The Price of Uncertainty: A Mathematical Analysis of Classic and Agile Processes

Kevin Thompson, Ph.D., PMP, CSP
Agile Practice Lead
cPrime, Inc.
4100 E. Third Avenue, Suite 205
Foster City, CA 94404

Kevin.thompson@cprime.com

Abstract
Advocates of agile development claim that agile software projects succeed more often than classic plan-driven projects. Unfortunately, attempts to validate this claim statistically are problematic, because "success" is not defined consistently across studies. This paper addresses the question through a mathematical analysis of these projects. We model agile and plan-driven software projects with identical requirements, and show how they are affected by the same set of unanticipated problems. We find that that the agile project provides clear benefits for return-on-investment and risk reduction, compared to the plan-driven project, when uncertainty is high. When uncertainty is low, plan-driven projects are more cost-effective. Finally, we provide criteria for choosing effective process types.

Contents

1 Background

The discipline of project management focuses on the processes, tools, and techniques used to organize the efforts of a group of people to produce desired results. The PMBOK® Guide, from the Project Management Institute (PMI), emphasizes this point in its definition:

> ***"Project Management is the application of knowledge, skills, tools and techniques to project activities to meet project requirements."*** [1]

Up to the present, the PMI perspective on managing projects has focused primarily on classic *plan-driven* strategies.[2] Plan-driven strategies assume that the work of a project can be planned in advance of its execution, and that execution can be made to follow the plan reasonably well. Plan-driven projects emphasize gathering requirements (scope), defining the project's work items, dependencies, sequence, and estimated effort (the Work Breakdown Structure), and executing the work to create the desired results.

In the field of software development, plan-driven projects frequently use the *Waterfall* model,[3] which divides the schedule into explicit phases, as in Figure 1:

