Agile Methods: Fast-Paced, but How Fast?

Michael L. Harris  
*Indiana University - Southeast, harris60@ius.edu*

Rosann Webb Collins  
*University of South Florida, rcollins@coba.usf.edu*

Alan R. Hevner  
*University of South Florida, ahevner@coba.usf.edu*

Follow this and additional works at: [http://aisel.aisnet.org/amcis2009](http://aisel.aisnet.org/amcis2009)
Agile Methods: Fast-Paced, but How Fast?

Michael L. Harris
School of Business
Indiana University - Southeast
harris60@ius.edu

Rosann Webb Collins
Information Systems and Decisions Sciences
College of Business
University of South Florida
rcollins@coba.usf.edu

Alan R. Hevner
Information Systems and Decisions Sciences
College of Business
University of South Florida
ahevner@coba.usf.edu

Proceedings of the Fifteenth Americas Conference on Information Systems
San Francisco, California August 6th-9th 2009
Agile Methods: Fast-Paced, but How Fast?

ABSTRACT
What are the roles of time and time pressures in design and performance of agile processes for software development? How do we plan our rapid development activities given the constraints of due dates? What does it mean to be on ‘internet time’? Agile methods are meant to be fast-paced, but are they fast in an effective way? How do time-pressures influence the productivity of a project team and how do they impact the motivations of developers? This paper considers the time issues in agile approaches to managing software projects and posits research propositions to guide further study of this area.

Keywords
Software Product Development; Time Pressures; Adaptable Software Development.

THE COMPLEX ROLE OF TIME PRESSURES

Agile methods are associated with rapid development techniques, with short iterations leading to frequent software releases. Fast-paced development with frequent deadlines leverages the deadline effect: as deadlines near, employee motivation and work accelerates, leading to higher productivity. But when an entire project needs to be delivered on an aggressive schedule a plan-driven approach, similar to the waterfall development method, often may be quicker than an agile approach. It is important to understand under what circumstances agile methods adapt effectively to the demands of time constraints and when they may be counterproductive.

The guidelines of method engineering suggest that the software development method chosen to manage a project should vary based on the characteristics of a project (Fitzgerald, et al. 2006). As we focus our attention on the dimension of time, we consider the ways that time pressure can influence the choice of software processes and development methods. This paper contains a conceptual analysis of time illustrated by examples extracted from a qualitative study of two software development organizations and the literature on the management of software projects that employ agile methods.

The importance of time pressure in software development project was identified in many of the interviews that we performed in the two organizations. A typical example:

Interviewer: So, could you have somehow waited to start development until they knew more of what the details were?
Response: No, … it was going to go live … as mandated by [the government] we had to have our product ready to collect data …

Pressure to meet a market entry date with a minimum set of acceptable features may override other development priorities and influence the selection of the ‘best’ development process. The following quote illustrates the selection of development method under extreme time pressure.

“… [management says] oh by the way, we need this in 35 days. … we have methods to try and do that kind of procedurally, but if, at the extreme what you start doing is really just, is balancing risk with quality then and cutting corners. … there are certain [times] where you have to break out of the box and kind of just get something done”

Development tradeoffs with time pressure are as complex and the effect of time pressures on the choice of development processes and methods is far from simple. If market uncertainty is low and system requirements are well-defined, a plan-driven approach may be the quickest way to complete a project. With extreme time pressure, it may be impossible to complete a plan, but an agile approach may not be faster. An agile approach may involve a search process and feedback cycles that can result in rework, and thus can slow delivery. If there is not time for a plan-driven approach, and not enough time for the rework of an agile approach, the team must accept increased delivery risk.

In this paper, we consider the role of time constraints in agile software development. We explain how, and why, agile methods may be considered to operate under time pressure, but we show that the fast-paced development that is used by agile methods is not necessarily equivalent to a short-time to completion. We provide a conceptual analysis of the nature of time constraints and their impact on process selection. The paper concludes with a set of research propositions for future research investigations on the role of time in the selection and performance of agile processes and methods.
THE NATURE OF DEADLINES

Deadlines or due dates on a project may be driven from two different sources. Developers work from the bottom-up based on the requirements and resources to determine the time required to complete a project. Management establishes goals based on their priorities and on exogenous factors. The deadline is negotiated between developers and managers based on the information that each brings to the discussion.

