Balancing Plan-Driven and Agile Methods in Software Engineering Project Courses

Barry Boehm, Dan Port, and A. Winsor Brown
Department of Computer Science, Center for Software Engineering, University of Southern California, Los Angeles, CA, USA

ABSTRACT

For the past 6 years, we have been teaching a two-semester software engineering project course. The students organize into 5-person teams and develop largely web-based electronic services projects for real USC campus clients. We have been using and evolving a method called Model-Based (System) Architecting and Software Engineering (MBASE) for use in both the course and in industrial applications. The MBASE Guidelines include a lot of documents. We teach risk-driven documentation: if it is risky to document something, and not risky to leave it out (e.g., GUI screen placements), leave it out. Even so, students tend to associate more documentation with higher grades, although our grading eventually discourages this.

We are always on the lookout for ways to have students learn best practices without having to produce excessive documentation. Thus, we were very interested in analyzing the various emerging agile methods. We found that agile methods and milestone plan-driven methods are part of a “how much planning is enough?” spectrum. Both agile and plan-driven methods have home grounds of project characteristics where they clearly work best, and where the other will have difficulties. Hybrid agile/plan-driven approaches are feasible, and necessary for projects having a mix of agile and plan-driven home ground characteristics.

Information technology trends are going more toward the agile methods’ home ground characteristics of emergent requirements and rapid change, although there is a concurrent increase in concern with dependability. As a result, we are currently experimenting with risk-driven combinations of MBASE and agile methods, such as integrating requirements, test plans, peer reviews, and pair programming into “agile quality management.”

EDUCATIONAL BACKGROUND

For the past 6 years, we have been teaching a two-semester software engineering project course. The students organize into 5-person teams and develop...
largely web-based electronic services projects for real USC campus clients. The applications have included multimedia archives (cinema, medicine, fine arts, rare manuscripts); electronic resource access (business data, journals, dissertations); management and collaborative service applications; and web mail and other information infrastructure services. About 15–20 teams develop a Life Cycle Architecture package (operational concept, prototype(s), requirements, architecture, plans, and feasibility rationale) in the fall semester. The most-desired 6–10 applications are developed and transitioned to clients as Initial Operational Capability packages in the spring semester.

We have been using and evolving a method called Model-Based (System) Architecting and Software Engineering (MBASE) for use in both the course and in industrial applications. It is based on the spiral model, with extensions for stakeholder value proposition elicitation and reconciliation, and concurrent–engineering phases and phase gates developed in collaboration with Rational and its Unified Process (RUP). It is supported by a mix of industry and USC tools, such as Rational Rose, SoDA, and ClearCase; Microsoft Project; GroupSystems.com/USC’s Easy Win-Win; USC COCOMO II; and a USC MBASE Electronic Process Guide. To meet the inflexible 12-week schedules for the fall and spring semesters, we have developed a specialized MBASE variant called the Schedule as Independent Variable process. It has the clients prioritize their features, and the teams architect their applications to be able to add or drop borderline features to meet the schedule constraints. This approach has resulted in a 92% success rate in delivering on-time, client-satisfactory applications (Boehm et al., 1998; Boehm & Port, 2001; Port & Boehm, 2002).

The MBASE Guidelines include a lot of documents. Students learn to develop an Operational Concept Description, Requirements Definition, Architecture Description, Life Cycle Plan, and Feasibility Rationale in the fall. In the spring, they learn to develop a Quality Management Plan, Iteration Plans and Reports, Peer Review Plan and Report, Test Plan and Report, Release Description, User’s Manual, Transition Plan, and Support Plan. We teach risk-driven documentation: if it’s risky to document something, and not risky to leave it out (e.g., GUI screen placements), leave it out. Even so, students tend to associate more documentation with higher grades, although our grading eventually discourages this.

We are always on the lookout for ways to have students learn best practices without having to produce excessive documentation. Thus, we were very interested in analyzing the various emerging agile methods: Adaptive Software Development, Crystal Methods, Dynamic Systems Development Method,
Extreme Programming, Feature-Driven Development, Lean Development, Scrum (Highsmith, 2002). Our main objectives were to see how agile methods and MBASE could be combined to produce better approaches for both large complex systems and smaller applications.