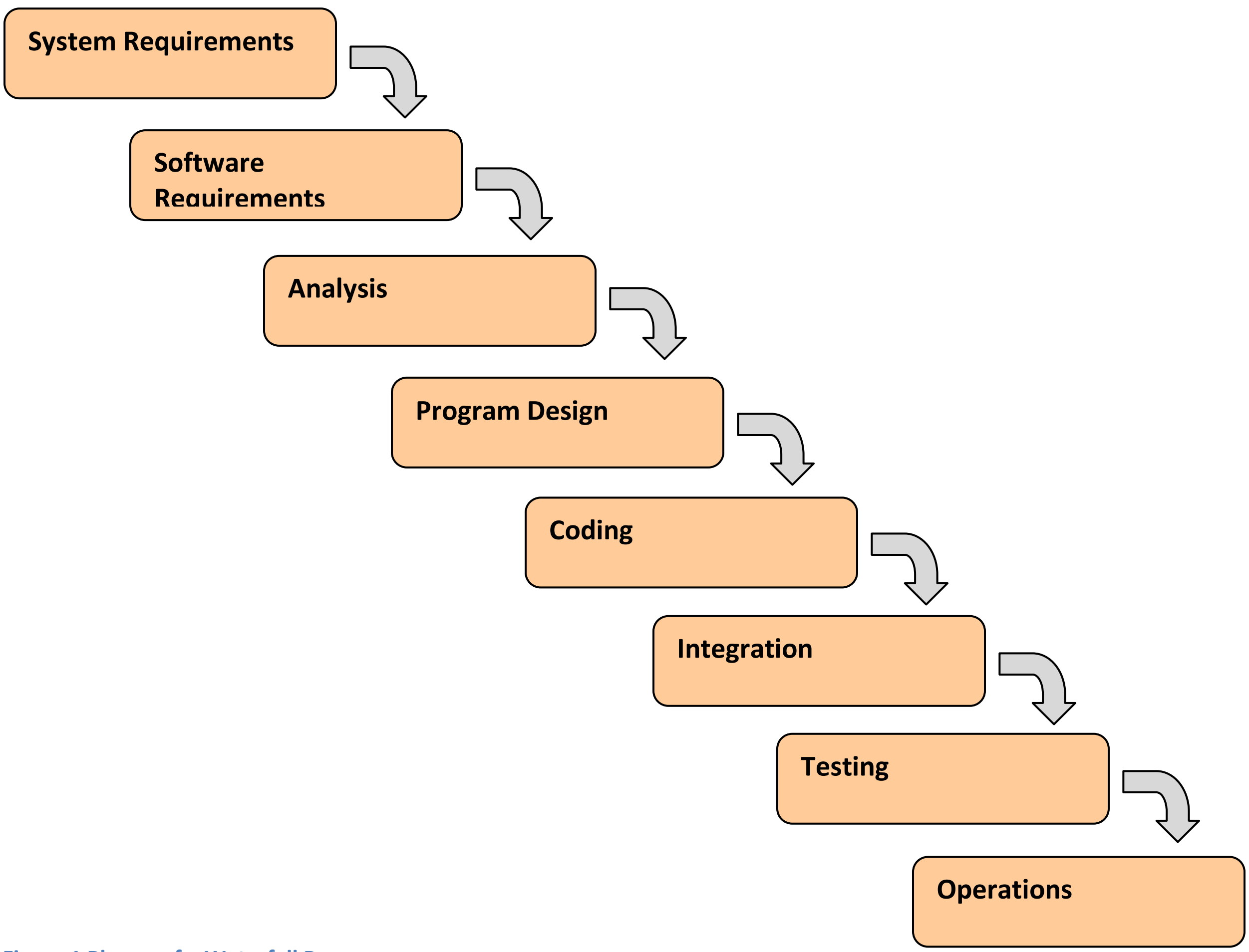


Figure 1 Phases of a Waterfall Process

Plan-driven strategies have been effective for projects in many industries, but attempts to apply these strategies to the world of computer software development, using the Waterfall model, have been less successful. For example, the Standish Group's 2009 survey of 50,000 software projects revealed that 32% of the projects succeeded, 44% of projects qualified as "challenged" (late, over budget, lacking functionality, or having low quality), and 24% failed.[4]

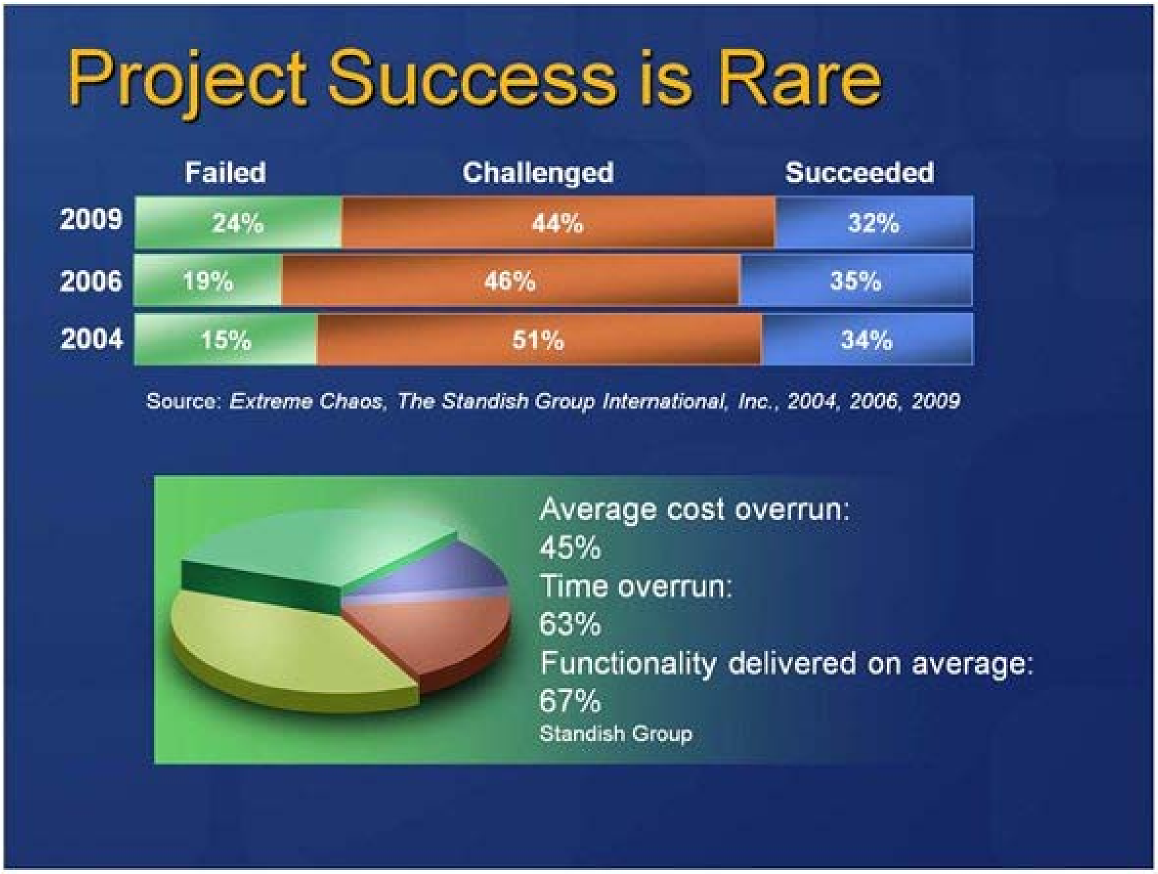


Figure 2 Standish Chaos Report: 2004 – 2009. Copyright © 2010 by The Standish Group International, Inc.

The difficulty of managing software projects has long been noted, and many alternative strategies have been proposed. A major contribution to this effort occurred in 2001 with the publication of the Agile Manifesto[5], which emphasized collaboration, results, and adaptability over process, documentation, and adherence to plans. While none of these concepts was innately incompatible with plan-driven strategies, taken together they represented a significant shift in perspective regarding what matters more in the management of successful software projects, and what matters less.