If perfect information is available, it would appear that a due date would be a single, objective date. This date would be the time required to complete the project. In the real world, there are many sources of uncertainty that lead to imperfect information. Requirements are often fuzzy and may emerge throughout the process. Design is still an art, and two different software architects may develop different designs to solve the same problem. As a result of varying requirements and potentially different design decisions there can be variability in the scope of the project. Execution of the plan can also vary since developers are human, not machines. Since there are multiple sources of potential variability, it can be difficult to prove that a given estimate is an unbiased estimate of the project duration. This can lead to second guessing of time estimates.

Management may see due dates as project controls that can be manipulated to achieve management objectives. One view is based on Parkinson’s Law (Parkinson, 1993) that available work will expand to fill the available time. If management is guided by this viewpoint they may feel a need to shorten estimates in order to eliminate unnecessary effort. Management may also use due dates to signal the priority of a project through its urgency.

Deadlines also serve to motivate work, through the deadline effect: as deadlines near, employee motivation and work accelerates, leading to higher productivity. This phenomenon is observed in many settings, and employees themselves report that deadlines are among the key factors that lead to higher productivity (White and Locke, 1981).

Project scope can also be signaled through the establishment of a due date. Consider a project team charged with delivering a new accounting system. Ideally, the project may require 52 weeks of development and management may have the leeway to negotiate that estimate to 51 or 50 weeks. However, if management has a firm requirement to deliver the project in three months the only way to accomplish the goal is to completely change the scope of the project.

When management manipulates due dates to control or signal team members they are manipulating subjective time (He and Souren, 2008). Subjective or perceived time pressure (Austin, 2001) is used to create a deadline effect and it may be independent of any objective due date for the project. These subjective due dates can be perceived as useful (as noted earlier, through the deadline effect), but they may also create unnecessary pressure for the developers. For example in one interview, a developer stated:

“A long time ago when I was having migraines and all that stuff, doctor took .. well, he said … just take it easy. And, that’s what I do. So, timelines … I think they’re important for managers … So, timeline just puts unnecessary pressure.”

Individual developers may also have their own concerns when calculating delivery dates. For example, the same developer quoted above stated:

“Other people like to have the code done in the shortest amount of time and I’m like, well that’s not really important if it’s not maintainable, cause then I’m going to be spending my time afterwards to maintain this thing. So, I’d rather spend some more time in making it maintainable.”

As a result of legitimate differences and varying agendas there can be considerable controversy in the negotiation of a project due date. This may lead to game playing in estimates. Parties may arbitrarily pad or cut estimates in expectation of future negotiations.

In addition to a negotiated, subjective due date there may be an objective due date. This objective due date is given by exogenous factors. For example, a Y2K fix must be delivered by midnight on the last day of 1999. When these exogenous factors are sufficiently distant compared to the scope of the project they may not change the negotiated due date. However, an emergency may occur when a due date is shorter than the required time to complete the project.
INTERPLAY BETWEEN PROJECT DUE DATES AND METHODS

In order to understand the effects that due dates and time pressures have on methods we begin with some definitions.

We start by considering the requirements of a project. There is a difference between the full set of requirements and the minimum needed to meet a given due date. This was illustrated by the comment from the following developer:

“I would say about 20% of the projects are not done the way a developer would have liked it done because of time constraints. But we have what we have done in those cases we have said O.K.; here is what we want to do what is going to be absolutely needed to meet the client’s requirement. This is what we will do right now and this portion that we don’t do at the moment gets put in detail into the [tracking] system to be able to be implemented at a later date.”

This leads us to the following distinction in type of requirements:

- **Minimum Requirements**: Recognizing that it may be possible to deliver the project in phases, these are the minimum features required in the first phase. This is assumed to be a subset of the full requirements.

- **Full Requirements**: The minimum requirements plus other capabilities that are justifiable.

Next we consider the time that it will take to deliver these features. As these definitions state, these times assume varying levels of ‘ideal’ performance, and are unlikely to be achieved in reality.

- **Theoretical Minimum Completion Time (Theo-Min)**: Assume that a perfect team exists (i.e. it never makes mistakes) and that it has perfect knowledge. This is the time that the team would take to complete and deliver a product that meets the minimum requirements with no planning time (perfect knowledge), no testing time (perfect team), and instantaneous rollout. Unless the project is trivial it is assumed that no real team can complete the project this fast.