**ASSESSMENT OF AGILE AND PLAN-DRIVEN METHODS**

In summary, our assessment of agile and plan-driven methods indicated that:

- Both agile and plan-driven approaches have a responsible center and over-interpreting radical fringes (by “plan,” we are including documented process plans: tasks and milestones; and product plans: designs, architectures, and requirements).
- Each approach has a “home ground” of project characteristics within which it performs very well, and much better than the other.
- Outside each approach’s home ground, combined approaches are feasible and preferable.
- Good combined approaches can be determined by using risk analysis to determine how much planning and agility is enough for your situation.
- Marketplace trends are making agility increasingly important.

**THE PLANNING SPECTRUM**

Figure 1 places various agile and plan-driven development approaches along a spectrum of increasing emphasis on plans. Compared to unplanned and undisciplined hacking, the agile methods do a fair amount of planning. Their

---

**Fig. 1.** The planning spectrum.
general view, though, is that the planning process is more valuable than the resulting documentation (Highsmith, 2002), so they often appear less planned than they are. It is true that hard-core hackers can interpret the agile principles to say that they are agile. But in general the agile methods have criteria, such as XP’s 12 practices, which help people determine whether someone is using an agile method or not. Another encouraging trend is that the buzz of agile methods such as XP is drawing many young programmers away from the cowboy role model toward the more responsible agile methods. The price that XP pays for this, though, is a reluctance of more conservative managers to sanction something called “extreme” (Copeland, 2001).

Plan-driven methods also have a responsible center focused on major milestones and their overall content, rather than on micro-milestones (inch-pebbles) locked into an ironbound contract. Excessively prespecified plans overconstrain the development team even at minor levels of change in personnel, technology, or COTS upgrades; and are a source of major contention, rework, and delay at high levels of change. On the other hand, we found on reviewing many projects using the risk-driven spiral model, that projects (particularly larger ones) frequently lost their way without at least three major anchor point milestones to serve as project progress indicators and stakeholder commitment points (Boehm & Port, 2001).

Figure 1 also shows the location of the current CMU-SEI Software Capability Maturity Model (CMM) and its successor, the CMM-Integrated (CMMI; Ahern, Clouse, & Turner, 2001). The Software CMM has often been used in an overly restrictive and bureaucratic way, but can be interpreted to include some forms of milestone risk-driven models. The CMMI, which adds process areas for Risk Management, Integrated Teaming, and Organizational Environment for Integration, can be interpreted to include the agile methods as well.

THE “HOME GROUNDS” OF AGILE AND PLAN-DRIVEN DEVELOPMENT

Table 1 is our attempt to characterize the “home grounds” (where they perform best) of agile and plan-driven methods, in terms of developers, customers, form of knowledge, requirements, architecture, refactoring, size, and primary objective. Each of these discriminators is discussed further in (Boehm, 2002).
Particularly in the e-services sector, companies with a large installed base of customers do not need just one or the other of rapid value or high assurance. They need both. But pure agility or pure plan-driven discipline cannot do both by themselves. A mix of each is needed.

Martin Fowler in “Is Design Dead?” (Fowler, 2001) and Jim Highsmith in Adaptive Software Development (Highsmith, 2000) provide ways in which agile methods can be combined with plan-driven methods. Another approach that can help you balance agility and discipline is risk management, which is good for answering, “how much is enough?” questions (Boehm & Hansen, 2001). Here we will apply it to address the question, “How much planning is enough?”

A central concept in risk management is the risk exposure (RE) involved in a given course of action. It is determined by accessing the probability of loss P(L) involved in a course of action and the corresponding size of loss S(L), and computing the risk exposure as the expected loss: \( \text{RE} = P(L) \times S(L) \). “Loss” can include profits, reputation, quality of life, or other value-related attributes.

Figure 2 shows risk exposure profiles for an example e-services company with a sizable installed base and desire for high assurance; a rapidly changing marketplace and desire for agility and rapid value; and an internationally

<table>
<thead>
<tr>
<th>Agile home ground</th>
<th>Plan-driven home ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile, knowledgeable, collocated, collaborative developers</td>
<td>Plan-oriented developers; mix of skills</td>
</tr>
<tr>
<td>Above plus representative, empowered customers</td>
<td>Mix of customer capability levels</td>
</tr>
<tr>
<td>Reliance on tacit interpersonal knowledge</td>
<td>Reliance on explicit documented knowledge</td>
</tr>
<tr>
<td>Largely emergent requirements, rapid change</td>
<td>Requirements knowable early; largely stable</td>
</tr>
<tr>
<td>Architected for current requirements</td>
<td>Architected for current and foreseeable requirements</td>
</tr>
<tr>
<td>Refactoring inexpensive</td>
<td>Refactoring expensive</td>
</tr>
<tr>
<td>Smaller teams, products</td>
<td>Larger teams, products</td>
</tr>
<tr>
<td>Premium on rapid value</td>
<td>Premium on high-assurance</td>
</tr>
</tbody>
</table>
distributed development team with a mix of skill levels and a need for some level of documented plans.