A number of agile project-management frameworks have arisen, including Scrum, XP (Extreme Programming)[6], Commitment-Based Project Management (CBPM)[7], Kanban[8,9], and others. In many cases, these strategies predate the Agile Manifesto (Scrum, for example, debuted in 1995), but have been grouped under the umbrella of agile development because of a common emphasis on the agile principles listed in the Manifesto.

That agile projects can succeed is not in dispute. Less clear are the answers to these two questions:

1. Do agile strategies work better than plan-driven strategies, for software projects?
2. If so, how do we characterize projects for which one strategy is more effective?

This paper will attempt to answer these questions.

2 Common Problems in Software Projects

Before trying to answer the above questions, it is worth reviewing common problems in software projects. The author's personal experience, and numerous reports, consistently shows the following elements:

1. Creating detailed specifications is time-consuming, and requires substantial effort.
2. Creating the project schedule is challenging because the work breakdown structure is complex, and the associated estimates require substantial effort.
3. Although work seems to progress smoothly at first, getting an accurate picture of project status is difficult, as it is often not clear whether work items are truly complete.
4. Various unexpected things occur throughout the project:
    a. Requirements are not accurate and detailed enough for implementation. Some have been omitted, and some are incorrect. Resolving the discrepancies takes time.
    b. Design artifacts that appear at first to be complete are not, and require an unpredictable number of revision cycles to finalize.
    c. Development work proceeds more slowly than expected. Design issues have to be re-visited, code reviews uncover problems that need to be fixed, and estimates are often unreliable even in the absence of these issues.
    d. The development teams are distracted by the need to fix critical production problems.
    e. Because the project will take many months to complete, customers who "missed the boat" during the requirements phase submit change requests to get their high-priority features implemented. The resulting scope changes increase project duration.
    f. Integration of new components is more difficult, and takes longer, than expected.
    g. Testing turns up more defects than can be fixed in the allotted time.
    h. Defect repairs create unexpected regression errors that require fixing.
    i. The deployment process contains new elements introduced by this project. Initial deployment efforts fail, and deployment takes more time than expected.
5. The project finishes late, over budget, and with higher defect rates or less functionality than intended.
6. Customers discover that the new release doesn't provide what they really wanted.

The key point is that every aspect of the project is subject to substantial uncertainty. The cumulative effects of uncertainty extend the project duration well beyond the planned period, and waste development funds on the wrong features.

3 Statistics on Success Rates for Plan-Driven and Agile Projects

Our first attempt to answer the questions posed in this paper is to review the published statistics on success rates for plan-driven and agile projects. The 2009 Standish Report strongly suggests that plan-driven software projects have not been very successful, but provides no information about how agile software projects compare. Two studies that address the performance of agile versus plan-driven projects are described below.

3.1 Scott Ambler, 2007

Scott Ambler's 2007 survey[10] in *Dr. Dobb's Journal* provides information from 586 respondents. The results showed success rates of 63% for Waterfall projects, and 72% for agile projects. Unfortunately, the study did not attempt to define a meaning for "success," so the results reflect the respondents' (unknown) definitions.

3.2 QSM Associates, 2008

In 2008, Rally Software commissioned QSM Associates (QSMA) to assess the performance of agile versus plan-driven projects.[11] QSMA compared 29 agile development projects against a database of 7500 primarily Waterfall projects. The study shows that the agile projects were 37% faster in delivering software to market, and 16% more productive, while maintaining satisfactory defect levels.

3.3 Conclusions from the Surveys

The survey results are suggestive, but have some limitations:

- Sample sizes for agile projects are much smaller than for Waterfall projects
- The three studies (Standish, Dr. Dobb's Journal, QSM Associates) measure and report on different characteristics of the projects, rather than on a uniform definition of success

The Ambler and QSMA surveys both indicate better results for agile projects, but the absence of common metrics for success means that the three reports are not directly comparable.

In fact, it may not be possible to define success consistently across plan-driven and agile projects, given the radically different ways these projects operate. "Planned scope, on time" makes sense for a plan-driven project, but is not directly applicable to agile projects that freeze schedule and adjust scope. As a result, comparisons of success rates for the two styles of project may not be meaningful.

Yet we would still like some guidance as to whether (and why) plan-driven or agile processes are more effective for software projects, so we will develop a different approach below.

4 Key Differences between Agile and Plan-Driven Strategies

One challenge to comparing the effectiveness of agile and plan-driven strategies for software project management is the absence of a unique definition for either. However, the table below captures common distinctions observed in many projects that fit into these categories, with Agile Process characteristics drawn primarily from Scrum.

