2002

Agile Methodologies and Process Discipline

Mark C. Paulk
Carnegie Mellon University
Agile Methodologies and Process Discipline

Mark C. Paulk
Software Engineering Institute

Agile methodologies have been touted as the programming methodologies of choice for the high-speed, volatile world of Internet and Web software development. They have also been criticized as just another disguise for undisciplined hacking. The reality depends on the fidelity to the agile philosophy with which these methodologies are implemented, and the appropriateness of the implementation for the application environment. This article addresses these issues and summarizes and critiques the compatibility of agile methodologies with plan-driven methodologies as described by the Capability Maturity Model® for Software.

Agile Methodologies

Many names have been used for the agile methods, including Internet-speed, lightweight, and lean methodologies. Similarly, plan-driven methodologies have been described as rigorous, disciplined, bureaucratic, heavyweight, and industrial-strength. Some of these descriptors can be considered derogatory, e.g., lightweight or bureaucratic. Agile is the term preferred by the AgileAlliance, a group of software professionals dedicated to promoting the concepts of agile software development. Plan-driven was coined by Barry Boehm [2] to characterize the opposite end of the planning spectrum from agile methodologies.

Any discussion of agile methodologies should begin with the fundamentals of agile as expressed by its proponents. The Manifesto of the AgileAlliance found at <www.agilemanifesto.org> states:

We are uncovering better ways of developing software by doing it, and helping others do it. Through this work we have come to value:

• Individuals and interactions over processes and tools.
• Working software over comprehensive documentation.
• Customer collaboration over contract negotiation.
• Responding to change over following a plan.

That is, while there is value on the items on the right, we value the items on the left more.

The principles behind the agile manifesto are as follows:
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity – the art of maximizing the amount of work not done – is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methodologies are usually targeted toward small- to medium-sized teams building software in the face of vague and/or rapidly changing requirements. Agile teams are expected to be co-located, typically with less than 10 members.

**Process Discipline in the Agile Methods**

Why would we challenge the principles of agile methodologies? Do not most professionals share the objectives espoused in the agile movement? In one sense, the values expressed in the agile manifesto should be captured in any modern software project, even if the implementation may differ radically in other environments. Customer satisfaction, communication, working software, simplicity, and self-reflection may be stated in other terms, but without them, non-trivial projects face almost insurmountable odds against success.

In effective CMM-based improvement, when defining processes, organizations should capture the minimum essential information needed, use good software design principles (such as information hiding and abstraction) in structuring the definitions, and emphasize usefulness and usability [3]. One of the consequences of the Level 1 to Level 2 culture shift is demonstrating the courage of convictions by becoming realistic in estimates, plans, and commitments.

Much of the formalism that characterizes most CMM-based process improvement is an artifact of large projects and/or severe reliability requirements, especially for life-critical systems. The hierarchical structure of the SW-CMM, however, is intended to support a broad range of implementations within the context of the 18 key process areas and 52 goals that compose the requirements for a fully mature software process. It is true, however, that the CMM emphasizes explicitly capturing knowledge via documentation and measurement – a process emphasis. The challenge for many in adopting agile methodologies lies in the (perceived) problem in de-emphasizing processes, documentation, contracts, and planning.

**Individuals Over Process**

Agile methodologies assume the programmers are generalists rather than specialists. Competent generalists are hard to come by, but this is an endemic problem in any technically demanding discipline. Specialist knowledge in a domain can be needed, however, which may lessen the effectiveness of practices such as pair programming.

The foundation of software engineering in the SW-CMM at Level 1 is competent people, sometimes doing heroics, all too frequently working to overcome the system to do professional work. In spite of heroics, the foundation that is assumed in the CMM is competent professionals.

"Software professionals want to take pride in their work, but how can they when managers say, 'I'd rather have it wrong than have it late. We can always fix it later.'"

Without competent professionals, the best software process is ineffective – because the work we do in software projects is human-centric and design-intensive, and the process is what we do.

Software professionals want to take pride in their work, but how can they when managers say, “I'd rather have it wrong than have it late. We can always fix it later.” When program managers acknowledge that making the schedule is the primary consideration in raises and promotions, what is the impact on motivation, quality, and professionalism? These are fundamental management issues, which is why the focus at Level 2 of the SW-CMM is on project management, which empowers competent professionals to do quality work.

It is interesting to note that, although agile methodologies emphasize individuals over process, the set of practices in an agile methodology addresses the same kind of planning and commitment issues as the focus on basic project management at CMM Level 2. An agile methodology that ignored customer collaboration and incremental development would almost certainly fail. Agile practices are synergistic, and the success of agile methodologies depends on the emergent properties of the set of practices as a whole.

