People and Methodologies in Software Development

Alistair Cockburn

Submitted as partial fulfillment of the degree
Doctor Philosophiae
At the Faculty of Mathematics and Natural Sciences
University of Oslo
Norway

February 25, 2003

Abstract

This thesis reports on research performed over a ten-year period, interviewing project teams, participating directly on projects, and reviewing proposals and case studies. The research addressed three questions relating to people and software development methodologies (**Q1** through **Q3**), and produced six results (**R1** through **R6**).

Q1. Do we need yet another software development methodology, or can we expect a convergence and reduction at some point in time?

Q2. If convergence, what must be the characteristics of the converged methodology? If no convergence, how can project teams deal with the growing number of methodologies?

Q3. How does the methodology relate to the people on the project?

R1. A methodology is a formula describing conventions of interaction between roles.

R2. People's characteristics, which vary from person to person and even from moment to moment, form a first-order driver of the team's behavior and results. Such issues as how well they get along with each other and the fit (or misfit) of their personal characteristics with their job roles create significant, project-specific constraints on the methodology. This result indicates that people's personal characteristics place a limit on the effect of methodologies in general.

R3. Every project needs a slightly different methodology, based on those people characteristics, the project's specific priorities, and the technologies being used. This result indicates that a team's methodology should be personalized to the team during the project and may even change during the project.

R4. A set of principles were found that can be used to shape an effective methodology to the above constraints. These principles deal with the amount of coordination and verification required in the project, the trade-off between rework and serialization of work, and the trade-off between tacit and externalized knowledge in use by the team.

R5. A technique was found to create a situationally specific methodology during the project and in time to serve the project, and to evolve it as the project progresses.

R6. All the above suggests a repeating cycle of behavior to use on projects.

1. The members establish conventions for their interactions — a base methodology — at the start of the project. This can be likened to them "programming" themselves.
2. They then perform their jobs in the normal scurry of project life, often getting too caught up to reflect on how they are doing.
3. They schedule regular periods of reflection in which they reconsider and adjust their working conventions.

These results have been used successfully on several industrial projects having the usual time and cost pressures on the staff.

Acknowledgments

I would like to thank (in chronological sequence):

1991-1994:

Wayne Stevens and Dick Antalek, in the IBM Consulting Group, for introducing me to the topic of methodology in a way that preserved respect for the individual, and for coaching me in the larger sense of the term “organizational methodology.”

Kathy Ulisse, in the IBM Consulting Group, for deciding on an open-ended enquiry into the subject of OO methodologies, for getting me on the track of asking development teams what they thought of their methodology without worrying about what I thought the right answer might be, and for supporting me with endless airplane tickets to perform the early investigations.

1996-1999:

Sean Cockburn, then aged six through nine, for continually insinuating that I really *ought* to have a doctorate degree, and doing it in such a way that I finally stepped into the task.

1991-1999:

All the people on the projects I interviewed and joined, who had no idea they were contributing materially to a doctoral thesis, who described their ways of working, showed me their work products, and speculated as to what helped and hindered them.

1999-2002:

Lars Mathiassen and Jens Kaasbøll for their insights on the subjects at hand, for repeated careful reading of draft texts, suggestions of literature I had overlooked, and encouragement during the task itself.

Table of Contents

1. The Topic

Question 1: Do we need yet another software development methodology, or can we expect a convergence and reduction at some point in time?

Question 2: If convergence, what must be the characteristics of the converged methodology? If no convergence, how can project teams deal with the growing number of methodologies?

Question 3: How does the methodology relate to the people on the project?

The first question is in some sense a preamble, since it has been asked by other authors. The two reasons for including it are the following:

- ? Whichever way the answer comes out, that answer sets up a tension for the remaining questions. In fact, it sets the framework not just for the other questions, but also for subsequent research.
- ? There are so many conflicting answers in the literature and in the industry that simply performing a literature search does not yield a safe answer. The question must be asked again to carefully establish the answer.

The second question is the heart of the matter. Whether "one methodology" or "an ever-increasing number of methodologies," the consequences ripple across methodology suppliers, methodology consumers, contract software houses, product design houses, and even teaching institutions. If the answer is one methodology, we should identify its characteristics, design it, teach it, and use it. If the answer is an ever-increasing number of methodologies, we need an approach to handle that ever-increasing number and to teach that approach to practitioners and people just entering the field.

The third question is a reality check. What if methodologies don't really matter at all? What if people on a project ignore their methodology? What if the characteristics of the individual people on the project has a bearing on the answer to the second question? I phrase it as an open-ended question because we can't know in advance of the research what form the answer might take.

The third question turned out to be more significant than it originally appeared. The characteristics of people significantly affect the way the methodology lives in the organization. Details specific to the project — not just the people, but also room layout and communication channels between people (all of which I call the project "ecosystem") — affect the design of the methodologies and the times and ways in which they are shaped to the project team. The interplay between ecosystem and methodology became the primary issue of interest by the end of the research.

1.1 Clarification of Words

With the word *methodology* being used in different ways by various writers, and often interchanged with the word *method*, I must clarify my use of the words in order for the rest of the thesis to make sense.

Methodology

My use of the word comes from two sources: the Merriam-Webster Dictionaries and common usage among large contract software houses.

? The (U.S.-based) Merriam-Webster dictionaries list as their first definition: "A series of related methods or techniques."

? When I sat in a discussion of methodology with a group of senior consultants from Coopers & Lybrand, Ernst & Young, Andersen Consulting, and the IBM Consulting Group, one of them stood up and said (words to the effect of): "Let's get clear what we mean by methodology. It is the roles, teams, skills, processes, techniques, tools, and standards used by the project team." The others in the room nodded their heads, and the meeting proceeded. As one of the senior consultants later told me, "If it is what one or two people do while working on a task, it is technique. If it is what the team uses to coordinate their work, then it's methodology."

Those two meanings of the word are congruent and match the common use of the word.

Over the time period from 1991 to the present, I broadened the word's connotation to match the research. I allowed "methodology" to connote "any conventions and policies the team relies on to successfully deliver systems." The purpose of broadening the connotation was to make it easier to detect what teams were paying attention to, to catch information that might slip by unawares if my definition was accidentally too narrow. The shifted connotation turns out to be helpful in practice. First, it is shorter, which is useful when discussing the subject with busy project teams and executives. Second, it brings forth the idea that the conventions and policies (a) are set forth in part by the team itself, and (b) are likely to shift over time.

Many British and continental European speakers dislike the above definition, preferring to use the word *method* instead. They cite the (U.K.-based) Oxford English Dictionary, which in the early 1990s contained only one entry for *methodology*: "study of methods." This explains the cultural difference in usage. However, the 1995 Oxford English Reference Dictionary includes in the entry for *methodology*: "2. a body of methods used in a particular branch of activity." Both American and British dictionaries now agree on the subject.

Method

The Merriam-Webster dictionaries define *method* as a "systematic procedure"; that is, similar to a technique. This matches the way many European speakers and European research literature uses the word *method*. An example, from Mathiassen (1998), is the following:

> Lesson 2: Methods are primarily frameworks for learning.

The intention behind Mathiassen's word *method* is quite different from my intention for the word *methodology*. Mathiassen writes *method* to connote what one or a few people are learning to use, usually as a technique. This learning eventually disappears into the background skill set of the practitioner.

In contrast, I write *methodology* to connote the conventions people agree to across the team (including development techniques). While the techniques disappear into each person's skill set, the conventions in play do not disappear into skill sets.

We shall need both words to complete the discussion. This thesis primarily addresses methodology rather than method.

Shifts in Meaning

As the research took place over a ten-year period, there were several minor shifts in the way the word *methodology* got used, as well as some shifts in the way the practitioner community used the word. I deliberately leave in place some of those shifts, because readers will come to this thesis with their own personal interpretations, and part of the thesis topic is dealing with those personal interpretations.

The first research question, for example, asks: "**Question 1:** Do we need yet another software development methodology, or can we expect a convergence and reduction at some point in time?" In this question, I do not mean just, "Do we need yet another set of *team conventions*, or can we expect a convergence and reduction at some point in time?" Rather, I accept the fluid way in which people ask the question, which often includes, "Do we need yet another software development *technique*, or can we expect a convergence and reduction at some point in time?" The answer that I develop responds to both versions of the question. Research question #2 is handled in a similarly liberal way.

However, by the time we reach the third question, "**Question 3:** How does the methodology relate to the people on the project?" the meaning narrows. It is quite a different question to ask, "How does a *technique* relate to the people on the project?" That question has to do with introducing techniques into groups, as well as the relation between technique-in-theory and techniques-in-practice (the latter topic being well discussed in the Scandinavian literature, see Ehn (1988) and Mathiassen (1998)). In the context of this inquiry, it is relevant to use the narrower meaning, "How do the *team conventions* relate to the people on the project and other aspects of life on the project?"

1.2 Background to Question 1

Software development methodologies (in the various meanings of the word) certainly have been invented in many forms and studied in many ways (Lyytinen 1987, Mathiassen 1998, Monarchi 1992, to cite just a few). There is conflict in the results, however. On the one side, some people conclude that no one methodology can fit all needs, and work has been done to list the factors determining the appropriate methodology for a project (Glass 1995, Vessey 1998, Hohmann 1997). On the other side, many people are uncomfortable with that idea. Executives still request to have one common process built for all their software development efforts. The earliest versions of the "Unified Modeling Language (UML)" standard were called the "Unified Methodology (UM)," until opposition and examination revealed that the UM was only a notational standard, at which point the term was shifted to the more appropriate "Unified Modeling *Language*."

On the basis of these mixed opinions, it is relevant to ask whether the future holds for us a *convergence* or a *multiplicity* of methodologies, and whether one of those outcomes is inevitable or a product of clever social maneuvering. The question has significant economic relevance to both providers and consumers of methodology.

In the 1980s and 1990s, Andersen Consulting, to name only one of several methodology suppliers, made good use of the fact that it had *one* methodology (METHOD1), and the client could rely on that methodology being used. Andersen Consulting saved money because it could train all its new employees in that one methodology, and the client could rely on the fact that when people were moved onto the project, they would know how the project was operating. Those are the strong attractions of having a converged methodology.

In the 1990s, Ernst & Young concluded that it needed different methodologies for different projects and built an expert system called Navigator that generated specific methodologies for specific projects. There was, fortunately, enough similarity between the different methodology texts produced that the staff still could be trained in a common way.

It should be obvious that the two companies would describe to their clients the advantages of their respective approaches and look to the client to make the appropriate selection.

The arrival of object-oriented (OO) technology in the 1990s muddied the waters, since an OO project uses different specialists on the team and produces different work products. Neither METHOD1 nor Navigator properly matched the needs of early OO projects.

However, even within OO circles, the same dichotomy quickly arose: one common methodology, or a set of methodologies? The motivations for the two answers are the same as before.

The question is a crucial one. If there can be one common methodology, it is worth a lot of money to whomever uncovers it, from the perspective of intellectual property rights. The answer matters equally greatly to the methodology consumers, but in the opposite way: Training everyone in the organization on just one methodology is much less expensive than having to train them and retrain them on constantly changing methodologies.

1.3 Background to Question 2

Answering the first question immediately opens the second question, with similar significance to both research and industry.

If there can be one methodology, and uncovering it is worth a lot of money to both suppliers and consumers of methodologies, we should aim to uncover what properties that methodology will have. From that piece of research, continuing research can put together the characteristics and details of the methodology.

If, however, no convergence is possible, the number of methodologies will continue to grow over time. This leaves methodology consumers in an awkward position with unpleasant consequences. How should they handle the ever-increasing number of methodologies? How will they choose among the competing suppliers and methodologies, and how should they allocate their budgets to training?

For exactly those reasons, organizations needing to buy methodology continue to request "the One" methodology they should use. (My evidence for this observation is personal discussions with corporate managers around the world and comparisons with other methodology suppliers.)

Among researchers and the larger methodology suppliers, the non-convergence school seems to be the most vocal. The Software Engineering Institute produced the Capability Maturity Model Integration (Ahern 2001). The standards body Object Management Group has produced the Unified Process (UP), which is not a single process, but a process framework (Jacobson 1999). Rational Corporation produced the Rational Unified Process, which, again, is not a single process, but a process framework (Kruchten 1999). The OPEN Consortium produced the OPEN Process, which is not a single process, but this time a construction kit from which a process can be built (Graham 1997). I have publicized my Crystal methodology family, which consists of a collection of different, specific methodologies plus a set of rules for choosing and tuning them (Cockburn 2002 ASD).

Independently of those multi-method efforts, new single methods continue to be invented and named. For example, Extreme Programming (Beck 1999), Dynamic Systems Development Method (Stapleton 1997), Scrum (Schwaber 2001), and Feature-Driven Development (Coad 1999) have all been brought to the public since 1997.

This leaves the industry in a state of division and begs for an answer as to how to deal with the consequences of the answer from Question 1.

1.4 Background to Question 3

Early on in this research, I kept hearing statements like:

> Just give me a few good people and let us work in the same room, and we'll deliver you software.
>
> At key moments in the project, a few people stepped in and did whatever was necessary to get the job done.

I heard these sentences uttered on projects using differing technology (SYNON, Sapiens, Smalltalk, Java, etc.), in different countries, in different decades. These sentences match Weinberg's observations from the punched-card 1960s (Weinberg 1998). The phrases did not fit any theories available to me at the time.

Indeed, underlying all these methodology discussions is the uncomfortable feeling expressed by many experienced developers that methodologies have a limited effect on a project's final outcome. Many developers I encounter are openly skeptical whether it is worth any effort to discuss methodology at all.

On the other side, effective, repeatable processes have been – and still are being – declared essential to the success of organizations (Royce 2001). Which claim is true?

? If methodology actually is *irrelevant* to a project's outcome, then all the producers and consumers of methodologies should be alerted and educated.

? If methodology is *partially significant,* that portion should be highlighted and put into perspective.

? If methodology is *very significant*, but needs to be matched to the local culture, then the interaction of a methodology to the culture should be explored and understood.

Question 3 asks about the effect a process or methodology has on project outcome, considers how people issues relate to methodology effectiveness, and comes to include how else the methodology is affected by the details of a live, ongoing project.

Jones (2000) captured details of projects over several decades and produced one published result: Reuse of high-quality deliverables and high management and staff experience contribute 350% and 120%, respectively, to productivity. In comparison, effective methods/process contribute only 35%. On the reverse side, he cites reuse of *poor*-quality deliverables and staff *inexperience* as having effects of -300% and -177%, respectively, and *ineffective* methods/processes at only -41%. Boehm's results (2000) on cost estimation places personnel/team capability as the most significant cost drivers, affecting productivity at 353%, and process maturity twelfth, affecting productivity only 143%. These results imply that methodology plays a far less significant role than is indicated by all the attention and money centered around it.

Weinberg presented his views of the interactions between people and project outcome in the 1971 text *The Psychology of Computer Programming* (Weinberg 1998). DeMarco and Lister presented their views in their 1985 book *Peopleware* (DeMarco 1999). Soloway and Hohmann described their results in the 1980s and 1990s (Soloway 1988, Hohmann 1997). Guindon and Curtis in the 1980s studied the way that programmers jump around across conceptual levels while solving a problem (Guindon 1992). In the 1980s' "Scandinavian School," Naur described programming as a private mental activity (Naur 1992). Nygaard (1986) and Ehn (1988) described it as a social activity. Mathiassen (1998) created a methodology that requires the developers to reflect while developing, in order to detect what is happening around them.

And yet, all these still don't reach the question. Just how does the methodology-as-declared interact with the project-in-action (and the reverse), how important are those interactions, and what should we do about them?

Mathiassen (1998) provides an extensive but still indefinite answer. He writes:

> Systems development methods were seldom, or only partially, followed by experienced practitioners.
>
> Each environment had distinct characteristics that played an important role for the success and failure of projects. . . .
>
> It was possible, but extremely difficult to change working practices. . . .
>
> Depending on which perspective you have, requirements management has different meanings and practical implications. . . . [C]ommunities-of-practice foster a kind of learning-in-practice in which the specific context and the shared experiences and understandings of its members shape how new methods are adapted, modified, and maybe rejected to transform present practices.
>
> Systems developers must open their minds and engage in reflections and dialogues to generate the necessary insights into the situation at hand.

This answer informs us about the problem we face and indicates the direction that educators and training leaders should pursue to produce more effective practitioners. It does not provide instruction for the team already in action. A useful result for this research will contain sharpened observations about the interaction between a methodology (the team's conventions) and the project's specific details, and will also produce some advice about how to deal with the situation.

1.5 Placing This Work in Context

The previous three sections outline where this works fits in the research literature. I should like to make it more explicit.

? This research work builds on and extends Weinberg's *Psychology of Computer Programming* (1971), which examines the nature of the software development activity, concludes that the answer deeply involves people, and sets out recommendations for handling people.

? It runs parallel to DeMarco and Lister's 1974 *Peopleware* (republished in 1999). DeMarco and Lister start from the assertion that people are important and discuss what it means for technical people to work in teams.

? It runs parallel to and draws from Ehn's *Work-Oriented Development of Software Artifacts* (1988). Ehn also examines the nature of the software development activity. Ehn pays special attention to the interaction between those people who will have to incorporate the resulting system into their work lives and the people who are working to understand what system to put together. My research differs in that I am particularly interested in the dynamics *within* the project.

? It runs parallel to the work of Glass (1995), who searches for domain, technology, and project taxonomies that might serve as a base for making initial methodology recommendations.

? It touches the work of Jones's *Software Assessments, Benchmarks, and Best Practices* (2000), but only tangentially, because Jones produces an overwhelming 37,400 project categories, a number too large to seed with methodologies. Jones's primary drive is toward benchmarks and assessments.

? It repeats some of Allen's 1970s research on the role of building layout and seating distance on project outcome (Allen 1984). His results were quantitative, while mine are qualitative; his were for research and product development organizations in several fields but not computer software, while mine are strictly for software development organizations.

? It intersects research work examining properties of software development methodologies. Particularly acute examinations and taxonomies were constructed by Lyytinen (1987), Monarchi and Puhr (1992), and Mathiassen (1998). I investigate methodology taxonomies in passing, but the final results lie beyond the question of methodology taxonomies.

? It runs parallel with work by Mathiassen (1998), the OPEN Consortium (Graham 1997), the Object Management Group (OMG), and Rational Corporation (Kruchten 1999) on dealing with multiple methodologies. I produce a different final recommendation, which is in some sense competitive with theirs and in some ways complementary.

The most similar research work, though, is that of Mathiassen (1998). He used similar research techniques, based on participation, examination, and theory testing on live projects and in more controlled classroom situations, employing dialectical inquiry to examine situations from opposing perspectives. He addressed a similar question, how methods interact with the characteristics of people. He arrived at similar results, that a method ("technique") lives in interaction with the practitioners' constantly varying practice, and practitioners should reflect on their work while they work.

The differences between our research directions, though semantically minor, yield different outcomes and lead in different directions. His writing deals with *methods*, mine with *methodologies*. The distinction now becomes relevant:

? Mathiassen's studies focus on the relation between a *person*, that person's working practices and the team, and the growth of a reflective practitioner.

? My studies focus on the relation between a *team* and the individuals within the team, and between the team-in-action and the methodology they declare.

The two outcomes and the directions in which they lead are complementary:

? Mathiassen's results lead to growing more practitioners able to reflect while at work.

? My results lead to techniques and recommendations that can be applied to teams whose members may not yet be "reflective practitioners." Those teams can nonetheless become more effective through these techniques and policies, and from periodic and assigned reflection periods, the individuals may develop some interest and facility in reflecting on their work.

1.6 Personal Motivation and Direction

The research direction a person takes is inevitably linked with his or her personal philosophy, early teachers or work supervisors, and the resources he or she can bring to the research. In my case, a set of experiences over the period 1986-1994 had just that effect.

1986-1991: Formal Specifications and Automated Tools

In 1986, I was handed the research assignment to design a tool that would translate from one formal protocol specification language to another. This assignment carried with it a set of assumptions about what really happens on a protocol design project and what could possibly affect the project in a positive way.

After some research, I concluded that the assignment was improperly constructed. I decided that no automated translation tool could possibly perform the translation properly, for the key reason that the *person* who writes the specification embeds naming, format, and structural clues in the specification to assist the next *person* who will read the specification. Each formal specification framework naturally admits of different sorts of such clues. An automated tool will not be able to carry those clues over to a different specification model. I was able to reject the research assignment.

I was then handed the assignment to design a tool to allow early simulation of new communication protocols. The idea was that describing the protocol to this tool would be sufficiently easy that the protocol designers would use it in preference to whiteboards (which are easy to use but perform no computation). This assignment also carried with it a set of assumptions about what really happens on a project and what could possibly affect the project in a positive way. In this assignment, I was permitted only one day to visit the protocol designers. I then had to guess what tool they would accept that would also improve their efficiency at creating new communication protocols.

The outcome was hardly surprising. After four years of work (1987-1991), our team had produced a new tool, a Ph.D. thesis, and a number of technical publications. The protocol designers, meanwhile, refused to consider the tool, saying it was too invasive of their personal ways of working. In other words, the research effort produced a "null" result: Nothing significant changed on the basis of our results.

With this null result in hand, I performed a quick research study of other efforts in formal program specification. Aside from a few isolated incidents of success (mostly where the originator was present on the project), I found a similar null research result across the field. A second inquiry showed similar null results in the field of computer-aided tool design (the isolated successes generally being in the areas of compilers, configuration management systems, and program profilers).

At this point, a certain level of skepticism crept into my research. For any research proposal, I would need to establish as early as possible whether or not the research topic would actually have a positive effect on project life and project outcome.