Plan-Driven Process	**Agile Process**
Predictive	Adaptive
Fixed scope	Fixed schedule
Adjusts schedule to preserve scope	Adjustable scope to preserve schedule
Long development cycle (e.g., 6 months)	Short development cycle (e.g., 2—4 weeks)
Linear	Cyclic
Organizes work into major phases	Organizes work into small deliverables
Delivers value at project completion	Delivers value incrementally over time

The differences are extensive, and at first glance, it isn't clear which of the agile characteristics (if any) would lead to improved performance for software projects. It is also possible that no one of the characteristics dominates, and that a synergistic combination is responsible for benefits.

On further inspection, though, one can see that most of the characteristics in the table relate to the time dimension, which suggests an avenue for exploration.

5 Gedanken Experiment

Gedanken Experiment, or "thought experiment" in the German language, was a favorite term of Albert Einstein's. A Gedanken Experiment is an experiment performed in the mind, instead of the physical world, through which we explore the implications of our assumptions or theories. As the available statistics have not provided clear guidance about the relative merits of plan-driven and agile processes for software projects, we will design a Gedanken Experiment to address these merits in quantitative terms.

Section 2 suggests that the greatest challenge to successful software projects is uncertainty, which impacts the project in many ways. Thus our Gedanken Experiment will assess the impact of uncertainty on two projects that are designed to produce the same deliverables, using identical teams. One project will follow a plan-driven process, while the other will follow an agile process.

To make the comparison as simple as possible, we will focus on only one of the distinguishing characteristics of the two processes—the length of the development cycle—and ignore the others. We will then subject both projects to the same set of unexpected problems, and see how the impact of uncertainty differs between them.

5.1 Project Description

The goal of each project is to create a data warehouse and reporting system, with a set of five reports, and to deploy these capabilities in a new data center. The information in these reports is provided by existing business applications. Customers will use the new report capabilities to improve budgeting, and are willing to pay for the reports because the information may produce significant cost savings.

Analysis of expected costs and revenues indicate that the project should be cost-effective if we can begin to receive revenues after investing a maximum of one year's funding. Thus each project has funding for one year, and extension is contingent on showing revenues by the end of the funded year.

The system architecture contains the following components:

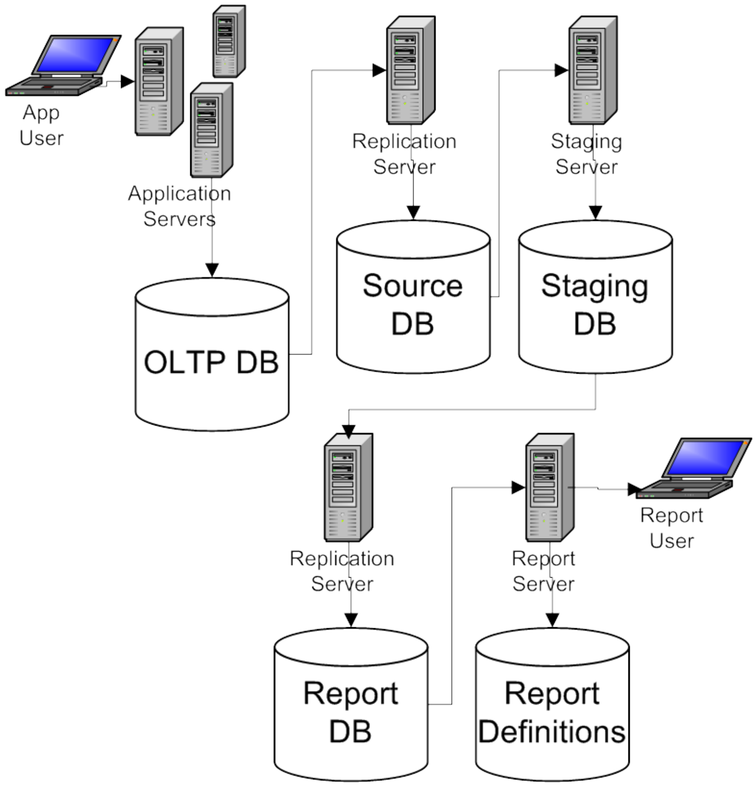


Figure 3 Data Warehouse and Reporting System

1. An OLTP (Online Transaction-Processing) database, which stores the business-application data
2. A replicated source database, which contains a copy of a subset of the OLTP data
3. A Staging database, populated by ETL processes that pull data from the replicated database, and transform the data into a table structure optimized for report generation
4. A Report database, which is a copy of the Staging database, which is used only by the reporting application
5. A reporting application, which runs on a dedicated report server, and which displays reports on request, using data from the Report database

The major pieces of work to be done are as follows:

1. Build the production environment
2. Configure production servers with database and reporting software
3. Create database tables in the various environments, for the various stages of the data pipeline
4. Develop ETL (Extract-Transform-Load) processes to transfer data between databases in the pipeline, and ultimately to the reporting database
5. Develop the reports

5.2 Uncertainty

The two projects are subjected to the same uncertainties, which manifest in the following ways:

- Work estimates are low, and all planned work takes 25% more time than expected.

