers http://www.nayima.be/writing/papers.html*

...was first called ExtremeScheduleNegotiation. Its name may change again in the future, as discussed at PlanningGameNameChange.

[was *How does this work in small teams our projects have to be ready in what a couple of months for full CRM systems ?*, tried translating it]

How does this work in small teams, or projects which have to be ready in a couple of months, or full CRM systems ?

How does this work in small teams, where projects for full CRM systems have to be ready in no more than a couple of months ?

Obviously the planning game is a balance between time and functionality, but personally I've found that development cost and development time aren't always proportional. Specifically, if you use a third-party product in your software to save time on a Date Driven Commitment, you generally just pass the price on to the customer. But shouldn't that influence Business's decisions, to a greater extent than just getting the most value (Story-Driven) or getting the most value by a deadline (Date-Driven)? How does the PlanningGame accommodate for Development coming back and saying "well, we can do stories A, B, and C by the deadline, but it'll cost you X for this third-party product"? A Change Value by Business on the story, or something?

Cross-reference with CostingExtremeProgramming.

-- JosephRiesen

In 2007, JoelSpolsky came up with a PlanningGame knockoff called EvidenceBasedScheduling, which features more noodling about with numbers in spreadsheets (which is largely automated), but which might deliver more accurate estimates than the orthodox Planning Game.

See StoryDependenciesInXp, PlanningGameEstimationUnits, GamesVsPatterns, PainlessSoftwareSchedules, XpPlanningGame, ToyStorePlanningGameAnalogy, ProjectPlanningSoftware.

CategoryPlanning

Last edit August 29, 2013, See github about remodeling.