- **Exogeneous Due Date (Exo-Due)**: The fixed due date that is derived from exogeneous factors. For example, the date of midnight on the last day of 1999 was the exogeneous due date for Y2K fixes. Some sources of exogeneous due dates include regulatory requirements, competitive activities, and customer contract requirements. At times there may not be any exogeneous due date and the due date will be solely based on management and team priorities.

TYPICAL WATERFALL SOFTWARE DEVELOPMENT LIFE CYCLE

Now let’s consider how the due dates are set for a team that uses a typical waterfall-style SDLC. First, we assume that the project is appropriate for a waterfall approach. For example, we assume that there is not a high level of uncertainty (Harris et. al., 2009). We assume that this team does not have perfect knowledge, and that it needs to get its requirements from stakeholders. Furthermore, although the team is very good, it is not perfect; therefore it needs to follow ordinary best practices in terms of quality control, documentation, and rollout transitions.

- **Ideal Minimum Completion Time (Ideal-Min)**: Stakeholders with perfect knowledge are available and the technology is well-understood. As a result, the team can develop a clear plan upfront and deliver without requiring adjustments or changes to the plan. The team perfectly executes a best practices process. This perfect execution means that a team follows all required steps, but that they get everything right the first time.

- **Real Minimum Completion Time (Real-Min)**: This is the same as the ideal minimum completion time, but it assumes a ‘normal’ amount of friction in the process. This friction appears as miscommunications, oversights, and mistakes. It results in some rework and in late changes to the software.

Given the previous definitions, consider the timeline in figure 1 below:
Examine figure 1, and assume that the Exogenous Due Date is in the range marked (3). This would imply that we can realistically expect to complete the project (Real-Min) before the project needs to be completed (Exo-Due). The project team can use a typical waterfall approach and achieve acceptable results as long as the project is of low uncertainty (Harris et. al., 2009). Although the Exogenous Due Date does not confine the project team, management may choose to set a more aggressive goal in order to exercise control on the team.

Now consider the circumstance where the Exogenous Due Date is in the range indicated by (2) in figure 1. If everything works perfectly the team can complete the project in ideal time and finish on time. The closer the exogenous due date approaches the ideal minimum it may want to consider adjustments to allow the team to meet the due date. Staffing the team with the organization’s best developers can reduce the project time towards the due date. Assigning a customer contact full-time to the team may reduce the impact of mis-communication in the plan. Pair programming may increase the likelihood that team members will get the solution right the first time. Of course, all of these adjustments carry an additional cost to the base project cost.

As the Exogenous Date approaches the ideal minimum, and even passes it (1), the due-date becomes infeasible if the best practices approach is used. In order to reduce the work time below the required time, shortcuts may be required (Austin, 2001). For example, the team might not conduct thorough planning or may reduce testing. One software manager described his approach in these circumstances.

“It’s like, oh by the way, we need this in 35 days. So … at the extreme what you start doing is really just .. is balancing risk with quality then and cutting corners.”

Taking shortcuts may offer the chance of on time delivery, but as the magnitude of shortcuts increase the probability of achieving the on time delivery will decrease. At the limit, development without any controls and without any planning carries a high risk of non-delivery.

Finally, we consider range (0) where the exogenous date is at or below the theoretical minimum. It will not be possible for the team to meet its due date given the planned approach. One option is to consider alternative solutions that change the playing field. For example, adopting an existing service bureau approach rather than building a custom solution. However, short of creative options, the team will be unable to meet the required deadline.

**AGILE METHODS**

Agile methods have a reputation for being fast-paced. The first principle in the Agile Manifesto is that “Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software” (http://agilemanifesto.org). As an example, consider eXtreme Programming (Beck & Andres, 2004). Every morning, team members report on the previous day’s progress. Daily software builds insure continuous attention to details. At any given time, the next release is no more than a week or two away. Even when agile projects are large, practitioners believe that it is critical to keep iterations of releases to weeks rather than months (Reifer et al., 2002). All of these techniques make work visible. If team members slack...
off for a day it will be very visible to their peers. These techniques insure there is no let-up in the development pace. A continuous series of short, achievable deadlines keep employees motivated and should result in higher productivity. However, as noted earlier, this could also lead to higher employee stress.

Such positive (increased productivity) and negative (higher employee stress) within-project effects of the accelerated pace of agile methods are different from the issue of faster overall time-to-completion of a software project. In a survey of organizations using agile methods, Reifer (2002, p. 16) found that a prime motivation for adopting agile methods was time: the organizations’ “poor record of delivering acceptable products on time and within budget.”

