Death By Planning

- **AntiPattern Name:** Death by Planning
- **Also Known As:** Glass Case Plan, Detailitis Plan
- **Most Frequent Scale:** Enterprise
- **Refactored Solution Name:** Rational Planning
- **Refactored Solution Type:** Process
- **Root Causes:** Avarice, Ignorance, Haste
- **Unbalanced Forces:** Management of Complexity
- **Anecdotal Evidence:** "We can't get started until we have a complete program plan." "The plan is the only thing that will ensure our success." "As long as we follow the plan and don't diverge from it, we will be successful." "We have a plan; we just need to follow it!“

Background

In many organizational cultures, detailed planning is an assumed activity for any project. This assumption is appropriate for manufacturing activities and many other types of projects, but not necessarily for many software projects, which contain many unknowns and chaotic activities by their very nature. Death by Planning occurs when detailed plans for software projects are taken too seriously.

General Form

Many projects fail from *over* planning. Over planning often occurs as a result of cost tracking and staff utilization monitoring. The two types of over planning are known as the Glass Case Plan and Detailitis Plan. The Glass Case Plan is a subset of the Detailitis Plan in that (over) planning ceases once the project starts. In the Detailitis Plan, over planning continues until the project ceases to exist, for a variety of unfulfilling reasons.

Glass Case Plan

Often, a plan produced at the start of a project is always referenced as if it's an accurate, current view of the project even if it's never updated. This practice gives management a "comfortable view" of delivery before the project starts. However, when the plan is never tracked against, nor updated, it becomes increasingly inaccurate as the project progresses.

This false view is often compounded by the absence of concrete information on progress, which often is known only after a critical deliverable slips its schedule.

Sometimes, when a project plan created prior to project launch, management assumes that the plan guarantees delivery automatically—exactly as specified with no intervention (or management) necessary.

Detailitis Plan

Sometimes the solution to effective delivery is regarded as a high degree of control via a continuous planning exercise that involves most of the senior developers, as well as the managers.

This approach often evolves into a hierarchical sequence of plans, which show additional (and unecessary) levels of detail. The ability to define such a high level of detail gives the perception that the project is fully under control.

Symptoms And Consequences

The Glass Case Plan symptoms are the first noticed in both it and the Detailitis Plan.

Glass Case Plan

The symptoms usually include at least one of the following:

- Inability to plan at a pragmatic level.
- Focus on costs rather than delivery.
- Enough greed to commit to any detail as long as the project is funded.

The consequences are incremental:

- Ignorance of the status of the project's development. The plan has no meaning, and control of delivery lessens as time goes on. The project may be well ahead or behind the intended deliverable state and no one would know.
- Failure to deliver a critical deliverable (the final consequence).

Consequences grow incrementally until finally the project overruns, with the usual options of:

- Further investment.
- Crisis project management.

- Cancellation.
- Possible loss of key staff.

Detailitis Plan

The symptoms are a superset of the Glass Case Plan:

- Inability to plan at a pragmatic level.
- Focus on costs rather than delivery.
- Spending more time planning, detailing progress, and replanning than on delivering software:
    - Project manager plans the project's activities.
    - Team leaders plan the team's activities and the developers' activities.
- Project developers break down their activities into tasks.

The consequences are intertwined and feed off each other:

- Each planner has to monitor and capture progress at the level shown by his or her plan, and reestimate.
- Endless planning and replanning causes further planning and replanning.
- The objective shifts from delivery of software to delivery of a set of plans. Management assumes that because effort and cost are tracked, progress must be equivalent. In fact, there is no direct correlation.
- Continual delays to software delivery and eventual project failure.

Typical Causes

In both cases, Glass Case Plan and Detailitis Plan the primary cause is lack of a pragmatic, common-sense approach to planning, schedules, and capture of progress.

Glass Case Plan

- No up-to-date project plan that shows the software component deliverables and their dates.
- Ignorance of basic project-management principles.

- Overzealous initial planning to attempt to enforce absolute control of development.
- A sales aid for contract acquisition.

Detailitis Plan

- Overzealous continual planning to attempt to enforce absolute control of development.
- Planning as the primary project activity.
- Forced customer compliance.
- Forced executive management compliance.

Known Exceptions

There should never be any exceptions to the Death by Planning AntiPattern.

Refactored Solution

The solution is the same for both the Glass Case and Detailitis Plans. A project plan should show primarily deliverables (regardless of how many teams are working on the project). Deliverables should be identified at two levels:

1. *Product(s)*. Those artifacts sold to a customer, which include the internal corporate lines of business that use them.
2. *Components (within products)*. Basic technology artifacts required to support a business service.