Further, I had become sensitized to the effect that a research-evaluation framework has on research. If the framework evaluates only tools, then non-tool ideas cannot succeed in that framework, and non-tool-building activities might even not be permitted. This applies equally to research on processes, methodologies, mathematics, or any other topic.

In 1991, I joined the IBM Consulting Group to research and create a globally applied software development methodology for object-oriented projects. I had again been handed an evaluation framework that included and excluded possible topics. In this situation, though, I was permitted, even encouraged, to visit project teams and hear what they had to say. The result was still to be a globally applicable methodology.

It dawned on me around 1994 that maybe we don't know what really happens on a project. Without knowing what really is in play on projects, we can only guess at what might make a difference in their outcome. To protect myself from continued null results, I would have to learn something about what really goes on in a project and feed that information into both the research topic ideas and the research evaluation framework. That is, my criterion for selecting a research topic and finding an answer would be that the topic mattered to project teams and the answer had a noticeable positive effect on the projects' outcomes. The rest had to be able to change.

Allowing the research evaluation rules to change meant that I was not merely performing research on a topic, but at the same time learning about what topics were relevant. Argyris and Schon (1996) refer to this as double-loop learning. Double-loop learning became core to my research approach and direction.

Double-Loop Learning

Argyris and Schon distinguish between single- and double-loop learning in the following way (see Figure 1.6-1).

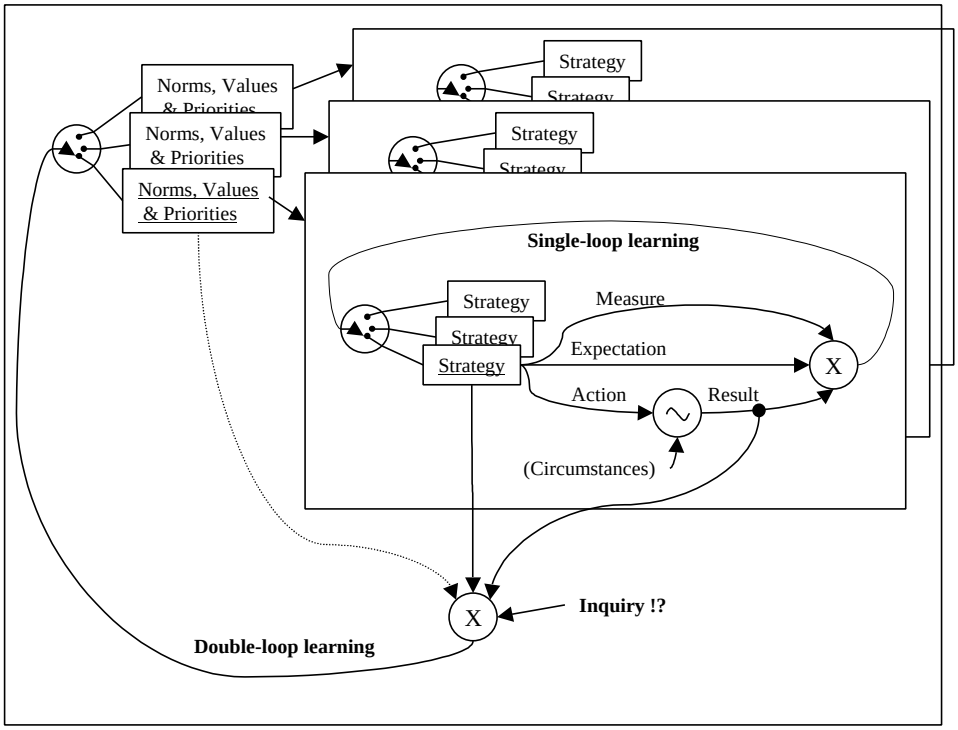

Figure 1.6-1. Schematic of double-loop learning.

Our norms and values give us our priorities. Based on these, we create and use strategies and assumptions to tackle tasks. The strategies and assumptions give us actions to carry out, plus expectations for the results of the actions and measures that we use to evaluate the result against the expectations. From this evaluation, we adjust our strategies, but not necessarily our norms, values, and priorities; this is single-loop learning. The problem is that only a certain range of strategies is consistent with these norms, values, and priorities. This means that the range of strategies that can be brought into play is limited, and therefore the range of results that can be obtained is limited.

In double-loop learning, we permit the norms, values, and priorities to be brought into question, and possibly altered. We do this by bringing the results plus the strategies themselves (and, eventually, the norms, values, and priorities themselves) into an inquiry. Inquiring about each of these

things may induce us to alter the norms, values, and priorities that are in place, which will consequently make available a different set of strategies and assumptions.

In performing this research, I started with the norms, values, and priorities of the tools-building community. Achieving a series of negative results and inquiring into the strategies available within the community, I shifted to using the norms, values, and priorities of the methodology community. Continuing to achieve what I perceived as inadequate results, I shifted to a dialectical model involving people and methodologies and started to see improved results. Each of these shifts required that I step away from the norms, values, and priorities of the previous community.

The Project Winifred Experience

The significance of being correct, incorrect, or ignorant as to what was really in play on a project became clear to me in 1994, in several instances on the project I call "Winifred" (and whose case study is included in the Appendix).

I was one of the lead consultants on the project, responsible for process, workstation architecture, OO design quality, and training people in various work techniques. I had, by that time, learned that developers will typically only follow the simplest process and use only the simplest techniques. The process was, accordingly, very simple, and the techniques were cut-down versions of responsibility-driven design (RDD) and use cases. (The specifics of these techniques are not of consequence to this thesis.)

After I had completed teaching a week-long course in OO design, I put on my ethnographer's frame of viewing to see how people were really working. (This dual mode of operation is described at some length in the next chapter.) I was shocked to discover that people were not using even the simplest form of RDD, although it is a simple technique, they had just gone through a week-long course on the subject, and I was available on the project to answer questions. This awoke in me the awful possibility that the smallest effective methodology one could design might already be too much for practitioners to practice on a live project.

A related moment came while I was using a state-of-the-art modeling and code-generation tool to prepare a drawing for a meeting. The tool was so awkward and inefficient to use that I found myself wishing I could just draw the picture on a piece of paper with a black marker — that would be sufficient for the purpose of the meeting and a lot faster (not to mention cheaper). With the addition of a paper scanner to put the drawing online, the paper drawing was equal to the computer tool in every respect we cared about on the project.

The third moment came when I watched an impromptu meeting form one day. Two people called in a third to ask a question. That third person pulled in a fourth person walking down the hall to corroborate some opinion. They phoned a fifth to check an assumption, and by now two other people had drifted over to listen and participate. In the end, the meeting included between six and eight people, many standing, some talking on the other end of a phone line. The whiteboard filled with various notes. As the issues were resolved, various of the participants drifted away. Eventually, the first three people settled the question. The answer was partly on the whiteboard and partly in text on their computer.

It dawned on me, watching that meeting, that even if I could capture in a technique guide exactly what had just happened and when to use it, that guide would be too complex for people to use on live projects. In other words, there is a fundamental limit to the process-writing activity, a limit both in detail and in value.

This last event raised questions relevant to this research thesis: If processes cannot usefully be specified in arbitrary detail, how are we to know which aspects of projects to capture? Which aspects are relevant to study, and which of those are simple enough to be applied in a second situa-

tion? What is actually going on, what out of that matters, and in what way can any of that be transferred to another project?

Post-Winifred

In 1995, I became an independent consultant. This freed me to expand the bounds of inquiry, to include many more factors than otherwise would be possible. My work provided access to what I call "live projects" — real projects in competitive situations, with real deadlines and cost pressures. I had access to almost all meetings in those projects, and sometimes I could experiment with new strategies coming from my research. My consulting work also gave me time to interview people in other projects, access to experts, and time to research and reflect. Without external agencies to impose their evaluation frameworks, I was better able to take advantage of the double-loop learning and change directions over time to accommodate the results from that learning.

2. The Approach

The research was a ten-year study of factors affecting the trajectory of software development. Over time, the study came to focus on methodologies and people factors. Both the time and the breadth of the study required multiple techniques of inquiry plus multiple feedback loops to evaluate the theories being mounted.

The research was carried out by reading the technical literature; by collecting observations through literature, interviews, and direct observation; and by joining and acting on live projects. Alternating between these activities gave me opportunities to create, test, and evaluate a variety of theories. Keeping notes over time permitted me to evaluate new proposals against old observations and generate new theories using grounded theory approaches.

In this practice, I employed multiple standard research approaches: field experiments, case studies, subjective / argumentative, empirical, reviews, action research, and descriptive / interpretive. My use of them can be seen to fit into the more recently described research approach *collaborative research practice* (Mathiassen 2000), which is an application of the principles of reflective systems development (RSD) (Mathiassen 1998) to the practice of research.

I used dialectical inquiry to detect issues that single-perspective investigation might miss. Two dialectical pairs were used in the research: methodology focused versus people focused, and description versus prescription. The outcomes of the research work were directed alternately toward both the practitioner community and the research community. Thus, each paper produced was focused as some combination of

$$\begin{bmatrix} \text{Methodology} \\ \text{People} \end{bmatrix} \times \begin{bmatrix} \text{Descriptive} \\ \text{Prescriptive} \end{bmatrix} \times \begin{bmatrix} \text{Practitioner} \\ \text{Researcher} \end{bmatrix}$$

What follows is a description of the practices I used in carrying the research forward and the research theory in which those practices are placed.

2.1 The Research Practice

The research practice was oriented toward three ongoing activities:

? Studying theories about the development of software

? Collecting observations

? Acting and observing directly on live projects

As described in the section "Personal Motivation and Direction," I discovered that the research required me to both learn about what might be relevant as well as to research particular topics. I had to use Argyris and Schon's double-loop learning model.

Using the double-loop learning model had several implications for the research practice. One was that I had to keep notes from early project investigations to review later, when the research evaluation framework changed. This practice, related to Ngwenyama's Longitudinal Process Research (1998), is described over the course of this chapter.

The other consequence was that my research practices were almost exclusively qualitative (as opposed to quantitative). I found that I couldn't place much reliance on my or other authors' perceptions of what really happens on projects. I broke theories with such increasing frequency over the period 1987-1997 that the biggest question for some time was simply what to pay attention to. While I may now be willing to claim strong enough qualitative results to start a quantitative inquiry, I certainly did not have a sufficiently reliable vocabulary along the way to create useful quantitative instruments.

To carry out the double-loop learning, I kept two groups of activities in play at the same time, with multiple feedback loops between them (see Figure 2.1-1). We should keep in mind that I did not draw this drawing at the beginning of the research. The drawing is an after-the-fact construction to permit the reader to visualize the feedback loops in play. It was always clear that I would need to do each of the six activities in the figure, but their relationships grew over time, particularly as I added double-loop learning to allow for shifting assumptions about the research theories.

The first group consists of my ongoing physical activities: reading theories about software development, collecting observations, and acting with projects. Each informed my construction of an "espoused theory" (Argyris 1996) for evaluating what is going on within a project. Having a theory in place, I collected observations to see whether the theory had good descriptive and predictive power and joined projects to see whether the theory gave useful advice for action.

On a project, I sometimes employed the espoused theory to create an experiment. This helped test the theory, grow the theory through small revealed discrepancies, and eventually break the theory when I detected large discrepancies. If the theory was not performing well on the project, I ignored it and did whatever was needed to accomplish the assignment. I later reflected on "what had happened." This reflection constituted another observation that would test or help reform the espoused theory.

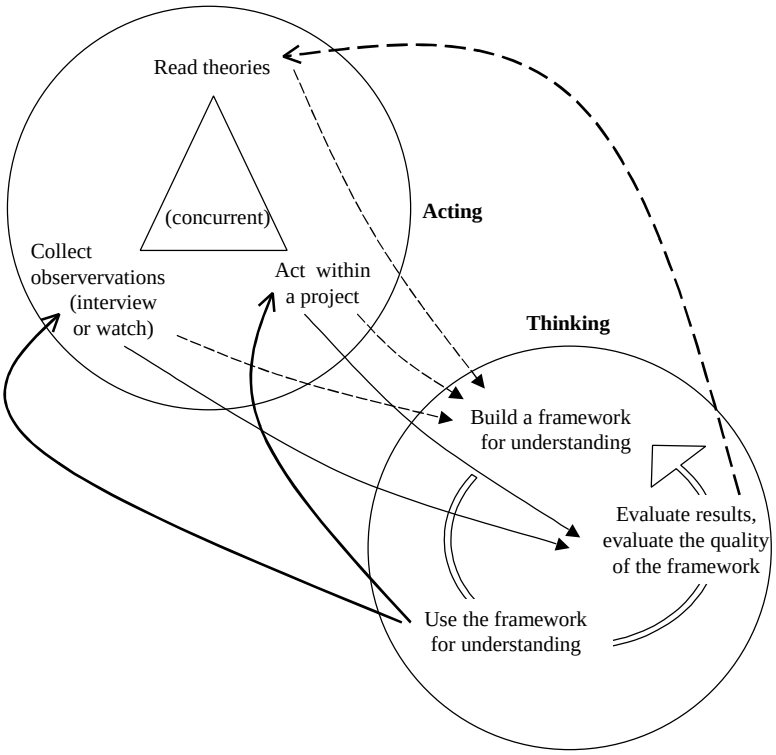

Figure 2.1-1. Schematic of the research practice.

The second group of activities were the mental activities related to the theories: constructing, using, and evaluating them. Reading the theories in the literature, watching and listening, and acting and reflecting all contributed to the construction of an espoused theory. Using the espoused theory on projects gave feedback as to how congruent the espoused theory was to the theory-in-action. I used the theory in a similar way while listening to or reading about a project's history: I tried to forecast the next part of the project history using the theory and evaluated the theory according to how well it predicted the story. Using the theory predictively provided information I could use to evaluate the quality of the theory.

Finally, as each theory grew unsatisfactory, I searched my notes, thoughts, and the literature for new avenues to pursue. This was the double-loop learning: I changed the base assumptions about the theory I was currently using and generated an arbitrarily different theory to pursue for the next period. Since 1998, I have been running multiple sets of assumptions concurrently, growing and discarding elements from the fundamentally different theories concurrently. I find that doing this permits faster evolution of the theories, although it is sometimes cognitively difficult.

Reading about Software Development

During different periods, I read different materials that might pertain to the framework in which I was currently operating. In the 1987-1991 period, I read about formal specifications and formal development techniques. In the 1991-1999 period, I read about processes and methodologies. In the 1999-2001 period, I read about social structures and organizational learning. In each period, I found the authors asserting, “The development of software is like this . . ." This reading seeded my thinking (and blocked some of my thinking!) and provided a reference framework against which to test my own phrasing of theories. Eventually, of course, I encountered the limitations of each frame of expression.

Collecting Observations

I began by reading the literature about OO development, literature that *described* particular OO development approaches and literature that talked *about* the approaches that had been described.

It became quickly clear that I had no basis upon which to evaluate the truth or falsity of claims made in the literature. I therefore started interviewing project teams, a practice that I continue even now. People continue to say surprising things, surprising in the sense that no espoused theory I have on hand adequately captures what they say, and also surprising in the sense that they describe new techniques or technique variations to develop software that I have not seen before.

To gather comments that would represent a wide cross-section of software development situations, I interviewed people in companies ranging in size from 20 people to 300,000 people, which were located in the U.S., Canada, England, Switzerland, Norway, Sweden, New Zealand, South Africa, or combinations of the above. Most teams worked in corporate IT departments or contract houses that built software for them. A few were building products for external sale, and more recently, for direct use on the Internet. The team sizes varied from 3 to 150 people, the project duration from a few months to ten years. The languages in use varied from assembler to COBOL, Visual Basic, Smalltalk, LISP, C++, and Java. Some were successful, some were unsuccessful.

Initially, I interviewed any project I could find and get permission from. I quickly discovered that the failed projects had a great deal in common and did not inform me on how to succeed. Two teams succeeded so easily that it was hard for them to say, and hard for me to detect, what they were doing that was so different. The best information came from projects that encountered difficulties and then succeeded because they had changed something significant along the way, and we could dig together for what that might be. The difference in information content between successful and failed projects was so significant that I stopped asking to interview failed projects and worked to interview primarily successful projects. Of course, if I came into contact with a team from a failed project, I did not exclude them from observation. For the purposes of selecting interviewees, I deemed a project successful if software was delivered to and used by users.

Selection of interviewees: My preferred mode of operation was (and is) to interview at least two people from the team. Given the natural human tendency to revise or invent history, and the obviously different views of project life held by people with different job roles, I tried to interview at least one developer and either the team lead, project manager, or process designer. I selected these roles because the person in the leadership role knows what he or she *wishes* to happen, and so tells a smooth story about the alleged process in action. The developer, on the other hand, usually tells a different story, one of compromise and arbitrary communication, with emphasis on getting things done rather than on fitting an organizational model of behavior. A typical example is the following:

> The process designer / manager told me the group had been using the OMT development methodology in an iterative approach and showed me sample work products created. I turned to the developer and asked:
>
> Me: Did you produce these drawings?
>
> Him: Yes.
>
> Me: Did you use an iterative approach? Did the requirements or the design actually change between iteration one and iteration two?
>
> Him: Yes.
>
> Me: How did you work to update these drawings in the second iteration?
>
> Him: Oh, we didn't. We just changed the code.

The manager / process designer either was unaware that the drawings had not been updated or was attached to the story that they were being updated, and in either case did not reveal that information to me. It was necessary to look for cracks in the story where the people might have deviated from the idealized process and ask someone what they recalled happening.

Initially, I trusted the manager or team lead to give a fairly accurate description of the process the team used. This trust was very short-lived. I had to severely discount (though not exclude) those project interviews in which I only got to talk to one person, even the manager or process designer. I count such interviews as "interesting stories told from one person's viewpoint." As such, any final theory should be able to account for the reasonableness of the story being generated by a person in that role. It is valid for testing theories, less valid for generating theories, and does not count at all as an objective rendition of the project history.

Questions used. I used primarily semi-open questions in the interviews. It was clear that using only closed questions (e.g., "Did you use any Upper-CASE tools?) would miss too many nuances in the recounting, and I would miss information that would later turn out to be important. At the same time, using completely open questions (e.g., "Tell me how you felt about the project.") would allow the interviewees to ramble and make it hard for me to compare notes across interviews and identify trends. I attempted to gather quantitative data at the beginning of the research: number of classes, lines of code, shape of the class hierarchy, etc. However, so few project teams could provide that information that I discarded it from the questionnaire.

These are the questions that I used to direct each interview:

? Please give me a brief history of the project: timeframes, changes in project size, problem domain, technology used, key events in the life of the project, the general approach.

? What would you say are the things that worked in the project?

? What would you say are the things that did not work well?

? If you were going to start another project, or give advice to someone starting a similar project, what things would you be sure be put in place (recommend)? What are your priorities for those?

? If you were going to start another project, or give advice to someone starting a similar project, what would you work (recommend) to avoid, or keep from happening?

? How would you prioritize the tools you used?

? Do you have any other thoughts about the project?

During the discussion around the question, if I sensed an interesting topic lurking below the conversation, I would branch to follow that topic.

> In one project, I heard the person say that the original team of two people grew to be three teams of six, two in the U.S. and one in Brazil. I asked, "Oh, and do all three teams follow this same methodology?" The interviewee, a team leader, answered: "No, the people in Brazil refuse to work this way. The other U.S. team skips major parts of it. But my team does it all."
>
> This gave me the opening to ask, "And how does your team feel about it?" His answer was, "They detest it, but I make them do it anyway."
>
> Later in the interview, I observed that there were a quite a number of work products that looked as though they took a great deal of time to prepare and asked how that all worked out. He replied, "Well they fired the project manager after a year because we hadn't produced any running code, and shut down the project. But I kept going, and within a few days had automatically produced executable code from that final model, there. We really could produce

production code from our model!" (At this point he showed me the award he had received for instituting a repeatable process in the company.)

Artifacts examined. I asked to see a single sample of each work product produced by the team. In the case of the Extreme Programming project, this consisted of story cards, the whiteboard with the task assignments, a section of code, and a section of unit test code. In the case of a Shlaer-Mellor project, there were 23 distinct work products produced by the programmer.

Examining the work products, I could generate situation-specific questions, such as the relation between the 23 work products ("These two work products look as though they contain quite a lot of overlapping information. Did the tool set you used automatically generate the overlapped information in the later work product?" . . . ["No."] . . . "How did your team feel about that?" ["They didn't like it, but I made them do it anyway."])

Acting on Projects

To gather data, to test theories (and of course, to help pay for the research), I joined project teams. I took on various assignments over the years: programmer, lead programmer, teacher, mentor, and project management advisor (not in any particular sequence).

In joining a team, my first promise had to be to participate in developing the system at hand, in whatever role I was needed. In general, I could not say that I was conducting research. The attitude of most project managers doing the hiring was that doing research would interfere with the progress of the project. When I employed an ethnographer one time, I had to provide her with a legitimate role requiring her to be with me at all times, so that she would have adequate access to the discussions taking place.

Typically on each project, I found one or more people open to the subject of improving their practice, testing theories of software development, and adding to the research base. We discussed, tested, and regenerated theories either openly or just amongst ourselves, as the project atmosphere permitted.

In the roles I was given, I was able to join almost every sort of meeting, from scheduling the project, to deciding on the process or tools to use, to teaching developers and project managers new techniques, to designing, testing, and packaging the system. During the meetings and during the regular daytime activities, I was not always fully engaged as a participant. Periodically, during those times, I shifted from being a participant to being an observer. I alternately took on the view of an uninformed observer and the role of an informed observer.

As an *informed* observer, I could apply my knowledge of the people, their tasks, and the subject matter to detect and interpret moves that might not be apparent to the uninformed observer. In one such case, I watched the way in which two junior programmers sequenced their movements as they worked through the daily defect list. I was later able to detect a pattern of search in line with Peter Naur's 1986 paper "Programming as Theory Building" (and later gave one team member an award for excelling at Use of Theory Building during Maintenance).