The black curve in Figure 2 shows the variation in RE due to inadequate plans, as a function of the level of investment the company puts into its projects’ process and product plans. At the left, a minimal investment corresponds to a high probability P(L) that the plans will have loss-causing gaps, ambiguities, and inconsistencies. It also corresponds to a high S(L) that these deficiencies will cause major project oversights, delays, and rework costs. At the right, the more thorough the plans, the less P(L) that plan inadequacies will cause problems, and the smaller the size S(L) of the associated losses.

The red curve in Figure 2 shows the variation in RE due to market share erosion through delays in product introduction. Spending little time in planning will get at least a demo product into the marketplace early, enabling early value capture. Spending too much time in planning will have a high P(L) due both to the planning time spent, and to rapid changes causing delays via plan breakage. It will also cause a high S(L), as the delays will enable others to capture most of the market share.

The blue curve in Figure 2 shows the sum of the risk exposures due to inadequate plans and market share erosion. It shows that very low and very high investments in plans have high overall risk exposures, and that there is a “Sweet Spot” in the middle where overall risk exposure is minimized, indicating “how much planning is enough?” for this company’s operating profile.

With the example company situation as a reference point, we can run comparative risk exposure profiles of companies occupying the home grounds of agile methods and plan-driven methods as summarized in Table 1. For example, Figure 3 shows the comparative RE profile for an e-services company.
with a small installed base and less need for high assurance, a rapidly changing marketplace, and a collocated team of highly capable and collaborative developers and customers. With this profile, as shown in Figure 3, the major change in risk exposure from Figure 2 is that the size of rework loss from minimal plans is much smaller due to the ability of the team to rapidly replan and refactor, and thus the company’s Sweet Spot moves to the left toward agile methods.

Figure 4 shows the corresponding RE profile for a company in the plan-driven home ground, with a more stable product line of larger, more safety-critical systems. Here, the major difference from Figure 2 is a much higher
size of rework loss from minimal plans, and a resulting shift of the company’s Sweet Spot toward higher investments in plans.

**USING HOME-GROUND ATTRIBUTES TO ASSESS RISK EXPOSURE**

In practice, it is hard to quantify estimates of $P(L)$ and $S(L)$. But if you’re trying to balance your use of agile and plan-driven methods, you can use each method’s “home ground” attributes to determine relative risks. So, for example, if your company or project fits an agile home-ground profile except for having a mix of equally-important customers with less-than-ideal development representatives, you can reduce your risk exposure by conducting stakeholder requirements negotiations among the developers and customers, and documenting the results to minimize the risk of future customer misunderstandings.

Or, if your project fits a plan-driven home-ground profile except for highly volatile user-interface requirements, you can use the concept of risk-driven specifications (Boehm & Hansen, 2001) to document those requirements for which it is a high risk not to specify them, and avoid documenting user-interface requirements whose specification would be high-risk. Risk-driven spiral methods and frameworks such as the Rational Unified Process (Royce, 1998), MBASE (Boehm & Port, 2001), and the CMMI (Boehm, 2002) provide guidelines for finding a good balance of discipline and flexibility, although the current CMMI approach needs more explicit support for agility.

**CONCLUSIONS**

Agile methods and milestone plan-driven methods are part of a “how much planning is enough?” spectrum. Despite certain extreme terminology, they are part of the responsible center rather than of the radical fringes. Actually, agile methods are performing a valuable service by drawing erstwhile cowboy programmers toward a more responsible center.

Both agile and plan-driven methods have home grounds of project characteristics where they clearly work best, and where the other will have difficulties.

Hybrid agile/plan-driven approaches are feasible, and necessary for projects having a mix of agile and plan-driven home ground characteristics. Risk
analysis of your projects’ characteristics versus the home-ground characteristics can help determine your projects’ best balance of agility and plan-driven discipline. Risk-driven spiral methods such as RUP, MBASE, and CMMI provide guidelines for finding the best balance.

Information technology trends are going more toward the agile methods’ home ground characteristics of emergent requirements and rapid change, although there is a concurrent increase in concern with dependability. As a result, we are currently experimenting with risk-driven combinations of MBASE and agile methods, such as integrating requirements, test plans, peer reviews, and pair programming into “agile quality management.”

REFERENCES