**Working Software Over Documentation**

The agile emphasis on tacit rather than explicit knowledge, as externally captured in documentation, can be a high-risk choice in many environments such as government contracting, but it can be an effective choice if rationally made. When agile advocates denigrate the value of documentation, they lessen their credibility in the eyes of many experienced professionals. The tone in arguing against non-essential documentation differs from arguing that documentation is an inefficient waste.

eXtreme Programming expert Bob Martin said at the 2001 XP Universe conference that he ran into someone who said his organization was using XP. Martin asked him how pair programming was viewed, and the reply was, “We don’t do that.” Martin asked how refactoring was working out, and the reply was, “We don’t do that.” Martin asked how well the planning game was working, and the reply was, “We don’t do that.” “Well,” Martin asked, “then what are you doing?” “We don’t document anything!” was the answer.

Success carries the seeds of failure, and the agile methodologists are concerned that some adopting these new ideas do not really understand what an agile methodology is – and it is not ad hoc, chaotic programming.

When considering process documentation, the element that is missing from agile methodologies, which is crucial for the SW-CMM, is the concept of institutionalization, i.e., establishing the culture that “this is the way we do things around here.”

Although implicit in some agile practices such as the peer pressure formed by pair programming, infrastructure is important for institutionalizing good engineering and management practices. The key process areas in the CMM are structured by common features that deal with implementing and institutionalizing processes. The institutionalization practices for each key process area map to all the goals within the area, so a naive agile implementation that ignored these cultural issues would fail to satisfy any CMM key process area.

As implementation models that focus on the development process, these issues...
agile methodologies, but they are arguably cru-
form for their successful adoption.

Over-documentation is a pernicious
problem in the software industry, espe-
cially in Department of Defense (DoD)
projects. Software maintainers have long
known that the only documentation you
can really trust is the code (and those of
us with experience debugging compiler
and run-time defects doubt even that).
Having said that, an architectural descrip-
tion of the system that provides a tour of
the top-level design can be invaluable to
maintainers.

From a technical perspective, as proj-
ec ts become larger, emphasizing a good
architectural philosophy becomes increas-
ingly critical to project success. Major
investment in the design of the product’s
architecture is one of the practices that
characterizes successful Internet compa-
nies [4]. Architecture-based design, desig-
ning for change, refactoring, and similar
design philosophies emphasize the need
for dealing with change in a systematic
fashion.

One of the compromises that agile
methodologists are likely to be required to
make as they move into larger projects
and applications that are life- or mission-
critical is a stronger emphasis on docu-
menting the architecture and the design
of the system. In turn, plan-driven
methodologists must acknowledge that
keeping documentation to a minimum,
useful set is also necessary. What benefit
do we really get from detailed designs
where the programming design language
is nearly as large as the code?

Much of the controversy with respect
to the technical issues centers on what
happens as projects scale up. Practices
that rely on tacit knowledge and highly
competent professionals may break down
in larger teams with their rapidly expand-
ing communication channels and coordi-
nation challenges. However, replacing
those practices with ones appropriate for
large teams may result in losing the emer-
gent properties of the agile methodology.

Customer Collaboration
Over Contracts
The degree of trust implicit in relying on
customer collaboration rather than a con-
tract is not justified in many customer-
supplier relationships. Even when the rela-
tionship begins with the best of intentions
and the highest of expectations on both
sides, one of the main difficulties in learn-
ing from experience is “the use of unaid-
ed memory for coding, storing, and
retrieving outcome information” [5], with
the consequence that “change can make
liars of us, liars to ourselves” [6]. As time
goes by, as things change, our unaided
memories become unreliable.

The reliance of agile methodologies
on tacit knowledge is therefore vulnerable
to perception shifts over time, yet tacit
knowledge may be much more effective
than external, explicit knowledge in set-
ing expectations and driving behavior. In
a government-contracting context, federal
acquisition regulations establish a context
for ensuring fair play – even if it is not
necessarily an effective and efficient envi-
ronment. This can be considered a prob-
lem in expectations management. The
agile methodologies manage customer
expectations by insisting on an ongoing
customer interaction and rapid iteration.

“One of the most
significant barriers to
implementing an agile
methodology is likely
to be an inability to
establish and maintain
close and effective
customer collaboration.”