In the role of an *uninformed* observer, I simply watched people come and go, speak and interact, with little technical interpretation, in order to detect items that might slip past someone looking for particular patterns of behavior. In one such case, an impromptu meeting grew from two to eleven people crossing multiple job roles, reached a resolution, and then dispersed. I simply noted who came by accidentally, who was called in, and other superficial details of the meeting. In another case, I employed a trained ethnographer to capture moment-by-moment observations of meetings and also to apply ethnographic analysis to the results.

As a senior member of the project teams, I was in the position to perform experiments. Some of the experiments were reversible, and some were not. We were able to reverse experiments with team structures, tool usage, level of detail in project planning, and work products used, to a certain extent. We were not able to reverse commitments to more central tools and work products or commitments to delivery schedules and financing. I experimented with the base vocabulary I used to express key ideas about running the project and observed how the people acted and expressed themselves with respect to that vocabulary.

Making and Breaking Hypotheses

To perform research, one operates from a hypothesis of some sort. At the most global level, the hypothesis of is the form, "Addressing this topic will make a critical difference in projects' outcomes." Initially, I thought that improving formal specification techniques might make a critical difference. That hypothesis broke when I saw the difference in thinking ability required by formal techniques against that available in most software development centers. I moved to the hypothesis that improved simulation tools would make a critical difference. The antagonism most developers feel toward those tools and the ease with which they can ignore tools they don't like broke that hypothesis. For several years I ran the hypothesis that methodology makes a critical difference. I was forced to partially abandon that hypothesis after watching and interviewing successful project teams who played fast and loose with their declared methodologies. What remained from that hypothesis is that methodologies matter more with increasing team size, communication distance, and project criticality. I currently work within the global hypothesis that people issues dominate project outcome and that addressing this topic will make a critical difference in projects' outcomes.

I formed the hypothesis that a sufficiently simple technique would actually get used. The experience on Project Winifred described earlier, in which the developers would not use even the simplest design technique I could find, broke that hypothesis. One hypothesis I currently carry with me is that simplicity in a technique is not sufficient to get it used.

I have adopted and rejected the hypothesis that it is a possible to construct a universally applicable methodology. This is major topic in the thesis. The event that finally broke it in my mind was sitting in Norges Bank in 1997, trying to come up with anything at all that could be said to apply to a low-priority, one-person SQL project; a low-priority, two-person Java/web project; an important, eight-person COBOL/Assembler/CICS project; and a mission-critical, 35-person Y2K conversion and testing project. I similarly rejected the hypothesis that it is possible to write down a detailed process that transfers across projects (see page 16 for the situation that broke this).

For any theory I had that should cross project and culture boundaries, I eventually visited a project where that did not apply. I even ran the hypothesis that heavyweight processes fail, but then I met a team that delivered a large project using a heavy, waterfall, paper-driven process. I ran the hypothesis that people are not consistent enough to follow high-discipline techniques such as Watts Humphrey's Personal Software Process (Humphrey 1995) or the Dijkstra-Gries program derivation technique (Gries 1986), but then found people using both of those.

I ran the seemingly odd hypothesis that if you simply put four people in a room with whiteboards and computers, give them access to the users, and have them deliver running software every month or two, they would have a high likelihood of success in delivering useful, working software. Surprisingly, this hypothesis has not yet been broken. Its continued success feeds me with new research questions, such as, "What is happening in that situation such that they succeed?" and "What critical elements are transferable to other situations?"

I have run the following hypotheses at various times:

- Increment length doesn't matter. Increment length does matter.

- Four months is the longest effective increment length.
- A team can only be effective with six or fewer people in a room.
- Automated regression tests are crucial. Automated regression tests don't matter.
- CASE tools are valuable. CASE tools are irrelevant.
- Anyone can be trained to program.
- Teams operate more effectively without a project manager.
- The printing part of printing whiteboards is important.
- Techniques are the most important single part of a methodology.
- Milestones are the most important single part of a methodology.
- Standards are the most important single part of a methodology.
- Dependencies are the most important single project management element.
- People want to play all roles on a project. People want to choose their own roles. People want to know the grand vision for the project.
- Team morale matters.

The list goes on. Most of the theories got broken along the way. A few of the above are still in play.

Gathering and Reviewing Data

In some research projects, the mode of gathering information is fixed: interviews, surveys, detailed studies, and so on. The drawback of fixing the information-gathering approach is that a great deal of information slips by unnoticed. I wanted to take advantage of as much information as I could, to feed the reflection and inquiry process that would steer the direction of the research.

Some of the information I gathered is of the more consistent, rigorous, and archivable form. I used the same questionnaire over a period of ten years, indexing the notes by project. At various times, I put them all on the floor and examined them longitudinally, looking for trends, ideas, or contradictions. I was able to keep some documents from projects: project plans, designs, meeting minutes, methodology designs. I reviewed these when I needed to check how we expressed ourselves at that time, what we mentioned, what we left unmentioned.

Some of the information I gathered was low-grade or transient information, as when a programmer said during a coffee break, "Oh, well, we evaluated a bunch of CASE tools and decided that they would slow us down too much, so we didn't use any." I often did not have a chance to follow up on such comments. It would be tempting to discard them as invalid data. However, we can record that this sentence was expressed by a programmer, and if he was speaking truthfully, there was time spent on some evaluation of some CASE tools, and however they carried out that evaluation, the end of the story was that they concluded they didn't like them so much, and now CASE tools are not in general use on this project. We could also note the characteristics of the company involved, team sizes, country, culture, and so forth. I rarely had time to make all those notes on paper, given the social situation at the time, and so I used such sentences as hints for listening in other circumstances or as leads for further inquiry.

The following episode provides an example of using casual remarks to generate leads.

> The person sitting at the next table in the coffee shop was reading a book on becoming a good salesman. We started talking. He told the story of a consultant who visited his company and said words to the general effect of, "If I go into a company and see one super-salesman who is

> much better than all the rest, I tell the owners to fire this man immediately. Once he is gone, all the others improve and net sales go up."
>
> As it happened, I had just visited a project team consisting of one super-programmer and many ordinary programmers. I immediately called the president of the company and told him this story from the coffee shop. He said that he had seen this effect among stock traders and that it might apply to his situation with the programmers. We discussed various options and decided to bring the super-programmer into the discussion, to look at ways he could be excellent without acting as a ceiling to the others (he might, among other ideas, change departments or work at home).

No hard data was recorded, but the two conversations contributed to generating a small theory worth investigating through other approaches. In fact, it was a collection of many comments like this over time that led me to look through my interview notes for comments about people, use grounded research theory on those notes, and investigate the effect of individuals on project outcomes.

Figure 2.1-2 illustrates this gathering of information using multiple approaches. Any moment in any day provides an opportunity for investigation, if we can recognize it. Some opportunities are deliberate, as when I requested to interview a project team or hired an ethnographer. Others are half-deliberate, as when I joined a project team, sat in meetings, or worked with other people. Still others are opportunistic, as when the person at the coffee table starts talking about CASE tools or super-salesmen.

For each opportunity, one or more research methods could be brought to bear. The characteristics of the opportunity carried with them implications about the quality and archivability of the information gathered. In the worst cases, I had only my memories to replay when a new theory was brought into view. Where possible, I at least created a shared memory with another person through explicit discussions about what was happening at the time and the theories that might reflect on it. This allowed me to check my memory later with someone who had been there at the time. In the best cases, I had written artifacts on file that coincided with the theory being tested.

The news is, of course, good and bad. The bad news is that large parts of the ongoing store of research information are in my own memories and my shared memories with other people. The good news is that I was able to make use of more information than otherwise would have been possible. The factor that mitigates reliance on unreliable data is that, however the theories were generated, they were tested for effectiveness on live projects, as well as for predictive value while listening to accounts of project life.

The various output provided additional information for the archival store. I wrote reports to the companies I had visited. I put some case studies into the literature. I described project episodes in research reports. These reports themselves became part of the information repository, and I reviewed them periodically

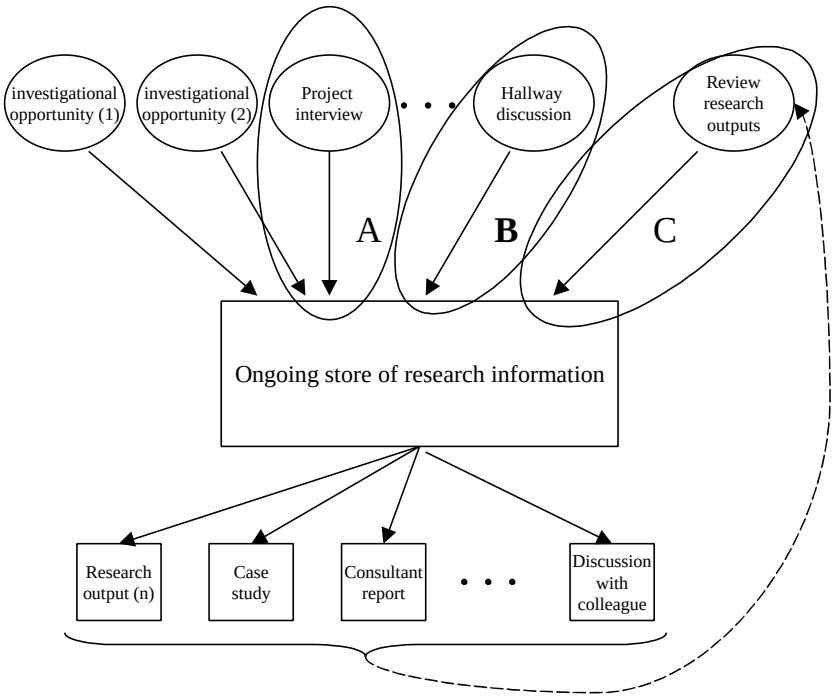

Figure 2.1-2. Making use of multiple information-gathering approaches.

2.2 The Research Theory

In this section, I relate the research practice to the research approaches conventional at the time, and then to Mathiassen's more recent collaborative practice research approach.

Traditional Research Approaches

Galliers (1992) describes 14 different research approaches that might be used: laboratory experiments, field experiments, surveys, case studies, theorem proving, subjective / argumentative, empirical, engineering, reviews, action research, longitudinal, descriptive / interpretive, forecasting / futures research, and simulation.

Of those 14, I used field experiments, surveys, case studies, action research, subjective / argumentative, and descriptive / interpretive approaches. The following characteristics of these six are taken from Galliers and related to my research practice.

Experiments

Experiments in a laboratory setting allow one to control variables at the possible expense of realism. Field experiments have the opposite characteristic: realism at the expense of control. Even within field experiments, the degree of formality and control can vary.

I consider two of the practices I used to fall into the experiment category, both of them field experiments.

The more carefully controlled were experiments I ran in courses on object-oriented design and on use case writing. I taught each course approximately 20 times in the research period, generally to students who were post-college, practicing developers on live projects. Initially, I taught in a single, consistent teaching style. Over time, I introduced variations in presentation, sequencing, and the techniques introduced. I compared the class responses to those variations to the baseline behavior I had observed in previous classes.

> One experiment I ran over a period of several years was to ask the attendees, at the very start of the course, to design a bank and present their bank design to the class. This experiment was designed to bring out the students' notion of what "design" was and to observe how they represented their idea of "design" to others, in words and pictures. I used the results to formulate theories about communicating designs and tested those theories against the discussions used on projects (and then, of course, used the results back in the classroom, etc.).
>
> Another experiment was to investigate ways to raise the quality designs created by novice OO designers. I had been teaching exclusively responsibility-driven design, but was dissatisfied when the students would miss key domain abstractions in their designs. I experimented with introducing domain analysis at different points in the course and observed the extent to which the students could follow the discussions and generate designs containing the abstractions.

The less formal experiments were those I used on live projects. Due to the delicacy of staying within the boundaries of project success, I had only a limited range of experimentation and limited measurement devices. The simplest experiment was to vary the conceptual language I used at the start of the engagement and note the extent of congruence between my language and the successive speaking and behavior within the team. If the successive behavior was congruent but ineffective for the problem at hand, I had to act to support different behavior, and I considered the experiment as producing a negative result. Successive behavior that was effective but incongruent I also considered as a negative result, as it indicated a shortcoming in the espoused theory being tested.

> An example of a successful result was the use of the vocabulary of incremental development, and in particular the "VW" staging model (Cockburn 1997 VW). After introducing the VW vocabulary and concepts to one team, the team leader, that very night, rebuilt his project plan according to the constructs, because, as he said, he knew the old plan was bad, but he didn't know how to fix it or how to present it to his boss. Several projects running according to this model succeeded in their planning and tracking activities, staying within the VW vocabulary.

The least controlled sort of experiment was to vary the strategy in use on the project. At different times, we would shorten or lengthen the increment period, or add, remove, or change documentation demands. I count as experimental data those times when the strategy was deliberate, I could watch it get introduced into the project, and I was there to observe the project and the team over time as the project history evolved.

Surveys

Surveys include literature surveys and structured interviews. I performed a literature survey at the start of the research period and included it as part of one paper included in this thesis ("The impact of object-orientation on application development").

The structured interviews I used were described in the previous section. Galliers writes that these sorts of surveys "are essentially snapshots of practices, situations, or views at a particular point in time, undertaken using questionnaires or (structured) interviews, from which inferences may be made." I performed structured interviews, as opposed to questionnaire-based or multiple-variable surveys.

Two observations emerged from the interview notes. The first was commonality in response to the question, "How would you prioritize the tools you used?" The common response was: (1) configuration management, (2) communication, and (3) performance tuning (there was no common fourth answer). The second was local to one company, in which four people each did two or three interviews. They found a common project success criterion ("Where we had good communications, internally and with our sponsors, we did well; where we didn't, we didn't do well.").

Structured interview surveys have the strength of revealing trends quickly. Their weakness is that most of the questions were open-ended, making quantitative and statistical processing impractical. Where there was a homogenous group of interviewees, the sample size was too small for statistical effect.

In all cases, it should be clear that surveys capture espoused theories, not theories-in-use. The information they provide must therefore be rechecked against behavior on live projects.

Case Studies

Case studies are descriptive reports of project or episodes from a project. From a research perspective, each case study provides a data point. The strength of the case study is that it captures the local situation in greater detail and with respect to more variables than is possible with surveys. Its weaknesses are the difficulty in generalizing from a single case study and the unintentional biases and omissions in the description. Case studies are helpful in developing and refining generalizable concepts and frames of reference. I used cases studies both from my notes and from the literature to detect similarities with and contradictions to my current theories, and to detect patterns of behavior not described in those theories.

Action Research

Action research is an approach in which the researcher participates directly in a project. According to Hult (1980), "Action research simultaneously assists in practical problem solving and expands scientific knowledge, as well as enhances the competencies of the respective actors, being per-

formed collaboratively in an immediate situation using data feedback in a cyclical process aiming at an increased understanding of a given social situation, primarily applicable for the understanding of change processes in social systems and undertaken within a mutually acceptable ethical framework."

In action research, the intended consumers of the research results are not only the research community, but the involved practitioners themselves. Galliers comments that the strengths are the fact that the researcher's biases are made overt in undertaking the research, while the weaknesses are its restriction to a single project and organization, the accompanying difficulty in generalizing results, lack of control over variables, and openness of interpretation.

Subjective / Argumentative Research

Subjective / argumentative research is creative research based primarily on opinion and speculation, useful in building a theory that can subsequently be tested (Galliers). Its strength is in creation of new ideas and insights; its weakness is the unstructured, subjective nature of the research process. As Galliers put it, "Despite making the prejudice of the researcher known, there is still the likelihood of biased interpretation." This form of research was in play each time we invented new strategies of action on a project and each time I reformed the base of assumptions for a new theory.

Descriptive / Interpretive Research

Descriptive / interpretive research is phenomenological. "The argument goes something like this: All we can ever know are phenomena, since there is no such notion as a 'thing in itself.' However, once we have understood phenomena correctly, we know all that there is to be known. . . . The strengths in this form of research lie in its ability to represent reality following an in-depth self-validating process in which pre-suppositions are continually questioned and our understanding of the phenomena under study is refined. The weaknesses relate to the skills of the phenomenologist and their ability to identify their biases and unheralded assumptions" (Galliers). This approach was used in the current research to question the project histories as being told by the interviewees and in reflection about the live projects.

Three additional research approaches must be mentioned: longitudinal process research, grounded theory, and collaborative practice research.

Longitudinal Process Research

Longitudinal process research (Ngwenyama 1998) was developed in response to some of the perceived weaknesses of action research. It calls for systematic data collection over a long period of time, which can be reviewed to reveal new patterns as new ideas are put forth. Its strength lies in the richness and amount of data that is collected; its weakness lies in the long time duration involved and the close ties needed to obtain and retain the data. The process I followed falls short of rigorous longitudinal process research, since I was not able to keep detailed records of project designs and meeting minutes. I was, however, able to gain some of the benefits of longitudinal process research through the data bank of project notes I had built up from 1991 to 2001, and from the longer-term involvement I had with three projects: one 18 months long from project proposal to delivery, one 14 months long from project proposal to delivery, and one 6 months long, an excerpt from the development period, which itself stretched over a 10-year period. On all three I was able to follow up with the project participants over the years that followed, extending the duration of observation for those projects. The data bank of notes proved useful over time as new theories were generated, as I already described.

Grounded Theory

Grounded theory (Glaser 1967, Thoresen 1999) is an approach for generating theory from collected situational data, “discovered, developed and provisionally verified through systematic data collection and analysis of data pertaining to that phenomenon. . . . One does not begin with a theory and then prove it. Rather, one begins with an area of study and what is relevant to that area is allowed to emerge” (Thoresen citing Strauss (1990)).

I grew into using grounded theory to generate hypotheses from my data bank after one theory after another failed to be congruent with or predictive of project trajectory. Ideally, one has a very rich set of objective data, such as from video and audio recordings, with which to do the grounded theory work. I could only work from the case studies in the literature and my notes of live projects, fitting Thoresen’s observation (p. A-130): “research cannot always choose its approach and topics independently when the project has an action interest. Research must, to some extent, adapt to the knowledge requirements that are imposed by the action aspect.” However, these notes proved sufficient, through comments such as "One outcome of the project is a team that works well together" and "A few good people stepped in and did whatever was needed at the time," for generating interesting new concepts to explore over time. Revisiting the notes repeatedly over the course of the research allowed me to detect new patterns in the notes as my own vocabulary and awareness changed.

Collaborative Practice Research

Mathiassen (2000) describes a research approach that incorporates multiple narrower research approaches. This approach, collaborative practice research, is a research style in which the researcher combines action research, experiments, and practice studies to balance the strengths and weaknesses of each (Figure 2.2-1). As Mathiassen describes, "Ideally we want the research process to be tightly connected to practice to get first-hand information and in-depth insight. At the same time, we must structure and manage the research process in ways that produce rigorous and publishable results" (p.134). He continues with the obvious hazard involved: "Unfortunately, these two fundamental criteria do not always point in the same direction." The resolution is to move between the three different research approaches depending on the specific needs and intentions and to obtain benefits from each.

Collaborative practice research requires three things, as its name implies. It must be *collaborative* — the investigator must collaborate with practitioners in both the practices and the research. It must involve *practice* — the investigator must "do" some of the work being studied. It must involve *research* — the investigator must build knowledge using the varied research approaches. The recipients of the research are, as in action research, both the practitioners on the projects and the wider research community.

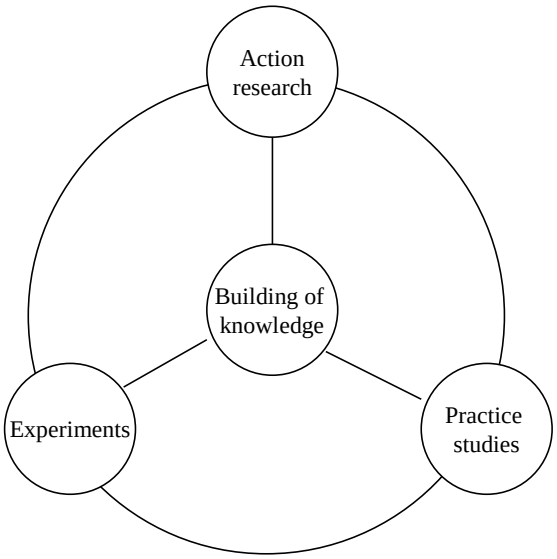

Figure 2.2-1. Collaborative practice research (after Mathiassen 2000).

Collaborative practice research is built from the principles that underlie reflective systems development (Mathiassen 1998). RSD operates as a systems development approach using research and practice as two modes of inquiry. In the practice mode, the person applies particular systems development techniques along with reflection-in-action (Schon 1983) to evolve the information system. In the research mode, the person applies reference disciplines and perspective as collaborative practice research (CPR) to evolve the research base. The two feed each other (Figure 2.2-2): The research base yields interpretive and normative information that is used in constructing the information system; the practice itself creates new systems research.