- Report #2 depends on a table in the OLTP source database that has 80 million records. The special processing required to deal with this table adds 3 weeks to the schedule.
- The report-software vendor releases a major upgrade ten months into the project. This upgrade is required to fix critical bugs, and adds 3 weeks to the schedule.
- Several source tables required for Report #3 contain duplicate data. Handling this problem adds 3 weeks to the schedule.
- Production deployment problems add 3 weeks to the schedule.

5.3 The Plan-Driven Project

The plan-driven project contains nine major phases, as shown in Figure 4.

Task Name	Duration	Start	Finish
⊟ Develop DWH and BI System	191 days	Mon 1/3/11	Mon 9/26/11
⊞ Write specs for report capabilities	12 days	Mon 1/3/11	Tue 1/18/11
⊞ Analyze source schema	25 days	Wed 1/19/11	Tue 2/22/11
⊞ Design ETL process	30 days	Wed 2/23/11	Tue 4/5/11
⊞ Implement ETL process	48 days	Wed 4/6/11	Fri 6/10/11
⊞ Develop BI metadata layer	8 days	Mon 6/13/11	Wed 6/22/11
⊞ Create reports	15 days	Thu 6/23/11	Wed 7/13/11
⊞ Test reports	15 days	Thu 7/14/11	Wed 8/3/11
⊞ Fix bugs in Reports	12 days	Thu 8/4/11	Fri 8/19/11
⊞ Deploy for users	26 days	Mon 8/22/11	Mon 9/26/11
Project Completed	0 days	Mon 9/26/11	Mon 9/26/11

Figure 4 Microsoft Project schedule for plan-driven project

The elapsed time estimated for the project is slightly less than nine months, which leaves about three months of buffer time in the funded period.

5.4 The Agile Project

The agile project does not contain phases in the same sense as the plan-driven project. Instead, it is structured to deliver functionality in increments, one report at a time, as shown in Figure 5.

Task Name	Duration	Start	Finish
⊟ Develop DWH and BI System	235 days	Mon 1/3/11	Fri 11/25/11
⊞ Develop Report #1	63 days	Mon 1/3/11	Wed 3/30/11
⊞ Develop Report #2	43 days	Thu 3/31/11	Mon 5/30/11
⊞ Develop Report #3	43 days	Tue 5/31/11	Thu 7/28/11
⊞ Develop Report #4	43 days	Fri 7/29/11	Tue 9/27/11
⊞ Develop Report #5	43 days	Wed 9/28/11	Fri 11/25/11

Figure 5 Microsoft Project schedule for agile project

At the conclusion of each major task, a new report is available for use in the production environment. This means that each major task contains testing and deployment overhead, including regression testing to ensure that the creation of each new report has not broken any of the previous reports. Thus the project as a whole contains more time allocated to these types of work than does the plan-driven project, which performs such work once for all reports.

We account for the additional overhead of the agile project by adding more time for requirements, testing, and deployment work, relative to the plan-driven project. The time increment (at 23%) is somewhat arbitrary, and may be an overestimate, but reducing it to, say, 10% does not materially affect the conclusions. (See Table 2 in the Appendix for details.)

The estimated schedule shows that Report #1 requires more time than the others, because the work includes setting up the production environment. The schedule as a whole requires approximately eleven months, which leaves one month of buffer time in the funded period.

6 Comparison

In this section, we will compare the two projects, in terms of their expected and actual results.

6.1 Comparison of Planned Project Schedules

The plan-driven schedule is the clear winner, with completion estimated in two months less time than the agile schedule. Not only do the plans show the plan-driven project completing earlier, but it has more buffer time in the funded year.

6.2 Comparison of Actual Project Schedules

The addition of 25% more work time than estimated expands the plan-driven schedule to 11 months, while the four 3-week delays expands the schedule further to 14 months.

The effects of uncertainty on the agile project also change its delivery timeline, as shown in Table 1.

Report	Expected Time to Delivery (months)	Actual Time to Delivery (months)	Problem Areas
1	3	4.5	Deployment
2	5	7.8	Huge table
3	7	11.0	Duplicate Data
4	9	14.3	Report-software upgrade
5	11	16.8	

Table 1 Effect of uncertainty on agile project schedule

Three of the reports are delivered within the funded year, and the total schedule expands to 17 months.

6.3 Comparison of Project Results

The plan-driven project would be canceled, as it does not complete or generate any revenues by the end of the funded year. This means that the entire investment in the project would be lost. However, the agile project would deliver three working reports in that same year, and thus generate revenues. This means that the project produces some return on the investment, and qualifies as a candidate for continuation.

7 Lessons Learned from the Gedanken Experiment

Several points of interest may be observed regarding uncertainty, risk, and return on investment (ROI).

7.1 The Financial Impact of Uncertainty

We see that agile projects are no more immune to delays from unexpected problems than are plan-driven projects. However, the way uncertainty impacts the projects has implications for ROI.

When uncertainty is low, ROI is a calculation: A project will take X months to complete, cost $Y, and yield $Z of revenue over the following twelve months.