Ignoring possible regulatory issues, the
stories in XP, in conjunction with an evolu-
tionary life cycle and ongoing customer-
supplier communication [7], document
requirements and commitments in a man-
er that could satisfy the goals of require-
ments management and software project
planning in the SW-CMM. Will such a set
of stories satisfy a DoD customer that the
requirements are adequately stated and
that commitments are driven by the customer
are being met? Or will the natural desire
for a more comprehensive requirements
statement drive the customer towards a
requirements specification that lacks the
dynamic capability desired for an agile
methodology?

Perhaps an honest answer to this type
of question reveals more about the com-
fort levels of both customer and supplier
in an agile relationship. One of the most sig-
nificant barriers to implementing an agile
methodology is likely to be an inability to
establish and maintain close and effective
customer collaboration – and this barrier
is likely to be erected on the customer’s
side of the relationship.

Responding to Change
Over Planning
Dwight Eisenhower is quoted as saying that
planning is more important than the plan.
And one of the great military axioms is that
no battle plan survives contact with the
enemy. That said, planning – and prepara-
tion – are prerequisites to success. Planning
for change is quite different from not plan-
ning at all.

Agile methodologies, with their rapid
iterations, require continual planning.
Customer collaboration and responsiveness
to change are tightly linked, if perhaps
inconsistent with typical government-con-
tractor relationships. One of the shifts in
acquisition strategy in recent years has been
toward prototyping, evolutionary develop-
ment, and risk-driven life cycles. With their
emphasis on addressing requirements
volatility, agile methodologies could be a
powerful synthesis of practices that DoD
contractors could leverage to make planning
more responsive to change.

Stepping Up to the Plate
The greatest challenge in taking advantage
of the virtues of agile methodologies may
lie in convincing acquisition agencies to step
up to the plate and use agile methods where
appropriate. Hardly lesser is the challenge in
convincing agile advocates to consider modi-
fying the agile methodologies to suit new
arenas. We have to decide where to place the
balance point in documentation and planning
to alleviate the concerns of the stakeholders
(and regulatory requirements) while achiev-
ing the flexibility and benefits promised in
the agile philosophy.

Agile methodologies may wind up being
the preferred process in many environ-
ments, yet be inappropriate in contexts such
as life-critical systems or high-reliability sys-
tems. Modifications to the agile methodolo-
gies needed for those environments may be
great enough that the synergistic effects of
the set of practices in an agile methodology
are lost. Emergent properties in a system are
sensitive to interdependencies. Arguing that
agile methodologies are not suitable for all
environments is not the same as saying they
are suitable for none.

Contractual commitments explicitly
based on evolutionary or incremental life
cycles are desirable. Plans with miniatures
milestones that are detailed in the short term
and conceptual in the long term are possible.
Processes that capture the minimum essen-
tial information needed to reliably and con-
sistently perform the work and documenta-
tion that captures useful information are
feasible. Just because these objectives are
desirable, possible, and feasible does not,
however, mean they are easily realized. Selecting an appropriate balance point requires an open mind from both agile and plan-driven methodologists on both the supplier and customer sides of the equation.

**Conclusions**

Agile methodologies imply disciplined processes, even if the implementations differ in extreme ways from traditional software engineering and management practices; the extremism is intended to maximize the benefits of good practice [8]. The SW-CMM tells what to do in general terms, but does not say how to do it; agile methodologies provide a set of best practices that contain fairly specific how-to information – an implementation model – for a particular kind of environment.

Even though agile methodologies may be compatible in principle with the discipline of models such as the CMM, the implementation of those methodologies must be aligned with the spirit of the agile philosophy and with the needs and interests of the customer and other stakeholders. Aligning the two in a government-contracting environment may be an insurmountable challenge.

As we learn empirically what works well in the agile methodologies and how far they can be extended into different environments, we should expect software engineering to adapt and adopt the useful ideas of the visionaries in the agile movement. This will include using data to separate the wheat from the chaff as we identify what is useful, and what is limited in its application. Laurie Ann Williams, for example, has integrated pair programming into an extension of the Personal Software Process called the Collaborative Software Process, and demonstrated that performance improves [9].

Many of the practices in the agile methodologies are good practices that should be thoughtfully considered for any environment. While the merits of any of these practices can be debated in comparison with other ways of dealing with the same issues, none of them should be arbitrarily rejected. Perhaps the biggest challenge in dealing effectively with both agile and plan-driven methodologies is dealing with extremists in both camps who refuse to keep an open mind.

**References**


**About the Author**

Mark C. Paulk is a senior member of the Technical Staff at the Software Engineering Institute. He has been with the SEI since 1987. Paulk was the “book boss” for Version 1.0 of the Capability Maturity Model® for Software and the project leader during the development of Software CMM® Version 1.1. His current interests center on high maturity practices and statistical control for software processes.