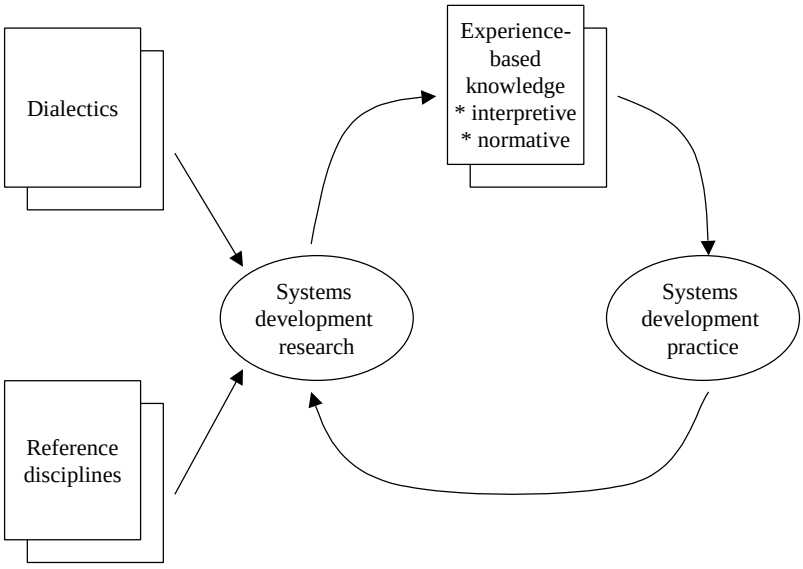

Figure 2.2-2. Reflective systems development (after Mathiassen 1998).

Collaborative practice research is an application of RSD to the research practice itself (Mathiassen, personal communication). The practice being performed is the research practice instead of the system development practice. The research activity of RSD is the research into the research practice. The research practice, as practice, informs the research about that practice, and the research into

research practice produces interpretive and normative knowledge of use in conducting the research practice. This is double-loop learning applied to the research practice.

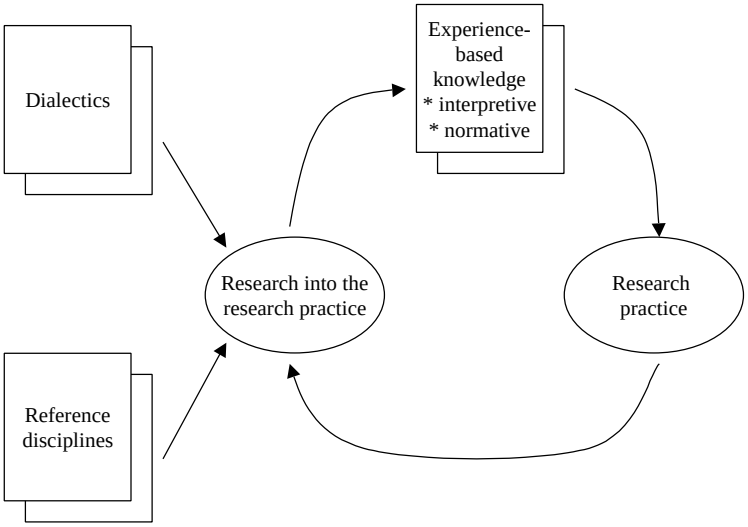

Figure 2.2-3. Collaborative practice research cast into the reflective systems development framework.

It is clear that we can continue a succession of inquiry for many levels. Consider, for a moment, a bank looking to improve its banking operations, viewed from the perspective of new software as an element of the improvement. In this example, there are five levels of investigation (Figure 2.2-4):

(I) The operations of the bank and reflection on how software can improve it

(II) The development of that software and reflection on how that software is developed

(III) The design of the software process and reflection on how that process might be improved

(IV) The research into software process improvement and reflection on research methods;

(V) The research into research methods, which is reflexive

(The bank example and Figure 2.2-4 are by Mathiassen, used with permission.)

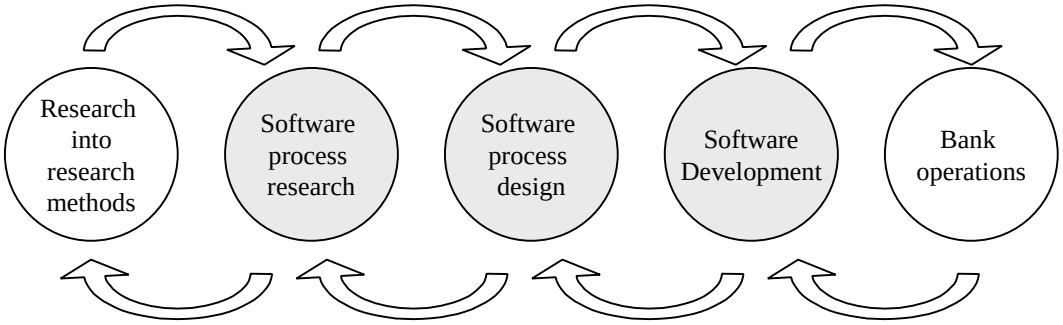

Figure 2.2-4. Five levels of activity and inquiry.

At each level, the practice informs the research, and the research makes new insights or tools available for the practice.

In the current research, I used the middle three of the five levels of activity and inquiry. That is, I do not examine the operation of any particular company, nor do I perform research into research methods. I do examine software development itself and software development methodologies, and I examine my own process for performing my research.

Reflective systems development and collaborative practice research support Argyris and Schon's double-loop learning, if the researcher can stand back and look at the ongoing output of the research. This is not a trivial proceeding, as the researcher must be willing to challenge his or her own prior claims, temporarily abandoning former results in favor of new hypotheses that still need testing. At no time may the researcher become too enamored of the value set currently in operation. The cost is time; the benefit is the testing of a stream of theories about development.

We can see that collaborative practice research, incorporating multiple periods of research activity from the three approaches it incorporates, may take a longer period of time than each of its three components alone. The current research bears that out, proceeding as it did over a ten-year period. The benefit is that the researcher can cross-check his or her views from several perspectives, change views along the way, and test those changes.

Reflective systems development and collaborative practice research were being developed and refined at the same time that I was carrying out the research project in this thesis. It would be improper, therefore, to say that I designed my research to use them. However, reinterpreting my practice in the current language of the field, we can say that I employed reflective systems development in the systems development activities and collaborative practice research in my research activities. I increased the library of reference disciplines in the collaborative practice research with longitudinal process research, experiments, and grounded theory.

Dialectic Investigation

I used dialectic inquiry to examine the research problems from opposing perspectives (McTaggart 1896). Each perspective illuminates aspects of the subject left dark by the other. As the McTaggarts write, "If we examine the process in more detail, we shall find that it advances, not directly, but by moving from side to side, like a ship tacking against an unfavourable wind."

The opposing sides are reframed into a third view that could contain them both. For example, as we have seen in RSD and CPR, the research output is given to the practitioner, and the researcher learns the practice.

I used two dialectical pairs in this research. The first pair uses the perspectives *methodology* and *people*. The methodology perspective focuses on, "What principles and elements underlie methodologies?" The people perspective focuses on, "How do these ideas fit the lives of developers on projects?" The second pair consists of *descriptive* and *prescriptive* perspectives. The descriptive perspective focuses on, "How do we perform systems development?" The prescriptive perspective focuses on, "How can we support systems development?"

The research results were addressed to two communities: *practitioner* and *researcher*. In collaborative practice research, both practitioners and researchers update their respective knowledge bases. Therefore, while other results must be made appropriate for further research inquiry, some results must be made consumable by busy practitioners. An example of the latter type of result is the theoretical paper "Using 'V-W' staging to clarify spiral development" (not included in this thesis). It was presented to the practitioner community as a Practitioner's Report at OOPSLA '97 rather than to the research community. This was done to pass back to practitioners a key theoretical lesson learned, in a way that they could apply directly. That this sort of activity is effective is evidenced by one project manager's comment a few years later: "I honestly divide my software development life into the period before I had heard of the ValidationVee and the period after" (Barnett URL).

Although the publications included in Appendix of this thesis address both sides of each dialectical opposite, one can ascribe to each paper a primary focus in addressing a methodology or people question, being descriptive or prescriptive, and presented to practitioners or researchers (Table 1). This thesis primarily addresses the researcher community, and is primarily descriptive, but integrates the methodology and people sides.

TABLE 1. Papers included in the thesis.

	Methodologies	**People**
Descriptive	**Researcher:** "The impact of object-orientation on application development" **Practitioner:** --	**Researcher:** "The interaction of social issues and software architecture" **Practitioner:** Project Winifred case study from *Surviving OO Projects*
Prescriptive	**Researcher:** "Selecting a project's methodology" **Practitioner:** "Just-in-time methodology construction" "Balancing lightness with sufficiency"	**Researcher:** "Characterizing people as non-linear, first-order components in software development" **Practitioner:** --

It is beyond the range of this thesis text to include all of the papers produced in the research. It is worthwhile, however, to note the way in which the papers use the perspectives. Table 2 lists the publications produced during the period 1993-2002. Each is labeled as to whether its intended reader is primarily a researcher or practitioner, whether it addresses primarily people or methods, and whether it is primarily prescriptive or descriptive. The papers marked in **bold** are those included in the Appendix. Full references to the papers, with online access where available, are given in the References.

TABLE 2. Complete list of publications related to the research.

Year	Publication	Audience	Subject	Form
1993	"**The impact of object-orientation on application development**" (*IBM Systems Journal*)	Research	Methods	Descriptive
1994	"In search of methodology" (*Object Magazine*)	Practitioner	Methods	Descriptive
1995	"Unraveling incremental development" (*Object Magazine*)	Practitioner	Methods	Descriptive
1995	"Using formalized temporal message-flow diagrams," with Citrin, von Kaenel, and Hauser, *Software Practice and Experience*	Researcher	Methods	Descriptive
1995	"Prioritizing forces in software architecture" (*Pattern Languages of Programs 2*)	Researcher	People	Prescriptive
1996	"**The interaction of social issues and software architecture***" (Communications of the ACM*)	Researcher	People	Descriptive
1996	"?(Ad) / ? (Hf): growth of human factors in application development" (*Humans and Technology Technical Report*)	Researcher	People	Descriptive

1997	"Software development as community poetry writing" (*Humans and Technology Technical Report*)	Practitioner	People	Descriptive
1997	"PARTS: precision, accuracy, relevance, tolerance, scale in object design" (*Distributed Object Computing*)	Practitioner	Methods	Descriptive
1997	"Using "V-W" staging to clarify spiral development," OOPSLA'97 Practitioner's Report	Practitioner	Methods	Prescriptive
1998	*Surviving Object-Oriented Projects*, Addison-Wesley, 1998. **(Project Winifred case study from book included in this thesis)**	Both	Both	Both
1998	"Exploring the methodology space" *(Humans and Technology Technical Report*)	Researcher	Methods	Descriptive
2000	"**Selecting a project's methodology***" (IEEE Software*)	Practitioner	Methods	Prescriptive
2000	"**Characterizing people as non-linear, first-order components in software development**" (4th International Multi-Symposium on Informatics)	Researcher	People	People
2000	"**Just-in time-methodology construction**" (2000 International Conference on Extreme Programming and Flexible Processes)	Researcher	Methods	Prescriptive
2000	"**Balancing lightness with sufficiency**" (*Cutter IT Journal*)	Practitioner	Methods	Prescriptive
2000	"The costs and benefits of pair programming," with Williams (2000 International Conference on Extreme Programming and Flexible Processes)	Researcher	Methods	Descriptive
2001	"Agile software development: the business of innovation" (*IEEE Computer*)	Researcher	Methods	Prescriptive
2001	"Agile software development: the people factor" (*IEEE Computer*)	Researcher	Methods	Prescriptive
2002	*Agile Software Development* (Addison-Wesley, 2002).	Practitioner	Both	Both
2002	"Agile software development joins the 'would-be' crowd" (*Cutter IT Journal*)	Researcher	Methods	Prescriptive
2002	This thesis	Researcher	Both	Descriptive

3. Issues and Results Chronologically

Each of the seven papers included in the Appendix added a particular piece to the research results. Here are the three research questions and the end positions we can see the papers drive toward:

Question 1: Do we need yet another software development methodology, or can we expect a convergence and reduction at some point in time?

(End Position: No, there can be no convergence; Yes, we will always need another one. Worse than that, each methodology is a "one-off," and even worse than that, each is actually a transient construction even within a single project.)

Question 2: If convergence, what must be the characteristics of the converged methodology? If no convergence, how can project teams deal with the growing number of methodologies?

(End Position: In the absence of a converged methodology, develop a set of principles the team can use to tell if their methodology is appropriate to their needs and a just-in-time (JIT) methodology-construction technique to serve the project within the project's timeframe. These are part of the research results.)

Question 3: How does the methodology relate to the people on the project?

(End Position: The methodology is a draft formula, which lives in interaction with the specific people and their working environment – which together I refer to as the project "ecosystem." The JIT methodology-construction technique turns out to be uniquely well suited to the problem of shaping the methodology and the ecosystem to fit each other.)

The research results fall into three categories:

Results Category 1: Establish that a methodology is a transient artifact that is created anew on each project and changes within the project.

Results Category 2: Establish that:

- People are a first-order driver of a project's trajectory.
- Other characteristics specific to each project, such as the room layouts and seating distances, also affect project outcome significantly.
- These "ecosystem" details materially affect the methodology.

Results Category 3: Find a way to deal with the first two results. Specifically, describe a technique to construct methodologies quickly and inexpensively enough to be done on the fly and in time to be of value within the life of the project. Analyze the interplay between the ecosystem and the methodology sufficient to inform the project participants on how to make specific moves to improve their situations.

Results Category 1: Methodology as a Transient Construction

The first two papers in the Appendix produce most of the first result.

"The impact of object-orientation on application development" identifies and categorizes key elements of interest in the overall research space, including incremental development and methodologies. It reveals that methodologies must necessarily change as technology changes; that is, there can be no convergence.

"Selecting a project's methodology" makes the point that every project needs its own personalized methodology, based on non-technology characteristics such as team size, geographic separation, project criticality, and project priorities.

These two papers hint that each methodology is a "one-off," but they do not address the transient nature of a methodology. That result emerges from the fifth paper, "Case study from Project Winifred," which describes how the project's methodology changed over time; the sixth paper, "Just-in-time methodology construction," which talks about evolving the methodology to meet the team's changing views; and the seventh paper, "Balancing lightness with sufficiency," which discusses how the balance point moves over time.

Results Category 2: People and Ecosystems as First-Order Considerations

The third, fourth, and fifth papers in the Appendix contribute to the second result.

"The interaction of social issues and software architectures" describes how a set of system architecture decisions were mapped back to issues about managing people. The surprise aspect of the paper is the extent to which social issues drive seemingly technical decisions.

"Characterizing people as non-linear, first-order components in software development" describes how people issues, seating distances, and communication channels affect a project's trajectory. It presents a few methodological principles for organizing project teams.

"Case study from Project Winifred" describes the organization of one project into teams with different working conventions. It also shows the transient nature of a methodology and the interplay between the project's methodology and ecosystem.

Results Category 3: Ecosystem and Methodology Co-Inform Each Other

After initiating the idea that each methodology is a "one-off" construct, the second paper, "Selecting a project's methodology," presents a set of principles that people might use to fit a methodology to their situation.

The fifth paper, "Case study from Project Winifred," outlines the setting of the particular project and then describes an initial use of the JIT methodology technique.

The sixth and seventh papers make explicit the transient nature of methodology, describe the "just-in-time" methodology-construction technique, and extend the discussion of the interplay between ecosystem and methodology:

"Just-in-time methodology construction" presents a technique for a team to build its own, personalized methodology usefully within the timeframe of a single project and to keep that methodology tuned to the changing needs of the project team.

"Balancing lightness with sufficiency" contributes the idea of "barely sufficient" methodology, with its more important counterpart, "just barely insufficient" methodology. Showing examples of just barely insufficient methodologies highlights the tension between not-enough, just-enough, and too-much. The paper discusses how the point "barely sufficient" shifts within a single project. The paper also introduces the cooperative game model for thinking about software development projects, which is intended to help the people on the project make appropriate decisions regarding the degree of lightness to employ at any given moment.

This chapter contains overviews of the papers. In each section, I highlight the key points of the paper and its contribution to the research results. The reliability of the results are discussed in the next chapter, "Consolidated Results and Reflection."

3.1 The Impact of Object-Orientation on Application Development

"The impact of object-orientation on application development" (*IBM Systems Journal*) was an early report that summarized several years of programming activity, project interviews, literature study, and reflection about object-orientation and its place in the overall application context. At the time, I worked in the headquarters of the IBM Consulting Group with people who had already developed an Information Engineering (non-object-based) methodology for IBM service teams. They, and many other people at the time, were not familiar with object technology, and they needed to see OO's place in the overall development context.

The reason for including this paper in the thesis is that it shows the results of the first phase of the research: a very broad investigation into the space of inquiry, categorization of the space, and selection of key topics. Being descriptive, the paper presents the topics in a value-neutral way (not indicating strong preferences).

The paper serves a second purpose now, ten years after it was first written. In those ten years, technology has had time to shift again. Reexamining the paper across that distance, we can see a pattern relating technology shifts to ongoing methodology shifts.

The paper was first published in 1993 and was used in the curriculum of graduate software engineering classes at the State University of New York (SUNY) at Oswego for several years. The *IBM Systems Journal* editors selected it as one of the 20 papers to be reprinted in the journal's 50th year retrospective issue in 1999.

Structure and Contribution of the Paper

> Object-orientation introduces new deliverables, notations, techniques, activities, and tools. Application development consists not only of these items, but also of work segmentation, scheduling, and managing the sharing and evolution of deliverables. This paper breaks application development into three major components: construction, coordination, and evolution, with the topic of reuse receiving extra attention.
>
> Object-oriented (OO) development is characterized by: (1) the encapsulation of process with data in both the application structure and the development methodology, (2) anthropomorphic design, in which objects in the application are assigned "responsibilities" to carry out, (3) modeling the problem domain throughout development, (4) emphasis on design and code reuse with extensibility, and (5) incremental and iterative development.
>
> The effects are far-reaching and have mixed value. Each new mechanism provided by object orientation requires training and judgment of engineering and business trade-offs.

The paper is structured into sections around:

(1) Construction, coordination, evolution, and reuse

(2) Encapsulation, anthropomorphic design, and inheritance

(3) The impact of those topics on application development

Because the paper is structured much as an encyclopedia entry on the topic of object-oriented development, I resist summarizing it on a point-by-point basis. However, the categorization of methodologies is relevant to this thesis.

> An evaluation published in late 1992 included 23 OO development methodologies or methodology fragments for comparison. At least four noticeably different methodologies have appeared since that article went to press, and more are coming. The methodologies fall roughly

> into three camps: those that work from commonly practiced non-OO techniques where possible (e.g., data flow diagram decomposition), those that are based on formal models with existing notations (Petri nets, finite state machines), and those that work from new and uniquely OO techniques (e.g., contracts, responsibility-driven design). Monarchi and Puhr call the three approaches *adaptive, combinative,* and *pure-OO*, terms I adopt here.
>
> The single most common characteristic among the methodologies is the call for incremental and iterative development. . . . Beyond that recommendation, and the need for objects, the methodologies diverge.

The prediction about more methodologies being invented has held true, and far from any convergence in the methodologies, there has been convergence on the opposite idea, that multiple methodologies are necessary.

The paper also highlighted what is still a major split among development groups: what kind of CASE tools to use. "Users of the pure-OO approach seem to have little need for many of the current diagramming techniques and corresponding CASE tools. . . Until quite recently, they worked largely with paper and pencil and a class hierarchy browser, demonstrating, more than anything, the power of those two tools."

In the years since the first publication of the paper, the industry has seen growing power in the tools that support drawing, with "round-trip engineering" to keep the drawings and the code synchronized, yet the preference for paper, pencil, and class browser still holds for many teams. Even within the "agile software development" section of the industry (Cockburn 2002 ASD), the participants are split between those who intend to replace programming with drawing and those who resist drawing entirely.

Reflection on the Paper

Initially, the paper served to instruct me on the separation between the programming technology being employed and the staging and scheduling strategies employed (use of incremental and iterative development), and to highlight to other people that at a very fundamental level, the co-encapsulation of data with function in objects must force the creation of new development methodologies.

From the perspective of ten years of technology evolution since the paper was written, we can observe that the latter observation applies generally. It highlights the effect that new technology has on methodologies. The telling sentence from the paper is the following:

> With disagreement on the deliverables, there can be no agreement on the activities, techniques, tools, or work segmentation.

The implication of this observation is that any technology shift that changes the set of deliverables will necessarily change the methodology. I saw this shift occur again in a website construction methodology that I reviewed in 2001. In creating a corporate web site, someone in the development team needs to create a style guide describing the *look and feel* of a corporate website, mentioning everything from placement of information on the screen, to color tones, to the voice, person, tense, and tone of the prose on each page. Adding this deliverable to the methodology meant that new roles, activities, technique, skills, tools, and work segmentation had to be arranged.

In later papers I discuss the need to vary methodologies based on project priorities. This paper is the only one that discusses the need to vary methodologies based on technology.

3.2 Selecting a Project's Methodology

"Selecting a project's methodology" (*IEEE Software*) was written to put into the research literature a defense of the idea that every project needs its own methodology. As part of that work, the paper also contributed:

? A clarification of the concept of a methodology's "weight"

? A very simple taxonomy of project characteristics that can be used to help select a methodology for the project

? Four principles that relate the weight of a methodology to the project's place in the taxonomy and allow a project to use a lighter methodology

While it should be obvious by this stage in the thesis that each project needs its own methodology, this paper differs from the previous one in that it argues for methodology-per-project not on the basis of individual personalities or technology shifts, as has been done already, but rather on the basis of the variations in projects sizes and priorities.

Structure and Contribution of the Paper

> This [paper] describes a framework for methodology differentiation, principles for methodology selection, and project experiences using these ideas.

The paper is structured in four parts:

? Structure of a "methodology"

? Four principles and two factors for methodology design

? The selection framework

? Application of the framework

Structure of a Methodology

A methodology covers the topics: people, skills, roles, teams, tools, techniques, processes, activities, milestones, work products, standards, quality measures, and team values. The standards may cover notation, tools usage, or the ranges of decisions that can be made.