When uncertainty is high, ROI is a gamble:

- The project might not finish
  - It may be late, and not complete before the company goes out of business
- Funding might disappear
  - Business priorities can change. The company may get out of the tire business, and start selling rear-view mirrors.
- Customer interests can change
  - Last year's best-selling Pet Rock is this year's gravel

In high-uncertainty environments, risk is more about losing the entire investment than it is in having less ROI than expected.

7.2 Risk

Risk scales with scope and project duration. Thus the plan-driven project's all-or-nothing approach to delivery risks the complete loss of investment, while the agile project's focus on delivering incremental value as early as possible reduces risk by maximizing the probability of getting some ROI.

7.3 Value Delivery and ROI versus Time

The agile project's incremental delivery of value not only reduces risk, but provides superior ROI compared to the plan-driven project.

It is generally understood that receiving $100 per month for ten months is better than receiving $1000 after ten months. This is because

1. Having cash now is more useful than not having it
2. The ability to supply money might stop after five months, and receiving some is better than receiving none
3. The time value of money favors increments over time (i.e., we might invest each increment and earn more interest than we'd get from the single large payment).

The same logic applies to agile projects that deliver functionality in useful increments over time, rather than deferring delivery of all functionality to the end of a large project.

8 Guidance for Selecting Processes

All else being equal, the incremental value delivery of agile projects reduces risk and improves ROI in high-uncertainty contexts, relative to monolithic, plan-driven projects. Both types of project will be impacted by uncertainty, but agile projects are more robust, in the sense that they can deliver value even when affected by a degree of uncertainty that can cause plan-driven projects to fail.

In practice, “all else” is seldom equal, and the incremental deliveries of agile projects may add significant overhead relative to a plan-driven project. The overhead is worth paying for high-uncertainty projects, because the agile project delivers value more reliably. However, if uncertainty is low, plan-driven projects provide greater predictability and efficiency, and thus lower cost.

This line of thinking suggests that we select processes that best fit the characteristics of projects, and the following sections provide guidance for how to do so.

8.1 Common Processes

We first consider four processes that either define or are common representatives of their categories. One of these (the plan-driven process) is classic, while the others are commonly considered to be agile processes.

Plan Driven

This classic category is described in Section 1, and includes Waterfall and SDLC processes.

Scrum

This agile process plans and implements work in short iterations, called Sprints. Each Sprint produces a set of completed and tested deliverables. Scrum is popular for software development and IT-project management.

CBPM

Commitment-Based Project Management is an agile process whose characteristics are driven by the hardware-design world in which it was born. CBPM assumes that work is continuous, because it cannot readily be divided into short iterations. Like Scrum, it presumes that work contains enough uncertainty to render detailed plans obsolete in short order. Thus CBPM incorporates frequent re-planning exercises every few weeks, in order to provide short-term plans of useful reliability.

Kanban

Kanban is an agile process that does no planning. Instead, it is a pull-oriented process that focuses on reacting effectively to unpredictable requests, by re-prioritizing them on a daily basis, and setting work-in-process limits to minimize the cost of uncompleted work.

8.2 Selection Criteria

We next consider the following four criteria:

Planning Criticality: The two values are *Needed*, if planning is needed, and *Not Needed,* if it is not.

Requirements Reliability: The two values are Low and High. *Low* means that requirements are not well understood, may contain significant errors, and may be unstable (i.e., change substantially over the duration of the project schedule). *High* means that requirements are accurate, complete, and stable over time.

Estimation Reliability: The two values are Low and High. *Low* means that estimates are not reliable (factor-of-two errors are common, and larger errors are not uncommon). *High* means that estimates are reliable (say, usually within 20% of actuals).

Cyclic Granularity: The two values are Yes and No. *Yes* means that a specific set of requirements can be completed and validated in short (2—4 week) development cycles. *No* means that the work cannot be divided into regular cycles.

8.3 Decision Matrix for Process Types

The table shows which type of process is best suited for a project, based on the project characteristics.[1]

Planning Criticality	Requirements Reliability	Estimation Reliability	Cyclic Granularity	Process
Not Needed	Any	Any	Any	Kanban
Needed	Low	Low	No	CBPM
Needed	Low	High	No	CBPM
Needed	Low	Low	Yes	Scrum
Needed	Low	High	Yes	Scrum
Needed	High	Low	No	CBPM
Needed	High	High	No	Plan-driven
Needed	High	Low	Yes	Scrum
Needed	High	High	Yes	Plan-driven

Figure 6 Decision Matrix for Process Types

Perhaps surprisingly, the greatest divide is not between plan-driven and agile processes, but between the agile Kanban process and everything else. The reason for this divide is that Kanban dispenses with planning entirely, while the others assume planning is necessary, but differ on how it is conducted.