Deliverables include such things as:

- Business requirements statement
- Technical description
- Measurable acceptance criteria
- Product usage scenarios
- Component use cases

The plan should be supplemented with validation milestones for each component, as well as the overall product, such as:

- Conceptual design approval
- Specification design approval
- Implementation design approval
- Test plan approval

The deliverable plans should be updated weekly to ensure appropriate planning and controls that reduce project risks. This allows issues, risks, slippages, and early deliveries of defined deliverables to be dealt with by appropriate, timely responses.

Tracking is done on the estimated level of completeness. This sometimes means regressing the completeness that was input to the plan in a previous time period. Completeness should be gross measurements rather than fine measurements, for example, tracking in 25 percent steps rather than 5 percent steps.

A Gantt chart can be used effectively to visually illustrate deliverables, associated dates, and interdependencies. By tracking against a baseline plan, the following states of deliverables will be immediately obvious:

- On schedule.
- Delivered.
- Early (with estimated new delivery date).
- Late (with estimated new delivery date).

It is essential to baseline early and rarely. Otherwise the ability to track changes is lost. Furthermore activities/tasks/deliverables should have dependencies set between them where they exist.

When estimating it is advisable to allow a contingency period for all those inevitable "unknowns," such as:

- Requirements increase (creep).
- Design dead ends.
- Third-party software workarounds.
- Defect identification (finding the problem in a set of integrated components).
- Defect correction (bug fixing).

It is also important to establish a minimum time frame in which to accomplish any activity. This prevents such constraints as two days to code and test a "simple" program.

Variations

The Death by Planning variations are in levels of detail, and can go from identifying major milestones, which are usually tied to funding/approval stages, to micro-deliverables within the project delivery stages for each team.

These variations are equally applicable to both the Glass Case and Detailitis Plans:

- Funding variations
- Micro-deliverables variation

The Glass Case version of the micro-deliverables plan only varies from the Detailitis Plan in that it is never updated. It shows very minor deliverables that cannot be fully understood prior to starting the project. Therefore, any estimates must by definition be incorrect based upon the lack of any real understanding of the software to be built. This type of plan is usually produced by technically gifted amateurs. Although the tasks need to be clearly understood, putting them in a plan results only in unnecessary planning and tracking (in the case of the Detailitis Plan), as opposed to doing.

Example

The examples are based on our own experiences in learning "the hard way."

Glass Case Plan

For this example, we'll say a systems integrator decides to build a middleware component not yet available from any of the major vendors, in spite of international standards issued over a year ago. The systems integrator agrees to a detailed delivery plan in advance of starting any project work in order to obtain a funding source. The plan is based upon estimates from staff who have not yet delivered software "in anger."

The plan is highly specific technically, and the estimates are very optimistic; and it is continually referenced by the project manager, but never updated to reflect any actual effort.

This leads to missed delivery dates. There is a general lack of knowledge as to the real progress of the project; the systems integrator is apprised of delivery date failure only *after* the dates have passed. The plan shown in figure below is an extract from such a development project.

Detailitis Plan

In an attempt to control development to ensure that full control is established, an end-user company produces a plan that has three levels:

1. phases of development.
2. team tasks and effort.
3. team member tasks and effort.

The plan in figure below indicates the complexity.

Detailitis causes an inability to track against the plan without taking considerable focus away from delivering the system. This results in significantly reduced staff productivity. Plan management quickly becomes unrealistic due to complexity.

The solution is to replace the detailed plan with one that shows key deliverables against dates, with dependencies and constraints. If the deliverable is missed, then something important has occurred. The previous plan attempted to track progress by effort that was unrealistic: Effort = staff x days elapsed. So what? E = MC2, but the formula does not create nuclear power!

Related Solutions

The Analysis Paralysis AntiPattern can exacerbate the consequences of Death by Planning. If the analysis phase runs over its allotted schedule, either a schedule must slip or inadequate analysis models are applied to downstream phases.

Applicability To Other Viewpoints And Scales

Death by Planning cannot match the chaotic (white-water) nature of software development, because it creates a significant disparity between the management's planning models and actual development activities.

Architects and developers often have to live double lives: On one hand, they must visibly cooperate with management's plan for the project; at the same time, they have to confront the actual status of the software, which may not resemble the management model. For example, the plan may be used to pressure developers into declaring software modules complete before they are mature. This causes downstream problems in integration and testing.