A methodology's "weight" is broken into two components:

1. The number of elements in the methodology (its "size")
2. The rigor with which the elements must be carried out (its "density")

The idea is that just as the physical weight of an object is the product of its physical size and material density, so, too, the weightiness of methodology is the product of its (conceptual) size and density. The methodology elements can be counted and therefore given a number, but the density is purely subjective and qualitative.

It is worth spending a few words on the subject of the weight metaphor. The phrase "heavy methodology," which occurs in popular use, contains a sensory contradiction between quantitative and qualitative measures.

A stone can be deemed heavy (or not) without knowing its weight. We can, however, use precision instruments to determine that weight numerically and objectively. Even if we establish an objective and numerical value for the stone's weight, though, people still differ on whether a stone is heavy, very heavy, or not really very heavy.

A methodology's weight is similar to a stone's in two ways, and different in one. As with the stone, people deem a methodology heavy or not without knowing its weight numerically. We have no precision instruments to determine a methodology's weight objectively and numerically. Even if we could establish a numerical value, people would differ on whether a methodology is heavy, very heavy, or not really very heavy.

We are unlikely to ever develop objective, numerical values for methodology weight. Although the size of a methodology is relatively easy to compute — simply count the number of roles, activities, reviews, deliverables, standards, and so on — the density is not.

"Density" comes from the analogy with physical weight. It is the rigor demanded by the various elements of the methodology: how tightly the standards must be followed, how detailed they are, how much variation is allowed in different people's work, how serious the consequences of a review are, and so on. These items are fundamentally subjective and qualitatively described. Therefore, we may decide that one methodology's elements are "more rigorous" than another's, but we do that without numerical measures.

What we can expect, therefore, is that a methodology will be judged using subjective, qualitative terms. One methodology, seen in isolation, will be judged heavy or not. Two methodologies will be judged as heavier or lighter in comparison to the other. The terms, "size," "density," and "weight" are useful in predicting and explaining those judgments.

The subject term "density" is useful for other reasons. It makes it clear that when the members of a team decide to use use cases, they have not yet said much about the weight of their methodology. They still have to decide whether to allow multiple formats and writing styles or standardize on just one; whether that format is numbered and follows a template or is just a text paragraph; whether the use cases receive a formal cross-project review or just an OK from a manager or user. Those differences exactly correspond to heavier and lighter methodologies, respectively.

Over the course of the years, people have not responded very well to the word "density," and so now I write about this dimension of methodology using the word "ceremony" instead (Cockburn 2002 ASD). The word "ceremony" has been informally attributed to one of Grady Booch's talks on methodology in the mid-1990s.

Some organizations have written methodologies that cover activities from initial sales calls all the way through system delivery and maintenance. Some methodology descriptions cover only activities within a subset of the development project lifecycle. Both are valid as "methodology" descriptions; they cover differ parts of the total project life by intention. Declaring those intentions is valuable, as it helps people recognize that two different methodologies may look different because they intentionally address different job roles or lifecycle segments.

Principles and Additional Factors

Four principles of methodology design are presented.

Principle 1 recognizes that a larger team needs a larger methodology. (Note: This should be a fairly obvious result, as a large team will necessarily do more communicating and need more coordination. It is stated explicitly to permit the further discussion about the limits of methodologies by team size.)

Principle 2 recognizes that a more critical system — one whose undetected defects will produce more damage — needs more publicly visible correctness in its construction. System criticality is here separated into four categories: loss of comfort, loss of discretionary moneys, loss of essential moneys, loss of life.

Principles 1 and 2 separate two aspects of methodology design: methodology elements to coordinate the team and methodology elements to reduce defects in the final system.

Principle 3 recognizes that relatively small increases in methodology weight add relatively large amounts to the project cost. (Note that these are still qualitative distinctions and values.) There is a small positive feedback loop between number of people and methodology size: Fewer people need fewer internal work products and less coordination, so reducing methodology size permits reducing team size, which may permit further lightening of the methodology, and so on. It works in reverse: Adding more work products reduces the team's productivity, so more people are needed, and so more coordination, and so on. Eventually, the positive feedback loop ends, and so there are two sizes that work for any project — a smallest-possible team size using a lightest-possible methodology and a much larger team size using a heavier methodology.

There is, of course, a catch:

> [F]or a given problem..., you need fewer people if you use a lighter methodology, and more people if you use a heavier methodology. But there is a limit to the size of a problem that a given number of people can solve. That limit is higher for a large team using a heavier methodology than for a small team using a lighter methodology. In other words, as the problem changes in size, different combinations of methodology and project size become optimal.
>
> The difficulty is that no reliable way exists to determine the problem size at a project's start, or the minimum number of people needed to solve it. Even worse, that number varies based on exactly who is on the team.

Principles 1-3 set up the result that, based entirely on problem and team sizes, different sizes of methodologies are optimal.

Principle 4 reiterates from the paper "Characterizing people as first-order, non-linear components in software development" that the most effective form of communication is interactive and face to face, as at a whiteboard. Two people at the whiteboard employ many communications mechanisms simultaneously (proximity, gesture, drawing, vocal inflection, cross-modality timing, real-time question and answer). As their communication moves to phone, email, and paper, they progressively lose access to these mechanisms. The principle does not imply that a few people sitting in a room can develop all software. It does imply that a methodology designer should emphasize small groups and lots of personal contact if productivity and cost are key issues.

Principles 1-4 reveal that methodology weight can be traded off against personal communications. With better communication, we can shed bureaucracy. Conversely, to the extent that the group cannot get frequent and rich personal communication, it must add compensation elements to the methodology.

The paper mentions two other factors affecting appropriateness of a methodology: the project's priorities and the methodology designer's personal background.

The project sponsors may need to have the system built as soon as possible, or with legal traceability in mind, or free of defects, or easy to learn, or something else. That dominant project priority directly affects the correct choice of methodology. In a race to market, many intermediate work products may be dropped; once shipped defects start to cost the company money, the sponsors may shift to prioritizing freedom from defects for the same product. Devices having to pass federal inspection standards may need more work products simply to pass the inspection, and so on.

The methodology designer:

> casts his or her experiences and biases into the methodology in an attempt to capture the invariants across projects. However, the risks the team faces vary with each individual project. Therefore, the final methodology design fits a project as well as the designer's assumptions fit the team members' attitudes and the project's true risk profile.

The Selection Framework

A simple chart shows the result of Principles 1 and 2 (see Figure 3.2-1). It is a grid indicating the number of people needing to be coordinated on the horizontal axis, and the criticality level of the system on the vertical axis. That creates a two-dimensional space in which a project's need for team coordination is separated from its need for correctness. For the sake of being concrete, the figure shows each axis separated into distinct zones, six horizontally and four vertically. The attraction of the resulting grid is that team size and system criticality are relatively objective values that can be observed on any project and used to identify the characteristics of a methodology that would be suitable for the project.

The figure adds a third dimension to indicate that different methodologies would be appropriate for projects having different priorities, even with the same team size and system criticality. The resulting chart presents a simple graphical framework that supports discussions of what methodology to apply to particular projects.

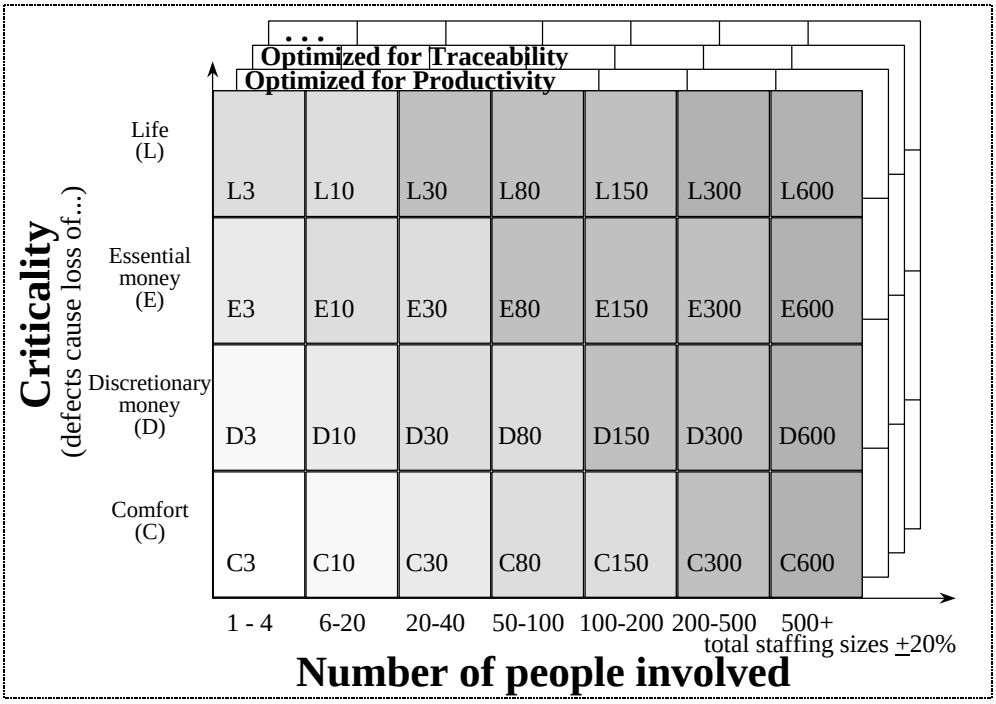

Figure 3.2-1. Methodologies, organized as people x criticality x optimization.

Applying the Principles and Framework

The paper includes case studies showing the application of the principles and the framework. In one case study, the grid was used in the middle of the project to shift the methodology in use. These case studies can be seen in the paper in the Appendix and need not be summarized here.

The paper, having defended the use of a separate methodology for each project, ends with a problem statement:

> The task facing us next is to find ways to tailor a methodology to the idiosyncrasies of any particular project fast enough to get the benefits of the tailoring before the project is over.

Reflection on the Paper

So many people have produced the conclusion "one methodology can't fit all projects" that the most surprising thing to me now is that *IEEE Software* considered the paper worth publishing at all. (Other authors reaching the same conclusion in publications available at the time include Glass (1995), Jones (2000), Mathiassen (1998), Hohmann (1997), Kruchten (1997)).

The fact that I would consider the conclusion novel enough to try to publish it — and that *IEEE Software* would actually consider it novel enough to publish — deserves some attention. Assuming the reviewers and I were reasonably well read in the field, this implies that the previous results were either quite well hidden or there were contradicting views being pushed at the same time. The latter was the situation for me. I had to reestablish the result afresh in order to know which of the voices was on the right track.

The paper produced a slightly novel way of looking at the situation. Some of the other authors established that different sorts of projects need different sorts of methodologies (Mathiassen 1998, Jones 2000), but they did not provide breakdown categories for choosing a methodology. Other authors (Glass 1995, Hohmann 1997, Jones 2000) created too many dimensions of analysis for us to create samples for and too many for the project leader in the field to manipulate with the time and energy available.

This paper created only two primary dimensions: number of people and system criticality, both of which are fairly simple to ascertain. The two dimensions are independent of each other. The number-of-people dimension affects the communications mechanisms in the methodology, the system-criticality dimension affects the verification activities in the methodology. The project priorities affect how those are tuned. A number of people have written to me (personal emails) saying that this two-dimensional split was, for them, the primary contribution of the paper.

In all cases, the differences in project situations and continually changing technologies combine to establish that we are not going to see a convergence of methodologies.

The paper also makes a second contribution: the four methodology design principles. These outline how to achieve a lighter way of working on a project. They also clarify the idea that not every project can work using the lightest methodology (lightness has limits). This idea is repeated and extended in the final paper in this thesis: "Balancing lightness with sufficiency."

In the time since this paper was published, I expanded the list of principles. A set of seven were published in *Agile Software Development* and are described and discussed in the "Consolidated Results and Reflection" section of the thesis.

3.3 The Interaction of Social Issues and Software Architecture

"The interaction of social issues and software architecture" (*Communications of the ACM*) shows how design decisions are implied or derived from decisions about soft issues such as people's skills and difficulties in hiring.

Originally, I had in mind to write a paper describing a project's architectural design decisions using "design patterns" (Gamma 1997). However, during the writing, I kept discovering that decisions had been made more due to social than technical reasons. This came as a surprise to me and opened the door to the idea that human issues are of greater importance to a project's trajectory over time than many people think.

The paper was first published in the results of the annual workshop on patterns, *Pattern Languages of Program Design 2* (Cockburn 1995 PLOPD), and then, in reduced and adjusted form, in the *Communications of the ACM* (Cockburn 1996 CACM).

Structure and Contribution of the Paper

> It has been said that architecture follows organization and organization follows architecture, but little has been written about just how design decisions are affected by organization. Architects do not like being told that their clean designs are the result of accounting for social forces. Project managers do not get to use their knowledge of social issues to influence architecture. Yet it is clear that social issues affect the software architecture in ways that the good architect takes into account.
>
> This article presents a pattern language with case study showing social forces affecting software design decisions. The forces are soft (skill mix, movement of people, team structuring), but the solutions are hard (inheritance hierarchies, access functions, subsystems).

Beyond its primary intent, the paper produced some secondary results. Two of these are the following:

> A principle creates a *force* on the design. It eventually meets another force that limits its use. In general, the counterforce is that too much of a good thing is not a good thing. The principle, ”have a single point of change for each decision,” has the counterforce, “too many isolated points of change make the software difficult to understand and slow to run.” . . .
>
> A designer puts a personal bias into designing a solution, a bias built from a privately evolved set of principles. . . . The usual form of presenting designs and patterns hides the designer’s bias, not letting the next designer know what depends on one of these personally evolved principles. This article is written as interwoven principles, to let you locate your point of disagreement. If you do not like a design decision, you must decide whether you can abide by its driving principle.

As a consequence of these ideas, the paper is written in two parts. The first part deals with principles that would push the architect in different directions. The second part covers a set of specific design decisions made on a project, showing how they follow from the principles and from previously made design decisions. The result is a pattern language (Alexander (1977)).

The six principles are presented in a particular pattern form, showing Intent, Force, Principle, and Counterforce. The six principles are:

Name	Intent	Force	Principle	Counterforce
Variation Behind Interface	Protect the system integrity across changes.	Things change.	Create an interface around predicted points of change.	Too many interfaces means complex, slow software.
Subsystem By Skill	Protect the system against shifting staff skills.	People have differing skills and specialties.	Separate subsystems by staff skill requirements.	Too many subsystems means complex, slow software.
Owner Per Deliverable	Protect the system against ignored and common areas.	People get confused if ownership is unclear.	Ensure there is an owner for each deliverable.	Too many owners and too many deliverables make for a heavy bureaucratic load.
Subclass Per Team	Give teams separate design areas.	Subsystem teams have differing interests and design points.	Where two subsystems collide in one class, assign them to different layers in the class hierarchy.	Excessive inheritance makes the system slower and harder to understand.
User Interface Outside	Protect the system against changed external components.	UI requirements change a lot.	Make the system program-driven; let the user interface be just one of the driving programs.	It is easier on the user if input errors are brought up directly upon entry. Having the user interface outside the application separates input from error detection.
Facade	Protect developers against rework due to an interface change.	Changing an interteam interface is expensive.	Provide a single point of call to volatile interteam interfaces.	Excessive levels of call make the system more complex and slower.

These six principles drove the following nine key design decisions in the case study system. The point for this thesis is not in naming the particular design decisions, but showing how very technical choices were traced back to social issues.

? *Create three subsystems: the infrastructure, the UI, and the application domain.* This was a straightforward application of Subsystem By Skill.

? *Identify the generic parts of problems. Let an expert designer design generic parts. Let the novices design the specific parts.* This derived from Subsystem By Skill and Subclass Per Team.

? *Use three levels of inheritance: Model, ValidatedModel, PersistentModel, with different people responsible for each class. In small designs, merge the first two levels.* This was an application of Owner Per Deliverable and linked to design decision #5, Edits Inside.

? *UI components never call the persistence subsystem directly, but always a domain object, whose superclass provides the single point of evolution.* This followed from Facade and design decision #3.

? *The editing copy of a persistent domain object resides inside the application, not in the user interface*. This followed from User Interface Outside. However, as the paper states, "There is no force deciding where in the application proper it should be. In fact, that design decision is left open [even] at the conclusion of this pattern language (see Shadow Data (DD8))."

? *Entry validation is done in two parts. Keystroke validation is done in the UI, value validation in the application. There must be a way to pass information about the error to the driver program*. The paper says, "This is a freely chosen balance point between the two forces in User Interface Outside."

? *Legislate only accessor methods for persistent instance variables of persistent objects. Other accessors are considered "local matter" to each design group. Persistence accessors belong to the infrastructure, not the domain, and may be regenerated at any time*. The paper says, "This is a balance point for a particular situation, using Variation Behind Interface."

? *Shadow Data — Persistence accessors are being used, but how is persistent instance data handled? Answer: Several possible depending on the variation assumptions*. The point of this design decision was to deliberately leave open a set of design choices, since the optimal design differs, depending on the what is supposed to happen to the persistence server. All designs rely on design decision #3 (Model Hierarchy) and #7 (Persistence Accessors).

? *Allow automatic accessor generation with method tailoring. Accessors are generated automatically. Suddenly the domain programmer discovers the need to be able to touch up the data after the accessor has produced it. How? The answer is to split the public method from the accessor. The accessor gets a predictable modification of the natural name, which the accessor generator knows about*. Example: the instance variable is *city*; the public method is called *city*; the private accessor is called *rawCity* The autogenerator generates the method *rawCity* only if there is already a method called *rawCity* (implying the need for this trick); otherwise it generates the method *city*. This design decision follows from the separation of activities between the infrastructure teams and the domain teams.

Reflection on the Paper

This paper and the one called "Characterizing people as first-order, non-linear components in software development" highlight two ways in which intrinsically *human* issues drive project outcomes.

On the project used in the case study, identifying the social background to these decisions simplified the life of the lead designers on the various teams. Their discussions became a matter of deciding which people and which interests to protect, rather than unreflective personal preference.

3.4 Characterizing People as First-Order, Non-Linear Components in Software Development

"Characterizing people as non-linear, first-order components in software development" (4th International Multi-Symposium on Systemics, Cybernetics, and Informatics) motivated the idea that people are devices with non-trivial operating characteristics, and so the design of methodologies — systems whose components are people — cannot be trivially performed using classical systems design techniques.

The original reason for writing the paper was that I kept hearing from people trained in systems design words to the effect of: "Systems design has been an area of study for decades, and we know how to design systems with recursive structures and feedback loops. There is nothing new about designing software development processes: Merely apply systems design thinking, and it will all fall into place."

That line of reasoning has a flaw. People in the systems design field characterize the components they use in the system: resistors, capacitors, transistors, electromagnetic devices of all types. Knowing the characteristics of these devices, they can combine them with some assurance that the resulting design will behave within some specified tolerances. However, the active devices in software development are *people*, devices with no well-understood operating characteristics. Worse, people have tremendously non-linear operating characteristics, which vary widely across individuals. Systems designers constructing a system design using these complex designers cannot be assured that the system will respond within specifications.

In this paper, therefore, the human individual is considered as a "component" within a system that produces software. The paper argues that the characteristics of the individual components have a very significant effect on the system's overall behavior, and then it presents some of the characteristics of these "components."

Structure and Contribution of the Paper

The paper is structured in two parts. The first part reviews several early research attempts to find ways to positively affect project outcomes and reflects on 23 projects that I participated in or studied:

> What I find striking about these projects is that they show:
>
> - Almost any methodology can be made to work on some project.
> - Any methodology can manage to fail on some project.
> - Heavy processes can be successful.
> - Light processes are more often successful, and more importantly, the people on those projects credit the success to the lightness of the methodology.
>
> "I finally concluded that . . . [p]eople's characteristics are a first-order success driver, not a second-order one. In fact, I have reversed the order, and now consider process factors to be second-order issues.
>
> Most of my experiences can be accounted for from just a few characteristics of people. Applying these on recent projects, I have had much greater success at predicting results and making successful recommendations. I believe the time has come to, formally and officially, put a research emphasis on "what are the characteristics of people that affect software development, and what are their implications on methodology design?

The second part of the paper names thirteen characteristics of people that significantly affect a project's trajectory and elaborates on four of them. Those four are:

? Communicating beings

? Tending to inconsistency

? Good citizenship and looking around

? People vary

Communicating Beings

The most important result of the section on communicating beings is the communication effectiveness curve:

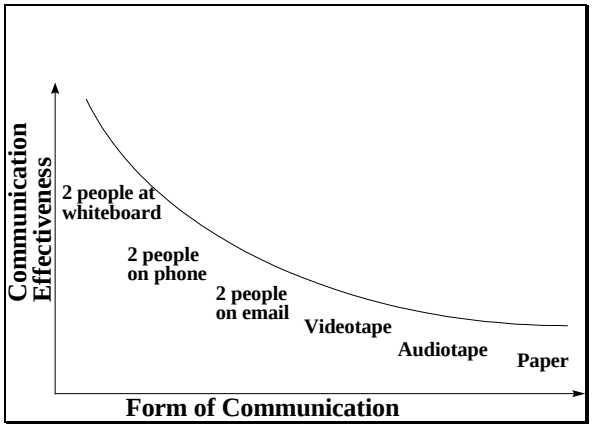

Figure 3.4-1. Communication effectiveness curve.

This curve expresses the following ideas:

When two people are talking face-to-face at a whiteboard (Figure 3.4-2), they have access to: real-time question-and-answer, body gestures, physical proximity cues, drawings, vocal inflection, vocal timing as well as words, and cross-modality timing cues. In addition, the whiteboard adds "stickiness" to sections of their discussion that they wish to refer back to.

On the telephone, the people lose proximity cues, visuals, and cross-modality timing, which makes their communication task more difficult. They still have access to vocal inflection and real-time question-and-answer. When they move from the phone to email, they lose both vocal inflection and real-time question-and-answer, but they still can get questions answered during the session.