However, we note that there is a distinction between a fast pace and a fast completion. As an illustration, consider a race from A to D in the diagram below. A fast-paced runner who travels from A to B to C to D (the agile approach) might be defeated by a walker who travels from A directly to D (the plan-driven approach when it is possible to know where you are going).

![Diagram](image_url)

*Figure 2: The Shortest Path from A to D is a Straight Line*

This is not to imply that agile approaches are inefficient. They are quite efficient in conditions of uncertainty (Harris et. al., 2009). The agile development process resembles a search process. Iterations deliver estimated feature subsets, and generate feedback for future iterations. Although this may work as a search process, a more direct route may be possible if the destination is known a priori.

These differences can be further illustrated if we compare the agile processes to a waterfall approach. The agile approaches save project time to the extent that they do not develop a complete plan before development begins. Even when agile projects are large, practitioners recommend that the up-front planning should primarily focus on the development of a strong, guiding software architecture rather than detailed system specifications (Reifer et al., 2002). However, this savings in up-front planning is offset by rework and by abandoned work that occurs when feedback is received after an iteration is completed.

The agile approach will also work well if the development environment is highly uncertain in either the requirements or technology (Harris et. al., 2009). In this latter case, the agile search process is appropriate given the uncertainty that exists. Likewise, the upfront plan of a waterfall process is likely to be wasted because the uncertainty will make it difficult to develop a priori plans. However, when uncertainty is low, planners will be able to develop an a priori plan before development begins and this will eliminate rework and eliminate time spent on abandoned features. In the low-uncertainty setting, therefore, a more plan-driven approach should result in a faster time-to-completion.

**THE LEAST TIME APPROACH**

The least time approach will therefore depend on the conditions. If uncertainty is high, an agile approach may deliver the quickest feasible solution. However, the agile search process can still take substantial time to complete and although this is the quickest feasible approach it is not necessarily a quick approach. Moreover, the continuous short deadlines could lead to increased employee stress, which could negatively impact productivity. The delivery time can be shortened if we can replace the search process with a thorough upfront plan, such as the plan in the waterfall process. However, this requires a low uncertainty project so that the upfront plan is effective. If the required due date (exogenous due date) is sooner than the plan-driven approach can deliver, the project team may be able to hit the due date by taking shortcuts. However, each shortcut taken increases the risk of non-delivery or poor quality. As the required due date becomes close to the theoretical minimum the number of shortcuts required will increase the risk beyond reasonable levels. At this extreme, the team will need to either find new delivery options or they will be unable to meet the required date.
RESEARCH PROPOSITIONS

As we can see from this discussion, the time-related conditions that guide the choice of development approach are complex, and research is needed to untangle and understand the various relationships. Based on the conceptualization of the time issues in the use of agile methods presented in this paper, we posit the following research propositions:

1. When agile methods are used that employ development iterations with continuous, short deadlines, employee motivation within iterations will be higher than for development using a single, longer-term deadline.

2. When agile methods are used that employ development iterations with continuous, short deadlines, employee stress within iterations will be higher than for development using a single, longer-term deadline.

3. In higher-uncertainty software development projects, an agile approach will result in faster overall time-to-completion than a plan-driven approach.

4. In lower-uncertainty software development projects, a plan-driven approach should result in faster overall time-to-completion than an agile approach. However, an alternative proposition is that even when there is lower uncertainty, the increased motivation from continuous, short deadlines in the agile approach may also result in faster overall time-to-completion.

FUTURE RESEARCH DIRECTIONS

In our on-going research, we have studied the selection and adaptation of agile processes and methods under conditions of market uncertainty (Harris et. al., 2009). However, we also found that other project priorities may apply in other conditions, such as a requirement for extreme time pressure. This suggests a portfolio of agile control choices and that selection of specific agile techniques would depend on the management priorities of the project at hand. In one case the priority might be market uncertainty, in another case time pressure, in a third case system reliability, and so on. In this regard, we call for further research into the conditions that shape the choice of methods used. We also recognize that conditions may not be clearly separable. For example, a project may require extreme reliability and may have hard time constraints to meet development milestones. Therefore, future research needs to address tradeoffs in these complex situations. The propositions proposed in the previous section provide a starting point for our future research directions in the study of time in agile process and method design and performance.

REFERENCES