Plan-driven processes work well when both requirements and estimation are reliable, which is commonly the case for repetitive work. When requirements, estimation, or both are unreliable, Scrum and CBPM are most appropriate, the choice driven by whether the work can be divided into iterations.

The following flowchart illustrates a common decision process for selecting process types.

[1] *Any* means that the line is valid for any possible value of the parameter.

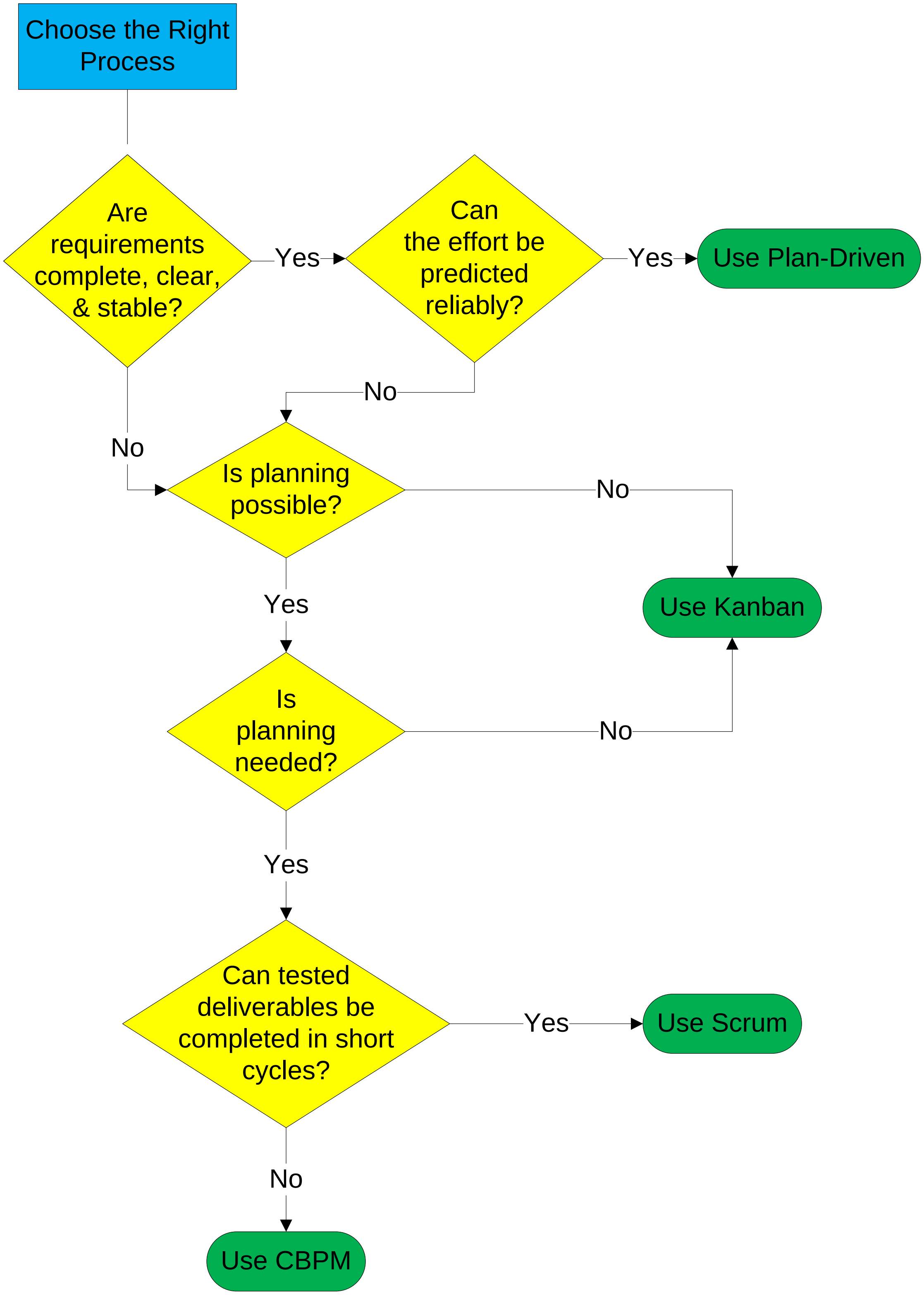
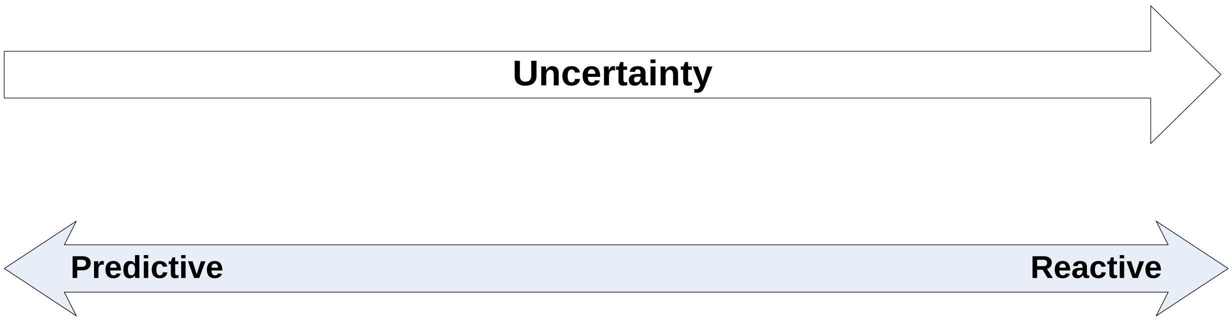