On videotape, the viewer once again has access to visuals and cross-modality timing, but has lost any chance to get questions answered or otherwise offer feedback to the speaker on what is needed next in the conversation. Audiotape again loses the visual aspects as well as cross-modality timing, and use of paper again loses vocal inflection.

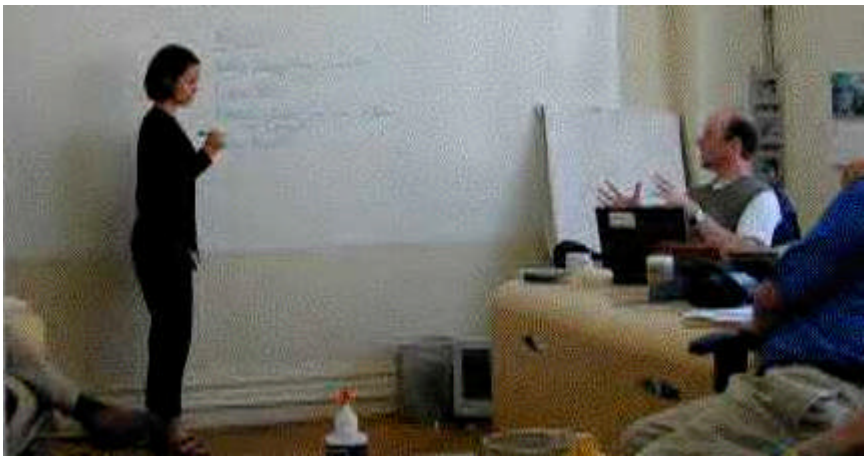

Figure 3.4-2. Two people at a whiteboard. (Courtesy of Evant Corporation)

Part of the value of the curve is the way it explains common recommendations by project leads: "Put all the people into one room" and "Make sure there are whiteboards and coffee corners all over the building."

The curve suggests that valuable archival documentation may be generated at relatively low cost by videotaping sessions in which one developer explains a section of the design to another person. The paper describes one case in which this was done.

The lesson is to take note of the different points on the communication effectiveness curve and to move team communications up the curve as far as possible given the situation at hand.

Tending to Inconsistency

> People are content to be lax about their behavior. One of the two most difficult requests I can think to make of a person is to do something carefully and consistently, day in and day out (the other is to ask them to change their habits).

This "tending to inconsistency" indicates that however effective a particular methodology may be, if it requires discipline — that is, daily consistency in action — the people on the project will probably stop doing it properly. That is, a high-discipline methodology is fragile.

Two high-discipline methodologies are Watts Humphrey's Personal Software Process (Humphrey 1995) and Kent Beck's Extreme Programming (Beck 1999). The paper quotes a CMM Level 5 organization:

> [U]se of the PSP in TIS started to decline as soon as the classes were completed. Soon, none of the engineers who had been instructed in PSP techniques was using them on the job. When asked why, the reason was almost unanimous: "PSP is extremely rigorous, and if no one is asking for my data, it's easier to do it the old way."

The section concludes with noting that this subject is once again affected by the specific individuals involved, "Sometimes simply changing the manager of a group of people can change the consistency of their actions."

Good Citizenship and Looking Around

The success modes of people that counteract the consistency problem are that people are generally interested in being "good citizens," are good at looking around, and take initiative.

A common answer to asking why a project succeeded at all is: "A few good people stepped in and did whatever was needed to get the job done." This idea already surfaces in the case study on Project Winifred collected in the Appendix: "On the good side, two excellent OO programmers were

hired, who almost single-handedly put out the first release, through their arduous work.", "Why was it able to succeed at all? Key individuals at key moments. . . ."

Examining this sentence, we see it contains the three success modes just described. A person is doing his work and "notices something" from just looking around in a normal fashion. He or she has enough pride in his or her work to care about the project's outcome, and then takes initiative to do something about it. Often, that involves stepping out of his or her job assignment.

Attending to these success modes of people suggests it is advantageous to a project to encourage good citizenship and pride-in-work, to let people see around themselves more easily, give them easier communication paths to pass along what they notice, and to take initiative.

People Vary

The point of this section is to remind the reader what should always be obvious, but which people have a tendency to forget when designing methodologies; namely, that different people like to work in different ways.

> Methodologies are largely group coordination rules, and so a recommendation appropriate for one person or group will be rejected by another. What applies to a consensus-minded group might not apply to a culture in which people wait for the boss to make a pronouncement.
>
> Methodologies currently are written to dictate the cultures and work habits of their target organizations. As described above, the group can, and often does, simply reject it.

Other Characteristics

The paper goes on to briefly describe eight other significant characteristics of people that affect project trajectories: learning from apprenticeship, use of "flow," working from examples, multi-sensory thinking, specific personalities, tending to fail conservatively, difficulty in changing habits, and cognitive limits. Detailed discussion of these characteristics is out of the scope of this particular paper, due to length limitations, but included in the book *Agile Software Development* (Cockburn 2002 ASD).

Reflection on the Paper

Relevant to the final point of this thesis, and with the benefit of several more years of thought, I can now make three observations about this paper's results:

1. The paper talks about "people" in the general sense, about what applies to statistically significant sections of the work force. The parts of those observations that are true across large numbers of people apply *only to the methodology as a formula.* Those results are interesting for the methodology designer and project team setting out the broad brush-stroke of the team's methodology and are therefore useful. (This is the intended and obvious result of the paper.) The result of the total research, however, is that the methodology needs to be reshaped to fit the *particulars of the individuals who show up on the project*, not just for statistically based anonymous stereotypes of people. Only in the end section does the paper refer to personalities of individuals.
2. A methodology — an abstract formula based on stereotypes of people — will naturally encounter a mismatch with the real world when it is "dropped" onto particular individuals with specific personalities. This will be the case however valid the methodology's characterization may be at statistical levels. There will be a mismatch between the formula and the practice.
3. Therefore (and this is not present in the above paper but leads toward the main conclusion of the research), there must be an interplay between the methodology-as-written and the method-

ology as it comes to life within a project. This can be seen in the Project Winifred case study description and in the later papers.

Two special points need to be made about the curve in Figure 3.4-1. The first is that the curve matches the results of McCarthy and Monk (1994), who did not present their results in graph form, but discussed the roles of different communication modalities, richness, and effectiveness of different communication channels.

The second point about the curve is that is makes two global assertions about people, and therefore is open to counterexample. It is informative to study those counterexamples.

The first assertion is that the warmer the communication channel, the more effective the communication. However, face-to-face is sometimes too "warm," too emotional. Two people told me that they did better with their team over the phone. They each had a strong personality, and the phone removed some of the intense personal pressure their staff felt. These observations are matched by four separate research results:

- Weisband et al. (1995) found that "status differences in participation may begin with a group's awareness that its members have differing statuses . . . once people categorize others as members of particular groups . . . they typically base impressions of the others on categorization alone."
- Hovenden (2000) describes a meeting in which the senior designer stood up and dominated the whiteboard, effectively cutting off the meeting plans of the person who had called the meeting. As Weisband predicted, the lack of anonymity of the respective speakers created a social ranking for speech opportunities.
- Bordia and Prashant (1997) describe that idea generation, in particular, benefits from the removal of the social ranking.
- Markus (1992) reports on collaborative technology: "The one group that used the technology asynchronously did so in part because of poor social relations. That is, the technology enabled group members to maintain social distance while performing group tasks."

In other words, we will find occasions where cooler, even anonymous, communications mechanisms are better suited to the individuals than warmer ones.

The second assertion is that the sooner the feedback, the better. However, a number of people have written to me that immediate feedback channels (face-to-face, telephone, and instant messaging) disrupt their work. Finding the questions annoying, they block the communication channel, setting up an asynchronous one in its stead ("asynchronous" here means that the answer happens at some arbitrary time later than the question). They close the door, turn the phone to voicemail, and the instant messaging to email. The prevalent use of voicemail attests to how widespread this feeling is.

The curve results, therefore, serve as a guide and first approximation.

3.5 Project Winifred Case Study

The book *Surviving Object-Oriented Projects* collected together my research and experiences up to 1997. Its intent was to provide practicing team leaders with practical advice on project management and methodology and descriptions of projects in action. Although much of the book was prescriptive in nature, the excerpt included in this thesis is a description of a particular project, code-named "Winifred."

Project Winifred was unusual in my case studies because I was a lead consultant on this particular project on a half-time basis almost continuously from its inception to its final delivery. I was privy to most of the decisions made on the project and could visit any work group to view their working habits. When I was actively involved in a situation, I was not able to watch the proceedings dispassionately. However, there were many occasions in which I was not directly involved in the proceedings and could view the scene for some period as an outsider.

As lead consultant, I was able to try out new theories during the project and reflect on what might have caused what. I was able to design (and help evolve) the methodology and to see how the presence of particular people affected the evolving design of the methodology. The way in which we formed and evolved the methodology became the basis and first reference case for the just-in-time methodology development technique described in later papers.

Structure and Contribution of the Paper

The project review is structured in four parts: a brief summary of the type of project, a review of the project's trajectory in four stages, a brief analysis of that history, and an extensive analysis of the project with relation to the 17 themes raised in the book.

Summary

The system was an internal business system, one whose users were all inside the company. The company sold food items through commercial store outlets (its "sellers"). The system had three goals: to extend the profile information kept about the sellers, to track the effectiveness of the company's promotions with its sellers, and to generate invoices to the sellers. The technology was object-oriented client workstations feeding a relational database server, which in turn synchronized with a mainframe system that kept the master copy of the information.

The project took one and a half years, growing from an initial staff of about 15-20 people to a peak staff of about 45 people, of whom about 20 were Smalltalk programmers of various calibers. There were, at peak, four expert Smalltalk programmers, all the rest being new to object-oriented development and Smalltalk. There were only two database designers, and there were about four mainframe programmers. All the people were located in one building, on the same floor, with the exception of the integration test team, which was located one floor below. The sponsoring company had done client-server programming and incremental development before.

The project was a fixed-price, fixed-time outsource contract at about the $15 million level.

The result of the project was a system developed on time and somewhat over budget. The code quality was never wonderful but progressed from awful to tolerable over the course of the project.

History

The project proceeded through four distinct structural phases: a "good start," with experienced people and good support; an near-failure at the first increment; new initiatives with more teams, mentors, and attention to architecture; and a final team structure and new technical infrastructure that worked well.

The project started with supportive sponsors, who allocated money for new workstations, software tools, and consultants and trainers. They arranged for expert users to be available for an hour each week. The project leads were all experienced in their areas — with the notable exception, it turned out, that the OO lead had only ever worked on a three-person, purely OO project before, and was now being asked to lead and coordinate over a dozen people in a joint technology project. This obviously caused some difficulty in the beginning.

The trouble with the first increment came on two fronts: the prime contractor introduced many, and shifting, sets of lead consultants into the project. These people disagreed with each other to the point that the woman running the IT department actually left one meeting in tears after hearing the consultants argue with each other. The other problem was that of the twelve to fifteen Smalltalk programmers writing code, only three had any previous experience. One of those was the team lead already mentioned, who spent his time coordinating rather than designing. The other two found themselves working around and against the other programmers. The result was that the scope of the first increment had to be drastically cut, and the code quality and workstation architecture design were very poor. However, a system was delivered at the first increment timeline. A key success factor was one of the two expert Smalltalk programmers, who "did whatever was necessary" to get the system to meet its minimal goals.

After the team reflected on what had happened on the first increment, there was a change in the lead designers. Three people were introduced: one to lead the design of the workstation infrastructure, one to be lead programmer of the function delivery group, and one to lead the OO modeling effort. The original three expert programmers left, so there was not an increase in experts, although, as it turned out, these three people provided an increase in the expertise available. The novices stayed.

During the second increment, the team structure changed twice. The group briefly tried a straight waterfall, "throw it over the wall" approach, which was quickly abandoned, and then settled back to an ad hoc structure in which people talked to whomever they needed. The result of the second increment was a second delivery, again with what the team considered poor-quality design and code.

The project entered its final stage in the third increment. After reflecting again on what had happened in the previous increment, the three lead designers created a team structure based on the knowledge and abilities of the individual people in the team. They hired another two expert programmers and started redesigning the infrastructure. Morale on the team was stable — mostly due to having delivered twice — if not high. The new team structures allowed the users and sponsors to change the requirements extensively, within the bounds of the original contract, up to eight weeks into the thirteen-week increment cycle. The new team structure stayed in place for the third, fourth, and fifth increments, at which point the project completed.

The new team structure worked, the former novices grew into their jobs, and the project reached what the client and contractor both considered a "happy end," with the final system delivered on time, somewhat over budget (but that was only the contractor's loss), and with the desired functionality.

Analysis

The case study excerpt contains a very brief analysis section, asking only two questions:

? What caused the early problems?

? Why was the project able to succeed at all?

The answer to the first question was: forgetting the lessons of previous decades relevant to staff expertise, staff responsibilities, and communication channels among the staff; consultant ego; absence of architecture; absence of expert staff.

The answer to the second question was: key individuals stepping up at key moments; use of incremental staging and delivery to allow process improvements; unwavering support from the project sponsors; and the team's developing a "habit of delivery" over the course of several increments.

Relation to the Book's Topics

It is not useful here to restate the discussion of how the project mapped to the 17 project success topics raised in the book *Surviving Object-Oriented Projects*. The discussion in the paper excerpt in the Appendix is itself quite short and can be read for itself. The topics themselves need to be mentioned. They are:

Charter: Did the project stay within its original charter?

Suitability: Was the project suited to the technology chosen for it?

Staff: What was the relation of the staff used to that needed?

Technology: What technology was used, and what was the team's experience in it?

Methodology: What methodology was used? Inside this question are questions about five specific sub-topics: What were the roles, techniques, deliverables, standards, and process that were used?

Training: What training was provided to the staff?

Consultants: What was the relation of internal to contracted staff?

Databases: What was the relation between the object technology used in the workstation to that used on the server and mainframe?

Domain Modeling: How was domain modeling handled, stored, evaluated, and evolved?

Use of "Cutting Edge": How much of the technology was cutting edge (i.e., dangerous), and how did that relate to the subject matter and the expertise of the team?

Estimating: How were estimates generated initially, and how were they modified over time?

Planning: What did the planning and the plan look like?

Increments and Iterations: What use was made of increments and iterations?

Risk Reduction: How were risks addressed?

Teams: What team structures were used, and how did they work?

Reuse: How was it attempted and managed, and how did it work?

Productivity: What factors affected staff productivity?

The discussions of each of these questions is contained in the excerpt in the Appendix.

Reflection on the Paper

The Project Winifred case study proved useful in many regards.

At that point in the research, I had collected a set of project stories that hinted at what affected a project's outcome. Key among those were that neither "process" per se, nor advanced CASE tools would have a first-order effect on the outcome. The project bore out those two predictions. More significantly, when things went wrong on the project, the project stories provided an encyclopedia of ideas to draw from. This turned out to be critically useful. Key among those ideas were the use of increments as a base from which to alter the team's working style.

Many people had made comments about how important it is to have a certain number of good people, and how important physical proximity and close communication are to successful project outcome. Those points had not sufficiently penetrated my consciousness before the project. The project provided a test-bed for varying those quantities and seeing the effects firsthand.

We experimented with different teaming structures. One technique we invented has turned out to be useful over the succeeding years. That is to *defocus* rather than sharpen the role assignments on the project. Rather than decompose team roles into smaller and smaller sub-roles and attempt to match those to the project, we deliberately created work groups in which roles were *not* assigned. Rather, we ensured that the team had a sufficient skill base across its individuals to accomplish its assignment and then let the different teams evolve their own role assignments. This idea runs counter to the prevailing process recommendations (as in the Rational Unified Process and the OPEN process framework (Kruchten 1998, Graham 1999) but has showed itself to work well on several other projects.

We evolved the methodology used on the project. This provided the base for the "just-in-time" methodology construction technique described in a later paper. That technique has been applied several times by me, and independently by Jens Coldewey on a project with different technology and in a different industry and country (Highsmith 2002), as well in other settings mentioned in my interview notes but not available in the literature.

I was able to observe the interactions between individual people and note the effects of: a lead programmer who did not like to communicate with his novice staff; a highly abstract OO programmer sloppy with requirements; a very concrete-thinking person working badly with objects but very well with requirements; and the effects of specific people joining and leaving the project. I was able to take notes on how spontaneous meetings formed and on the discussions held while programmers worked through their bug lists.

Project Winifred, like the other projects I participated in, broke some number of my ideas, reinforced others, and helped me construct new ideas.

- I thought that if only I could teach use case writing and OO design based on responsibilities, the designers would use those very simple development techniques in their daily practice. I personally taught those techniques, coached on them, and was present to answer questions, and still, to my knowledge, only one person on one specific occasion actually reached for the responsibility-based design technique. The use cases that were written were only written to the first approximation of what I taught. In other words, the people didn't practice the techniques they were taught. (While this was a big surprise to me, it matches the experiences of Mathiassen (1998).)
- The project reinforced the importance of physical proximity and the value of rapid communication, regenerating the results of Allen from the 1980s (Allen 1984) and matching Olson and Olson's (2000) review of ten years of field and laboratory investigations.
- I hypothesized that people would enjoy doing every part of development, from requirements, through design to coding, to database design and test. This experiment lasted just a few weeks before the team requested a change.
- After the failed experiment just mentioned, we constructed the idea of cross-functional teams (Holistic Diversity, in (Cockburn 1998 SOOP)) in which roles were not specifically assigned but selected by the team members themselves. This was successful, and it has been validated on different projects and by different authors (Harrison (1995), McCarthy (1995)).
- It reinforced the idea of increments being critical to a project's likelihood of success, that each incremental delivery provides, in particular, a time to pause and reflect on what is working and what needs changing.

I encountered new ideas in the literature during and after the project. Notable among these were two books:

? *Situated Learning: Legitimate Peripheral Participation* (Lave 1991) articulated the need for line-of-sight in apprenticeship learning, a practice I could immediately spot was missing in Project Winifred. I later saw this practice on Extreme Programming projects (Beck 1999). The idea eventually grew into the Expert In Earshot project management pattern (Cockburn 2000 LW).

? *The Goal* helped me understand some basics of modern process theory, particularly the significant way in which a "bottleneck" station affects the overall design of a plant. The surprise is that schedule time can be *improved* by running non-bottleneck stations at *lower efficiencies* (Goldratt 1992). We employed this principle on Project Winifred, but it took another five years before I was able to properly place it as a methodology design principle (see Principle 7 in *Agile Software Development* (Cockburn 2002 ASD)).

As the above lists show, on some topics, my practice was ahead of my theory — we did things that clearly worked well, but we didn't understand why they should work. On other topics, I had theory in place and used it. And finally, on some topics, it did not even register that we were doing certain things at all; I couldn't relate them to theory because it was not in my consciousness that we were doing them. Only on reviewing my notes some years later did I notice that I now had a label for what we had done, and then I could look to see how that correlated with actions on other projects and theory from the literature.

3.6 Just-in-Time Methodology Construction

"Just-in-time methodology construction" (2000 International Conference on Extreme Programming and Flexible Processes) addresses the question posed at the end of the paper "Selecting a project's methodology": "The task facing us next is to find ways to tailor a methodology to the idiosyncrasies of any particular project fast enough to get the benefits of the tailoring before the project is over."

Structure and Contribution of the Paper

> Projects and teams differ, so one methodology won't fit all of them. People vary and tend to inconsistency, but they are good at communicating, looking around, and often take initiative to do whatever is needed. Is it possible to build methodologies that are habitable and effective, and accommodate these two statements?

The paper is structured to present:

- The source of the technique
- The technique itself
- Basing a methodology around the characteristics of people
- A specific family of self-adapting methodologies

The Source of the Technique

The technique was used in Project Winifred. I later found, through interviews, that the IBM Consulting Group in England also tuned its methodology at the start of each project, a government contractor did approximately the same with his group of programmers in Florida, and that Siemens in Munich also included a process-tuning workshop at the start of its projects.

Only in Project Winifred, though, did we deliberately evolve the methodology during the course of the project. That experience formed the basis for the technique.

The Technique Itself

The technique proceeds in five stages.

1. Before the project

Before the project, the team interviews people from previous projects in the same organization to discover what worked well and what didn't work, what to strive to keep, and what to try to avoid. Through these interviews, they deduce the strengths and weaknesses of the organization, things to employ to advantage, and things to work against or around.

2. At project start

At the start of the project, the team constructs a starter methodology, using the results of their interviews and bearing in mind the characteristics of the project and the people on the team. They may spend two to five days on this assignment.

3. During the first increment

Around the middle of the first increment, they get together for a few hours to decide, "Are we going to make it, working the way we are working?" They might choose to restart the project, tune their way of working, or cut the scope of their work. Typically, scope is cut. Only in worst cases are major changes done to the methodology at this point, since it takes time to rebuild people's working habits. The point of this check is to detect possible catastrophe, and to get the team used to talking about their way of working.

4. After each increment

After each increment, the team gathers for a few hours or a day to reflect on what they did well or badly, what they should keep or change, and what their priorities should be for the next increment.

5. During subsequent increments

Approximately at the middle of each subsequent increment, the team gets together for a few hours to recheck their way of working and decide if some minor changes need to be made. After the first two increments have been delivered successfully, this can be a good time to experiment with new standards and techniques. The point of this check is to discover items that had been forgotten at the previous end-of-increment discussion and to give the team a chance to back out of failing experiments.

> The result of these twice-per-increment reviews is that the team is able to take into account their most recent experiences. They produce, during the execution of the project, a methodology specifically tailored to their situation.

Basing a Methodology around the Characteristics of People

The paper considers certain characteristics of people that affect the design of the team's starter methodology.

Based on the idea that lighter methodologies are faster, by using face-to-face communications, the team will identify how much of the project communications can be carried in informal channels and how much has to be committed to writing or other communications media.

For larger projects, based on the idea of people being good at communicating and looking around, the paper suggests that instead of trying to keep a detailed task assignment and dependency sheet up to date, the project teams meet weekly and update their cross-team dependencies in a simple table format. This table captures information about what each team needs in the way of decisions, designs, documentation, or code from each other team. The table holds in place the discussions that are supposed to be going on between the team leaders.

Based on the idea that people are variable and tend to inconsistency, the teams decide where to create mechanisms to hold strict behavioral standards in place and where to permit wider variation in people's work habits and outputs. For the second strategy to work, attention must be given to pride-in-work, team morale, and citizenship issues.

A Specific Family of Self-Adapting Methodologies

The final section of the paper describes the structure of a new methodology family, which I call Crystal. Crystal gathers together the recommendations of this research into a related structure. Crystal is not one methodology, nor a framework for methodologies, but is a family of methodologies related by adherence to common philosophy and principles.

> All Crystal methodologies are founded on a common value, "strong on communication, light on work products." Methodological elements can be reduced to the extent that running software is delivered more frequently and interpersonal communication channels are richer. These are choices that can be made and altered on any project.

To reduce the number of methodologies needed in the family, Crystal is indexed primarily by one variable — team size. The variations in criticality and project priorities are handled in the methodology-tuning workshop. Team sizes are color-coded: clear for six or fewer people, yellow for up to 20 people, orange for up to 40 people, red for up to 80 people, and so on. The darker colors indicate the need for more sub-team structures and more coordination elements in the methodology.

The paper describes two members of the Crystal family: Crystal Orange and Crystal Clear. I do not summarize those here, since they are presented in the included paper.

The conclusion of the paper is that while no single methodology can satisfy all projects, it is possible to have a family of methodologies related by a common philosophy, and that a project's specific, tailored methodology can economically be constructed at the start of the project and evolved during the course of the project.

Reflection on the Paper

The construction and testing of a technique to build a methodology within the economic timeframe of a project is possibly the most novel and significant result of the research. Without such a technique, we only know that no single methodology will suffice and that project teams need methodologies to coordinate their work. Having this technique in place allows a team to deal with the transient and personal nature of a methodology and permits the methodology to interact with the ecosystem.

The idea of using a "change case" to tune a methodology to a project has become part of the Rational Unified Process (Kruchten 1997). This technique typically takes weeks or months to perform, however, and is done only once, at the start of the project.

The valuable result in this paper is its strategy for tuning the methodology quickly enough (a few days at the beginning, a few hours each month) that the overhead of the tuning is small compared to the cost of using a standard methodology and living with the mismatch.

The notion of periodic methodology-tuning reflection workshops suggests that the team use a two-step cycle of behavior on projects: *self-programming* alternating with *performing*.

1. In the first step, the people construct rules and policies for their upcoming interactions, deciding on work habits, communication standards, project policies, and conventions.
2. In the second, they perform their jobs in the normal hustle and bustle of project life — probably getting too caught up in their work to reflect on what or how they are doing.
3. They schedule periods of reflection at regular intervals, in which to deliberately adjust their working conventions.

Cycling between self-programming and performing gives the people a chance to improve their way of working with each other, turning a generic methodology into their own. For people not trained in reflective practice, it provides a simple, enforceable mechanism for reflecting periodically and working "heads down" for the rest of the time.

3.7 Balancing Lightness with Sufficiency

"Balancing lightness with sufficiency" (*Cutter IT Journal*) addresses the general topic of light methodologies, specifically what it means for a methodology to be too light, and what one should do upon discovering that. It also introduces the cooperative game model for building software development strategies, and explicitly recognizes the transient nature of methodology. In doing so, it highlights interaction between the methodology and the ecosystem.

Structure and Contribution of the Paper

The paper recaps the situation set up by the previous papers, specifically that each methodology is a "one-off" construct and there is a technique to tune and evolve the methodology within a project's timeframe.

> What is not clear is where to go next. If concrete advice can't be given across projects, how are we to find meaningful methodological advice for our projects?
>
> We have to find an explanatory model that advises us as to how to make good choices. . . . [specifically] using a new explanatory cost model of software development to get meaningful methodological advice in real time.

The paper is structured in four sections:

- Recap of previous results
- The Cooperative Game
- Barely Sufficient
- Sweet Spots

In this thesis, I need not recap the first section. The paper sets up the running question for people on the project: "What is it that we choose to add or drop, and how do make sensible tradeoffs on the fly?"

The Cooperative Game

The need for making trade-offs on the fly is filled by a different model of software development, the idea of development process as a cooperative game.

> [W]e actually don't care, past a certain point, whether the models are complete, whether they correctly match the "real" world (whatever that is), and whether they are up to date with the current version of the code. Attempting to make them complete, correct, and current past a certain point is a waste of money. As one experienced programmer told me, 'I often feel I am doing something good when I start drawing these models, but after a while I feel I have somehow passed the point of diminishing returns, and my work becomes wasteful. I just don't know when I have passed that point of diminishing returns, and I haven't heard anyone talk about it.'
>
> To understand and locate this point of diminishing return, we need to shift away from thinking of software development as an engineering activity and think of it as a resource-limited, cooperative game of invention and communication. . . .
>
> [T]here is a primary and a secondary goal. The primary goal is to deliver the system. The secondary goal is to set up for the next game, which is to extend or maintain the system, or to building a neighboring system. Failing at the primary goal makes the secondary goal irrelevant. However, succeeding at the first and failing at the second is also sad, since failing at the secondary goal interferes with success of the next game.
>
> Thinking of software development as a cooperative game directs us to more fruitful questions, particularly the deployment of our scant people, time, and money. We are led to investigate better techniques for inventing and communicating, ways to get the same net effect consum-

> ing fewer resources. We are led to ask about whether any particular piece of work is sufficient to its task of invention or communication.
>
> We are led to the idea that a model need not be complete, correct, or current to be sufficiently useful. This notion allows us to explain the success of many projects that succeeded despite their "obviously" incomplete documents and sloppy processes. They succeeded exactly because the people made good choices in stopping work on certain communications as soon as they reached sufficiency and before diminishing returns set in. They made the paperwork adequate; they didn't polish it. "Adequate" is a great condition for a communication device to be in if the team is in a race for an end goal and short on resources. . . .
>
> Let us not forget that there will be other people coming after this design team who will indeed need more design documentation, but let us run that as a parallel and resource-competing thread of the project, instead of forcing it into the linear path of the project's development process.
>
> Let us be as inventive as we can be about ways to reach the two goals adequately, circumventing the impracticalities of being perfect. . . . Let us, at the same time, be careful to make it just rigorous enough that the communication actually is sufficient.

This cooperative game idea has become the foundation for the methodologies I design, which I call the "Crystal" family of methodologies.

Barely Sufficient

Some groups use a methodology that is too light, just as others use one that is too heavy to fit the needs of their "game." This section of the paper provides two examples of being "just barely insufficient" in the methodology. This point, "just barely insufficient," is valuable to find, since it informs us about the lower bounds of the light methodologies.

The two examples are taken from Extreme Programming projects, which were otherwise well run, but at some point needed more documentation or coordination than XP provided. Those two examples can be seen in the paper in the Appendix.

> The term *barely sufficient methodology* indicates that it is an ideal and thin line, and that it varies over time even within one project. The project that runs a barely sufficient methodology will be overly sufficient in some places at various times and insufficient in other places at various times. They will, however, be running about as efficiently as they possibly can, which is what is usually critical to the project.

Sweet Spots

"Sufficiency" is sensitive to the project particulars. Any ideal, ultimately "light" methodology will not fit many projects, as we know. A "sweet spot" is a circumstance on a project in which some ultralight methodology element can be used without being insufficient. This section of the paper names five such sweet spots, the ultralight methodology element that can be applied in each circumstance, and the changes that need to be made as the project circumstances lie further from the sweet spot.

The five sweet spots are:

Two to eight people in one room. With so few people and such close proximity, simple discussions and whiteboards can be sufficient to hold the project plan and system design in place. Lying further from this sweet spot implies that the team must employ more written documentation of plans and designs in order to stay in synch. A case is made that very high-speed communication technology can move a distributed team closer to this sweet spot.

Onsite customers. With an expert customer onsite, the team can get questions answered quickly and efficiently using face-to-face communication. Lying further from this sweet spot implies that

the team will expend more cost to get similar information. Mechanisms that can be used to deal with the distance from the sweet spot include weekly meetings with users, in-depth studies of users, surveys, and friendly test groups.

Short increments. Short increments provide for rapid feedback on both the process being used and the system being developed. There is an overhead associated with starting and ending an increment, however, so that increments cannot be made arbitrarily short. Moving away from the sweet spot of one to three months adds hazard to the project in terms of its likelihood of finishing successfully or delivering a useful system. When the user community simply cannot absorb new increments each three months, the team can deploy the system to a single, friendly user, in order to get some of the feedback benefits of short increments.

Fully automated regression tests. With fully automated regression tests, the team can alter and update their design as frequently as they wish and immediately know what defects they just introduced. Missing that sweet spot means that changing the existing code base becomes hazardous, since the team cannot quickly tell what they just broke. Automated regression tests can usually be made for parts of the system, if it can't be made for the whole system.

Experienced developers. Experienced developers move much faster than inexperienced ones. Missing this sweet spot means that the experienced team members must spend valuable time training the new people. Strategies for dealing with the presence of newcomers include putting them into their own learning team and hiring mentors to train the more novice people.

Reflection on the Paper

This paper contributes three things to the evolving research results.

? It makes clear through the project studies that it really is possible to be "too light," however attractive light methodologies may be.

? It presents a way of thinking about tradeoffs in shifting between lighter and heavier ways of working during a project.

? It directly addresses the worry that light methodologies cannot be used everywhere. By naming five "sweet spots," it highlights the multiple dimensions of movement in project strategies, that a project might lie in one or more but not all of the sweet spots, and particular costs and strategies associated with lying further from those sweet spots.

In a context of inquiry larger than this thesis, it is relevant to note that those sweet spots include tool selection as well as methodology design and individuals.

4. Consolidation and Reflection

The questions under examination are:

> **Question 1:** Do we need yet another software development methodology, or can we expect a convergence and reduction at some point in time?
>
> **Question 2:** If convergence, what must be the characteristics of the converged methodology? If no convergence, how can project teams deal with the growing number of methodologies?
>
> **Question 3:** How does the methodology relate to the people on the project?

The answers to those questions appear directly from the papers and the discussion of the papers in the previous chapter. In short form, they are:

> **Answer 1:** No, there can be no convergence; yes, we will always need another one. There are, however, principles of methodology design that can be applied.
>
> **Answer 2:** Develop and evolve a project-specific methodology within the project, in time to serve the project, paying attention to the methodology design principles. A technique for doing this was presented.
>
> **Answer 3:** The methodology is a draft formula, which lives in interaction with the specific people and their working environment.

In this chapter, I review those results separately and as a whole and consider what else can be said about the situation.

4.1 Answering the Questions

Answer 1: An Ever-Increasing Number of Methodologies

The paper "Selecting a project's methodology" describes characteristics found in different projects. That paper highlighted two particular dimensions: team size and system criticality, adjusted according to project priorities. Other researchers have categorized projects along different dimensions, including technology in use, problem domain, and staff expertise (Glass 1995, Hohmann 1997, Vessey 1998, Jones 2000). Each author selected characteristics that are fairly objective and can be ascertained by external or internal observers.

The different taxonomies lead to a similar end result: Many methodologies will be developed to handle the different project types. My paper names four categories of criticality, six of team size, and six to eight dominating project priorities, requiring 4 x 6 x 6 = 144 methodologies. If we recognize half a dozen different domain types, technology types, and staff experience mixtures, the number is in the thousands. Jones writes (2000, pp. 12-16):

> When performing assessment and benchmark studies, it is useful to be able to place a software project unambiguously among the universe of all possible kinds of software projects. . . . We developed a hierarchical classification method based on project nature, scope, class, and type. . . . The combinations of nature, scope, class, and type indicate that the software industry produces no less than 37,400 different forms of software. . . . Each of these 37,400 forms tends to have characteristic profiles of development practices, programming languages, and productivity and quality levels.

The situation is actually much more dynamic than that. From the paper "The impact of object-orientation on application development," we see that shifts in technology create a need for not just different methodologies, but continually new ones. As software technology changes, formerly appropriate methodologies will become inappropriate, and new ones will need to be invented.

This happened twice during the research period 1991-2001. The first shift was from procedural to object-oriented systems. The second was from systems with captive in-house users to web-based systems exposed to the public at large.

The growing number of methodologies stems neither from accident nor poor foresight. It is inevitable and inescapable. We can expect to see methodologies continue to change and grow in number.

Answer 2: Dealing with Multiple Methodologies

Several approaches been put forward to deal with the multiplicity of methodologies:

? Create expert systems to generate the methodology needed for different projects.

This was very effective during the early 1990s, when most commercial projects used structured analysis, structured programming, and relational databases. When the technology shifted, however, there was necessarily a period in which the expert system did not have the information on the new techniques to do its job. During that period, the companies had to invent new methodologies during live projects.

? Create a "kit" for methodology construction, consisting of different work product templates, techniques, and process paths (Graham 1997, James Martin, IBM, UP). The idea is that the project team assembles the methodology they need from the parts in the kit.

Assembling a methodology from such a kit takes a long time and is likely to be error-prone, with the result not fitting the project. I discuss how to get out of this dilemma shortly.

? Rational Corporation has produced a large sample methodology, the Rational Unified Process, along with some guidance for tuning it to a project through "development cases" (Kruchten 1999, Rational URL). The people work though the details of their development situation and identify what should be changed from the large sample provided to get the one needed for the project.

? The fourth approach is the one described in "Just-in-time methodology construction" and included in the methodology family called Crystal. At the start of the project, the team performs some interviews to discover the characteristics of the people and the organization. They select any base methodology they are familiar with, preferably one fairly appropriate to their project's characteristics. They then modify the rules of the methodology according to the information from their interviews, producing a "starter" methodology for the project. This period should take on the order of two to five days, a time period intended to fit well within any project's time budget. Every one to three months, they meet for a few hours to a day to discuss the conventions and rules they are operating by and adjust them to better fit their needs.

Each approach deals with the need for multiple methodologies. In the presence of changing technologies, the first three involve a time lag, during which new-technology projects are run with experimental techniques before the lessons learned from those projects can be placed into the master repository.

The first three approaches set the methodology at the start of the project and then use it unchanged over the duration of the project. Only the fourth includes a mechanism to change the methodology during the project, as mistakes and mismatches are discovered.

Answer 3: Methodologies Meet People

The paper "Characterizing people as first-order, non-linear components in software development" highlighted that, in practice, the characteristics of people have more effect on a project's trajectory than does the process or methodology used. Jones (2000) captured similar results: compared to a productivity effect of 139% caused by the factors of team experience and team placement, the process had an effect of only 35%. Boehm (1987) wrote, "Variations among people account for the biggest differences in software productivity." Mathiassen (1998) summarized one of the lessons from the MARS project as "Systems methods were seldom, or only partially, followed by experienced practitioners."

This raises a reasonable challenge: The answer might be to abandon the discussion of methodology altogether. Should we care about methodology at all?

The principles and graphs from "Selecting a project's methodology" provide an answer to the challenge: If the team consists of only two to three people, then the methodology as a distinct topic may be inconsequential. But as the team size grows, coordination issues become significant, and therefore more consideration must be given to the methodology.

Wayne Stevens of the IBM Consulting Group illustrated with the following anecdote:

> If only a few drivers cross a large parking lot at night, when there are no other cars, it doesn't matter where they drive. The few drivers can arrange to avoid each other. As the number of cars moving through the parking lot increases, eventually it does matter. With increasing number of drivers, it becomes more important that they establish and follow conventions about parking, driving lanes, and stop signs.

Building from this anecdote, we can think of a methodology as "the working conventions that the team chooses to follow." Two or three people don't need many conventions between them; forty or a hundred people need more. While methodology may not matter with very small teams, it becomes a non-negligible issue with larger teams.

The big issue is that people differ from each other and from the generic role descriptions given in a methodology text. This was described in "Characterizing people as first-order, non-linear components in software development" and the Project Winifred case study. Particularly significant is the fact that the people who show up on the project may not have personal characteristics that properly match the characteristics needed in the job role they play. They may not get along with each other on a personal level. This means that the project team needs to take into account the very specific characteristics of the people who show up on the team.

In fact, there is a larger effect in play. Not just the characteristics of the people, but also the characteristics of the building itself and the seating arrangement have a noticeable effect on the appropriateness of the methodology and on the team's effectiveness (Allen 1990, Cockburn 2002 ASD, Jones 2000, Osborn 2002). Mathiassen (1998) goes the furthest in his description:

> Each environment had distinct characteristics that played an important role for the success and failure of projects. . . . Systems development methods and tools play important roles in shaping and improving practices, but their impact is limited. Systems developers must, in addition to mastering a repertoire of general methods and tools, know how to cope with the specific environment in which they work.

In the language generated by the current research, the project team and the work setting together form a sort of "ecosystem." In this ecosystem, we find analogs to different species (the different roles), predators (people with dominant personalities), cliffs (stair wells and elevators), travel paths, watering holes, and so on. One could get fanciful about the analogies, but the point remains that each characteristic of the specific team working in its specific setting affects the behavior of the team in action. Weak- or strong-minded people join and leave the project; people change their seating arrangements. Any of these things might alter the conventions that the project team should and will agree to follow, making each methodology not only a "one-off," but a transient construct.

It is useful to have the term *ecosystem* to capture the characteristics of the team operating in its setting. It allows us to explicitly discuss the interaction of the methodology and the situation. I expand on this in the next section.

4.2 Consolidation and Reliability of Results

In this section, I consolidate the discussion that has been growing through the thesis, concerning the transient nature of a methodology and its interaction with the project ecosystem. I then discuss the reliability of the results and present some recent extensions I have found since the latest publication included in this thesis. Finally, I relate these results to the closest other research results, those of Mathiassen (1998).

Consolidation: Methodology Meets Ecosystem

A *methodology* is a formula, specified without reference to individual people. The *ecosystem* consists of individuals and an environment that may not fit the formula. Each needs to be informed by the other, and probably both need to be changed.

Of the four approaches to handling multiple methodologies described in the answer to Question 2, the "just-in-time" approach provides an answer to just how they inform each other.

? First, the team tunes the starter methodology to the characteristics of the project, the technology in use, and the specifics of the ecosystem. The ecosystem specifics are discovered during the interviewing period and during the methodology-tuning workshop. Those specifics are incorporated during the tuning workshop. The resultant methodology is the initial set of conventions the team agrees to follow.

? Periodically, the team reflects on their work and reviews those conventions, discussing which to drop, which to change, what to add. In this way, they form a unique, project- and ecosystem-specific methodology that tracks their changing needs.

The trick, of course, is doing this tuning and revision in a short enough time that the revision work pays for itself within the project timeframe.

The just-in-time tailoring technique might be applied with any of the other three approaches to multiple methodologies described in the answer to Question 2. Whether one is using a methodology produced by an expert system, or a methodology from a kit, or a methodology reduced from a large sample, the team can and should reflect periodically and revise their methodology accordingly.

Extensions to the Results

There are three extensions to the above results since the publication of the papers: the possibility of using different methodologies within different subsets of the project team, other times at which to call for a methodology-tuning workshop, and additional principles informing methodology design.

Using different methodologies within different subsets of the project team

Treating people's individual personalities as significant input to the methodology produces the consequence that possibly different team members should use different rules within certain bounded work areas.

One example of this is formalized in *Surviving Object-Oriented Projects* (Cockburn 1998 SOOP) as the strategy called "Holistic Diversity." The case study from Project Winifred describes the creation of "functionality" teams, whose work was bounded between the user expert and the test department. Within that range, each team was staffed with sufficient people to cover the skills needed, each team was given accountability as a whole to deliver a set of application function, and each team was allowed to arrange its people in whatever role mixture suited that team. The alloca-

tion of user interface design, business analysis, database design, and application programming varied across the teams. Each sub-team created its own localized tailoring of the methodology.

Another, more recent example from a South African company used a different partitioning.

> Two of the business managers had very different personalities and work styles, although both were hired for their business acumen and business contacts, and both of their job assignments entailed providing explicit direction to the programmers as to what should be built. The woman, a careful and methodical worker, was able to carry out this assignment. The man, who excelled at the marketing, sales, and contact aspect of his work, did not have the patience or inclination to create the detailed work products needed by the programmers.
>
> We were unable to create a role breakdown for the company that would describe how to fit both of these people into a corporate methodology description. Instead, we were able to identify a larger bounded work area in which two different methodology variants could be used. The work area was bounded on the one side at the entry to the business, the external market. It was bounded on the other at the entry to the software development group. The work to be done between those two interfaces consisted of analyzing the market to suggest appropriate initiatives and then framing those initiatives in sufficient detail that the programmers could do their work. The woman was able to do both of these tasks; the man not.
>
> The woman, able to do both of these tasks, was to be provided a junior co-worker able to work with her on the end-to-end development from business need to use cases and data description. The man was to be provided a co-worker able to develop the detailed use cases and data descriptions from his more loosely worded initiatives.
>
> In a formal situation, we would have created four job titles and role descriptions for these people, senior business manager and junior business manager for the first pair, and business market manager and business analyst for the second pair. The company's methodology then would specify the behavior of each pair and their interactions with the two outer interfaces (the market and the software development group). In our particular case, we decided not to go through all that methodology construction, but to keep the conventions fluid. With such a small company, each person knew what he or she needed to do.