Figure 7 Flowchart for Process Selection

Finally, the Adaptive Spectrum diagram below provides a convenient summary of the relationship between process types and appropriate projects, based on the total uncertainty (scope + effort estimates) inherent in the work of the project.

Adaptive Spectrum



Predictive Planning	Adaptive Planning	Reactive Planning
Plan-Driven	Scrum	Kanban
Waterfall	XP	
SDLC	CBPM	

Figure 8 The Adaptive Spectrum of process types

9 Conclusion

A mathematical analysis of the effects of uncertainty on agile and plan-driven projects shows that an agile process is more likely to deliver value when uncertainty is high, than is a plan-driven process. However, since the short iterations of an agile process impose more overhead, relative to a plan-driven process, the latter is likely to cost less and complete more quickly for projects where uncertainty is low. The concluding recommendation is to understand each project's characteristics, including its level of uncertainty, and choose a process that is well-suited to its characteristics.

10 Appendix: Task Durations for the Plan-Driven and Agile Schedules

Categories and Tasks		Duration (Days)	
		Plan-Driven	Agile
Write specs for report capabilities			
	Interview representative users	5	10
	Write specifications	3	5
	Get feedback from users	2	5
	Revise specifications	2	5
Analyze source schema			
	Identify relevant tables from OLTP system	15	15
	Document table relationships	10	10
Design ETL process			
	Develop schema for report DB	10	10
	Design ETL logic to populate report DB	20	20
Implement ETL process			
	Replicate subset of OLTP DB to local copy	5	5
	Implement ETL for local copy to staging	20	20
	Implement ETL for staging to warehouse	20	20
	Replicate warehouse to report DB	3	5
Develop BI metadata layer			
	Design metadata layer	5	5
	Implement metadata layer	3	5
Create reports			
	Write Report #1	3	3
	Write Report #2	3	3
	Write Report #3	3	3
	Write Report #4	3	3
	Write Report #5	3	3
Test reports			
	Test Report #1	3	3
	Test Report #2	3	3
	Test Report #3	3	3
	Test Report #4	3	3
	Test Report #5	3	3
Fix bugs in Reports			
	Fix all bugs	3	5
	Retest	3	5

	Fix any new bugs	2	5
	Retest	2	5
	Fix remaining bugs	1	2.5
	Verify quality is acceptable	1	2.5
Deploy for users			
	Set up servers	5	5
	Install software	5	5
	Create databases	5	5
	Build ETL process	5	7
	Perform "smoke test"	2	6
	Fix problems	3	7
	Go live	1	5
	Total Duration	**191**	**235**

Table 2 Task Durations for the Plan-Driven and Agile schedules

Trademarks

PMBOK® is a registered trademark of the Project Management Institute.

---

[1] *A Guide to the Project Management Body of Knowledge: PMBOK® Guide*, Third Edition. Project Management Institute, Inc. 2004.

[2] The PMBOK does not use the term "plan-driven," but the latter has become a common term for any process that assumes predictability over time scales longer than a few weeks.

[3] "Managing the Development of Large Software Systems," by Winston W. Royce. *Proceedings of the IEEE WESCON*, August, 1970. Royce did not coin the term "Waterfall," or recommend the process to which this term was later applied, but did document the process.

[4] *CHAOS Summary 2009,* The Standish Group International. (http://www.standishgroup.com/newsroom/chaos_2009.php). 2009

[5] *Manifesto for Agile Software Development*. www.agilemanifesto.org, 2001.

[6] *Extreme Programming Explained: Embrace Change* (2nd Edition), by Kent Beck and Cynthia Andres. Addison-Wesley Professional. 2004.

[7] *No Surprises Project Management: A Proven Early Warning System for Staying on Track*, by Timm Esque. Ensemble Management Consulting. December 1, 1999. See also the white paper by Jose Solera at http://www.pmlead.com/.

[8] *Kanban*, by David J. Anderson. Blue Hole Press. 2010.

[9] *Kanban and Scrum—Making the Most of Both*, by Hendrik Kniberg and Mattias Skarin. C4Media. 2010.

[10] *Defining Success*, by Scott W. Ambler. *Dr. Dobb's Journal*. (http://www.drdobbs.com/architecture-and-design/202800777). Oct 31, 2007.

[11] *The Agile Impact Report: Proven Performance Metrics from the Agile Enterprise*. Rally Software Development Corp. 2008.