Other times at which to call for a methodology-tuning workshop

The second extension to the results has to do with when to call for methodology retuning. The basic technique outlined above and in "Just-in-time methodology construction" calls for the tuning to be done before each increment (and checked once during the increment). However, the characteristics of the ecosystem may change abruptly at any time, with the arrival or departure of a key individual, a move to new office space, or a sudden change in business priorities or implementation technology.

Therefore, the methodology should be reviewed whenever there is a significant change in the ecosystem. An example of this happening is given in the story about the "NB Banking Project" in the "Just-in-time methodology construction" paper, although that paper does not specifically refer to ecosystem changes.

Seven principles instead of four

The paper "Selecting a project's methodology" was written in 1999. At that time, I only named four principles:

Principle 1. *A larger methodology is needed when more people are involved*

Principle 2. *More publicly visible correctness (greater density) is called for on a project with greater criticality, where more damage can result from an undetected defect.*

Principle 3. *"Weight is cost": a relatively small increase in methodology size or specific density adds a relatively large amount to the cost of the project.*

Principle 4. *The most effective form of communication (for the purpose of transmitting ideas), is interactive, face-to-face.*

As I continued to evaluate what I was *doing* on projects against those principles, I found three others were needed to explain effective methodology designs. The second of them comes from Highsmith (2002, but originally from personal communication).

Principle 5. *Increasing feedback and communication reduces the need for intermediate deliverables.*

Principle 6. *Discipline, skills, and understanding counter process, formality, and documentation.*

Principle 7. *Efficiency is expendable in non-bottleneck activities.*

The fifth principle underlies the design of the Crystal family of methodologies (Cockburn 2002 ASD). The sixth is common across all of the "agile" methodologies (Highsmith 2002). The seventh I had found on Project Winifred (Cockburn 1998 SOOP) but not named until just before I used it deliberately on the eBucks.com project (Cockburn 2002 ASD).

Reliability of the Results

Research in a field moves forward in the conversation between discoursants. One person produces a result and makes a claim; another extends, bounds, and/or counters that result with another. A research "result" is a proposition put forward in that conversation.

The results of this research are propositions put forward in the long-running conversation about what makes up software development and how to improve it. The conversation held in this thesis alone has run for over ten years. I held and broke many views over that time and found exceptions to most assertions that could be made.

Here are seven things I learned over the course of the research. The more reliable lessons are older and hence less my personal contribution. In what follows, I discuss the reliability of each.

Incremental development

I found incremental development to be constant and valuable over the research period, in all research methods. It is a reliable strategy. Incremental development is not, however, a "research result" of this work.

People as a first-order driver

People's characteristics have a first-order effect on a project's trajectory. However obvious this may seem after being stated, it needed to be rediscovered, given the relative paucity of material on the subject in the literature.

There is a difficulty in this result, which is that people, viewed in numbers, embody almost any contradictory pair of sentences one might care to construct. Therefore, in evaluating these results, we need to look for how reliable we can expect the results to be.

For qualitative results on a subject involving people, we might look for results valid for at least eight out of ten projects. Does my research support the claim that people are a first-order driver on projects at that rate? The paper "Characterizing people as first-order, non-linear components in software development" names a sufficient set of projects for such a claim to be made, and my project visits since that paper indicate the strength of that result is likely to be higher. Other results, such as the effectiveness of richer communication channels, also hold at the eight-out-of-ten level.

Proximity and informal communication

The power of proximity and informal communication has been validated repeatedly from the 1970s through the 1990s (Allen 1990, Olson 2000). Currently it is both being revived and challenged. The revival is coming through what are being called the "agile" methodologies and the "Agile Alliance" (Cockburn 2002 ASD, Schwaber 2001, Fowler 2000, Highsmith 2002, Agile URL). The agile methodologies uniformly call for proximity and informal communication (that many writers call for something does not yet make it reliable, of course!).

The challenge is that distributed workgroups are a reality in many companies. Fortunately, "Characterizing people as first-order, non-linear components in software development" (Cockburn 2000 SCI), *Agile Software Development* (Cockburn 2002 ASD), and "Proximity matters" (Olson 2000) all isolate properties of the proximate, informal situation and discuss how faster, better communication technology can improve communication-at-a-distance, even if not matching proximate communication. Now that we have isolated several key characteristics of proximity-based communication, it is a natural that technologists will work to invent and deploy technologies to capture some number of those characteristics. Curtis (1995) and Herring (2000) both discuss the use of technologies that work across distances; more are under investigation all the time.

The 2D project grid

The 2D project grid from "Selecting a project's methodology" (Figure 3.2-1) I view as an intermediate result. The separation of team size from system criticality has been useful to me, and other people tell me it is useful to them also. Part of its value is simply to get people to actually hear the two statements: "Different projects need different methodologies" and "Methodology weight *must* increase as the staff size increases." I have found this grid useful in showing people where their project resides in the taxonomy and getting them to contemplate the sort of methodology that ought to fit a project of that sort. Its power lies in its very simplicity.

More recently, I have found that the sections of the methodology affected by the criticality dimension can be shaped more simply by working through the priorities dimension. To draft a methodology for a project, I place the project in the grid to get an idea of its place in this simple taxonomy, but then I delete the criticality dimension and place the criticality concern among the other project priorities. I envision extending this idea in the future.

Overall, a project leader can choose to start from the grid, or Hohmann's sliding bars, or Glass's categories, or the change cases of the Rational Unified Process. Any of those provide a starting place, a "base" methodology, for the team. Their methodology-tuning workshops will introduce many other issues, far beyond what the simple grid can. Consequently, I consider the grid as a useful, heuristic by-product of the main research.

The methodology design principles

The paper "Selecting a project's methodology" presents four methodology design principles. As just described in "Extensions to the Results," I have since extended this number to seven (Cockburn 2002 ASD). These are indeed research results from my investigation period. How reliable are they?

I have found them reliable at the eight-out-of-ten level. I have found them reliable enough to use on projects when combined with my own, unarticulated judgment. I have validated them against the set of successful projects in my interview portfolio and for predictive power against project stories told to me informally. I have used them predictively and successfully on live projects. On the eBucks project, in particular, I used the principles intentionally and produced from them a methodology quite different from that described in the Project Winifred case study (Cockburn 2002 ASD).

These tests place the principles above the needed reliability threshold. They are reliable enough for other people to put to the test. Although they work for me and have passed other people's scrutiny, do they work as written? What caveats should be placed around them? Are other principles needed to counteract overreliance on these seven?

These questions will have to wait for another researcher. They cannot be addressed any further by me.

The interaction between methodology and ecosystem

The term *ecosystem*, to refer to a project's specific and local characteristics, is a by-product research result. The term only names an analogy, of course. It is not a claim to be validated. However, having the name in place permits a particular sort of discussion that was more difficult to hold before. That the name has perceived value is marked by the appearance of a book entitled *Agile Software Development Ecosystems* (Highsmith 2002).

The term "ecosystem" is a metaphorical term. As such, its claim to power is based on how well the metaphor matches continued use, as the concepts embedded within the term "ecosystem" are extracted and related to a software team. It has served well, so far, in suggesting that stairs act as "cliffs," which reduce interchange and help species protect their niches, and in suggesting that the arrival and departure of particularly strong-willed people ("predators") should affect the ecosystem enough to generate a methodology retuning. How much further the metaphor can serve remains to be seen.

Software development as a "cooperative game of invention and communication"

The cooperative game metaphor is another by-product of this research. From the metaphor, I was able to generate and reproduce several ideas:

? Mechanisms that help people *invent* better should help a project's progress. This leads us to pay attention to brainstorming sessions, paper-based user interface prototyping, , use of whiteboards and “class-responsibility-collaboration” (CRC) cards, and collocation of teams, to name some standard invention-support mechanisms.

? Mechanisms that help people *communicate* better should help a project's progress. The lead ideas here come from capturing the characteristics of people communicating in close proximity. These ideas have been extensively discussed already.

? *Community* and *morale* show up as ideas relevant to a cooperative game. As core elements affecting software development, these ideas are still new. I consider them useful leads for ongoing examination. The term *pride-in-work*, related to morale, has already shown up in my project interviews.

The power of a metaphor is how well it matches continued use. This metaphor has passed an initial plausibility test, but is too new for further reliability evaluation.

The JIT methodology tuning technique

The JIT methodology tuning technique is a major result of the research. The test of its validity lies partially in my use of it and partially in other people's use of it.

? I "discovered" the technique from an interview of the Canadian project "Ingrid" in 1993 (Cockburn 1998 SOOP, Cockburn 2000 SCI). Project Ingrid involved 30 people using C++ for the first time, on a workstation-mainframe architecture. They, by their description, failed badly on their first increment and attributed their eventual success to rebuilding the methodology after every four-month increment.

The project Ingrid team did not name the technique, nor even hint at its being a technique *per se*. They simply behaved in a certain way. This is normal, that people *do* something without bothering to name or even isolate it. Indeed, I isolated and named the technique only *after* having used it myself several times.

? In 1994-1995, I helped apply the technique on Project Winifred. Winifred took place in the U.S. and involved 24 programmers among 45 people total, a mixed experience base, and Smalltalk-Relational-COBOL technology on a three-tier client-server-mainframe architecture. The application was for use by in-house users. Shifting the methodology to match the needs of the project within its duration helped the team operate effectively during the project.

? In 1998, I deliberately applied the technique to the NB Banking Project (Cockburn 2000 SCI). That project took place in Norway and used COBOL, assembler, CICS, and LU 6.2 technology on IBM mainframes. The system managed transactions between Norway's banking institutions and was primarily driven by asynchronous banking requests. Shifting the methodology was critical to that project's success, since the staffing changed from 3 experienced developers at the very start to 8 mostly new people a few months later, and it subsequently turned out to be necessary to coordinate with another, 35-person project located at a different site.

? In 2001, I specifically introduced the technique to eBucks.com, a 50-person South African company with 16 programmers. Their web-based system provides banking-related services to the public at large. I phrased the company's "methodology" as conventions, which got reviewed every two weeks, after each new release of functionality to the web. I helped write the starter methodology and was present for the first reflection workshop. They continued those workshops afterward. (This case is discussed in the description of Crystal Orange Web (Cockburn 2002 ASD).)

A better test of reliability, however, is when the technique is used by other people, without my intervention.

? In 1998, I described the technique to Jens Coldewey, who used it on a project, "Insman," in Germany from 1998 to 2002 (Highsmith 2002, Cockburn 2000 SCI). He used the approach without my supervision or direct involvement (although I did attend one of the post-increment reflection workshops). That team of six people considered the approach a critical success factor for them, because their sponsors kept changing their project mandate and the technology to be used. By delivering running software every three months and revising their working conventions after every delivery, they were able to follow changing business mandates and also change implementation technologies and architectures.

A third test of reliability is when other people discover it for themselves. After I named the technique and started publicizing it, other people started to see that they had already been using it, to various degrees. These days, Extreme Programming instructors make the post-iteration reflection session a standard practice. It is not named in the original XP book (Beck 1999), but is often referred to by XP teachers as "the unwritten, 13th practice of XP."

? The technique became used in the Chrysler "C3" project (personal communication), although that practice never reached the stage of being described in the documentation on the project.

? Joshua Kerievsky, once he encountered it, adopted it for himself. Having been a party to its use on an XP project (personal communication), he now considers it a core practice even on non-XP projects. He publicized his poster method for collecting the team's recommendations (Cockburn 2002 ASD).

? The reflection workshop is regularly practiced by Ken Auer on his XP projects (personal communication).

What is interesting here is that both Jeffries and Auer practice the reflection workshop, both have written books on installing and applying XP (Jeffries 2000, Auer 2001), and yet neither thought to mention that practice in their books. In other words, the practice was still part of their undifferentiated activities at the time they wrote those books. More recently, each of them has spoken in public about the use of those workshops.

Highsmith mentions a similar sort of workshop, the product review workshop, in his *Adaptive Software Development* (2000), and he extends that to a review of both the product and the methodology in his subsequent *Agile Software Development Ecosystems* (2002).

In each case I have heard of its being applied, the people involved mentioned it as having a significant positive effect on the project's trajectory — whether the product eventually shipped or the project was canceled — and said they would certainly apply it again in the future.

Further testing the reliability of the technique requires it to be applied by more people. The next test is for people to apply it from the written description and for researchers to inquire into what occurred as a result.

4.3 Relating to Mathiassen's Reflective Systems Development

Of the various authors in the literature, Mathiassen's (1998) results most closely match mine at this stage of the research. He similarly identifies a need for multiple methodologies and a need to fit the team's practices to their specific situation. He understands the tentative relation between the methodology as *espoused theory* and the behavior of the team as *theory-in-practice*. He also takes into account the fact that people in different job roles see the various work products in differing ways:

> [W]e might share the same ambition, but based on the understanding that any such abstract formulation represents complex and dynamic relations between the project and its use environment. Depending on which perspective you have, requirements management has different meanings and practical implications.

His resolution is to ask for the developers to become reflective practitioners:

> Systems developers must open their minds and engage in reflections and dialogues to generate the necessary insights into the situation at hand.
>
> [C]ommunities-of-practice foster a kind of learning-in-practice in which the specific context and the shared experiences and understandings of its members shape how new methods are adapted, modified, and maybe rejected to transform present practices.

Mathiassen's results are, of course, more extensive than just these short sentences. He is particularly interested in the difference between espoused theory (what people say they do) versus theory-in-practice (what they actually do). Written methods and methodologies are espoused theories. They are what the people say they did or will do. The actual methods and methodologies they use involve some differences, things they do but don't mention, either because they don't think to mention or don't know to mention them. His work does include the team to some extent, as well as the individual, primarily viewed as a community of practice.

Bearing that in mind, we find two key differences between his results and mine:

Method versus methodology (practices versus technique-as-policy)

Mathiassen uses the word *method*, and I use *methodology*, in exactly the manner set forth in the first chapter of this thesis. That is, Mathiassen is concerned with the practices of the individuals in their daily life, including their knowledge about their practice and also the relationship of their knowledge and practice to the team's. He analyzes introducing a new method (a normative practice) and how that method works its way into the daily practice of people in the team. In the end, the practice that he introduces to the team is "to reflect while in action," to become a reflective practitioner.

The consequence of his approach is that we are led to address the matter (in my words): "How do we get practitioners to become reflective practitioners? How do we arrange for them to internalize and make part of their working habits the habit of reflecting on their work in the course of their work?" The emphasis is on changing the work practices of a number of individuals.

With the word *methodology*, I am concerned with the *team's* adopted conventions and policies, not their *individual* work habits.

One result of my research is the construction of a particular technique that can be employed on specific occasions. It happens that the technique involves reflection, but my intent is not directly to evolve all the team members into reflective practitioners. My intent is to present the team with a technique that is simple enough that they can select it as a project policy, apply it on purpose on specifically demarcated occasions, and then return to their daily work practices, which may be reflective or not, according to each person.

The two results complement each other. Reflective practitioners should find the reflection workshops a natural extension to their work habits. Non-reflecting practitioners may increase their tendency to reflect through practicing the periodic reflection workshops. In an ideal world, both results would occur.

Analysis of human characteristics

Mathiassen analyzes the characteristics of people and evaluates behaviors in the acquisition of new practices, thus ruling out certain approaches for introducing new practices.

I analyze the characteristics of people and recommend certain project policies (use of proximity and personal communication, for example). I identify selected social elements project leaders should attend to (morale, communication frequency, pride-in-work), to help the team find ideas to use as conventions and to help them adapt the project conventions to the particular individuals who show up.

4.4 Reflection: The Limits of People-Plus-Methodologies

It is instructive to reconsider the research work to show both its limitations and its possible future.

Limitations

Software development is a business that is very technical and dynamic, changing every year with the arrival of new technologies. I found it sobering to receive the following in an email request (in 1998):

> For security reasons, I can't tell you about our project very much. This is the Electronic Card Payment system. In general, a framework of this project consists of Sybase PowerDesigner, Delphi, C++, RDBMS Oracle/UNIX/NT. I propose to use UML notation and Rational Rose98 to improve project team efficiency and project documentation. But I have no experience of mid-size real UML projects. So, this is the difficulty. I'm doing C++, Delphi modules with DCOM (and technical writing, also). Sometimes I need to use PowerDesigner. My background responsibility is RDBMS/SCO/NT management.

This writer is first and foremost faced with learning a bundle of new technologies and then putting them into their respective places in the software project.

I have argued so far in this thesis that people and their coordination are key factors in predicting project trajectories. However, merely setting up a good collaborative environment will not solve this designer's problem. His immediate problem is information overload, not methodology or software design techniques or even communicating.

A second limit to the results is that the answers I propose to the research questions presuppose a certain amount of expertise by the methodology adopter or project manager. Advice such as "let people communicate" and "tune the methodology on the fly" are hard to apply by people who have not already done some similar work before. According to (Cockburn 2002 ASD), we can divide the audience into three groups:

> Level (I): Those who are newcomers and need concrete rules to apply, in order to gain experience
>
> Level (II): Those who know the rules and are looking to understand how and when they may be broken
>
> Level (III): Those who are so experienced that they adopt and drop rules on the fly, often without noticing

The information in this thesis may be immediately applied by members of the level (III) software development audience: They will simply fold these observations in with those they already know. They will use and alter these new rules as they already do. On the other hand, people in the level (I) audience are likely to be disconcerted by the contents of the thesis: "What do you mean, vary the rules on each project and just let the people talk to each other? How do I do that? How do I ensure they have the right skills, are talking about the right things, and are being rewarded in the right way?"

There will always be a level (I) audience, because people are continually moving into the field and growing professionally. This means that experts should continue to produce rules, tips, and advice for level (I) and level (II) readers, even though level (III) practitioners know such advice is limited in applicability. Such texts are, of course, already available. They are simply the normal texts that are used to teach people entering the field: how to gather requirements, write use cases, design user interfaces, do OO design, write test cases, and so on.

It is important and perhaps comforting to recognize that this thesis's research results do not invalidate those texts. This thesis's research results make it clear that use of those techniques lives inside a larger context, one of constantly shifting social contracts. One point of this thesis is to alert newcomers to the field of that larger, shifting context and to give team leaders a handle on it.

Future Work

This thesis points to a number of future areas to investigate. Broadly, they fall into three topics:

Statistically demonstrate (or refute) the predictions made by the thesis

In this research work, I suggested distinctions, words, and theories that, based on the case studies and interviews, offer themselves as *useful* to live projects. The next step is to create statistical studies of various of the principles. Here are some specific questions for future examination:

- ? To what extent does pride-in-work reflect on morale, product quality, and timely delivery?
- ? To what extent do project teams feel that process is less important for successful project outcome than communication proximity and citizenship?
- ? Can citizenship, amicability, and pride-in-work be affected or manipulated by deliberate intervention techniques?
- ? What is the difference in communication quality between electronically mediated multimedia communications and in-person ones?
- ? Why is video less effective for discussion than physical presence: Is it visual field, temporal feedback, or something else?
- ? Is there a way to quantify the additional cost of additional methodology elements?

One other question is not immediately obvious but relates to a team's ability to apply these findings; that is, the relationship of the methodology to specific individuals:

- ? Iterative development is one technique that is considered a "better" practice than waterfall development, but this presupposes a greater tolerance for uncertainty and ambiguity in its practitioners. In general, how does the tolerance for uncertainty or ambiguity among the project staff relate to the productivity of techniques they are to use?

Allowing for the vast differences across software projects, name some subsets for which specific recommendations or predictions can be made

It would be a shame if the final result of all this work were to be, "Nothing can be said in general; every project must be viewed on its own." I would like to believe that we can identify broad subsets of projects, based on project size, team seating distances, personality characteristics of the team, technologies being used, etc., and that we can then say something specific and useful to the project team based on that subset.

Jones (2000) and Glass (1995) have both worked on categorizing projects. Robert Glass (personal communication) is working toward an understanding of this question — which methodologies apply to which classes of projects.

Two questions for future work are:

For how many categories do we need to capture base methodologies?

Is there a way to predict the domain of validity of a base methodology?

Name other characteristics of people that noticeably affect project trajectory

My notes contain project stories whose elements still lie outside the lexicon developed in this thesis. For example:

> Liz is not a particularly outstanding programmer, but she is friendly and a good communicator. When she is on projects, they tend to do well. In some way, she evidently lowers the tension between people and increases trust and communications flow. But what, more precisely, is her effect?
>
> Volker, a manager, saw four expert programmers fighting for weeks over which development environment they should use. He replaced three of the four expert programmers with three less excellent ones. The team went on to deliver the software without further trouble. What did he notice that caused him to intervene? What distinctions and words explain why the three less-qualified programmers outperformed the three senior programmers they replaced?

Within and beyond these stories are hints and cues that I have not even noticed enough to name. Just to name a few, I have not yet investigated neurolinguistic programming, body rhythms, or socio-cultural variations.

Name other characteristics that noticeably affect project trajectory besides people

It is appropriate to end with the observation that I have been heavily focused on project cues related to the characteristics of people. That focus probably obscures important, non-people issues. The final challenge is to drop the vocabulary built up so far and start noticing other ways to discuss what really is happening on projects.

References

R.1 References to Other Authors

Ahern, D., Turner, R., Clouse, A., *CMMI Distilled: A Prcatical Introduction to Integrated Process Improvement*, Addison-Wesley, Boston, MA, 2001.

Alexander, C., Ishikawa, S., Silververstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S., *A Pattern Language*, Oxford University Press, 1977.

Allen, T., *Managing the Flow of Technology*, MIT Press, Boston, 1984.

Argyris, C., *Increasing Leadership Effectiveness*, Wiley, New York, 1976.

Argyris, C., Schon, D., *Organizational Learning II: Theory, Methods, and Practice*, Addison-Wesley, Reading, MA, 1996.

Barnett, B., "ValidationVee," http: / / c2.com / cgi / wiki?ValidationVee.

Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Boston, 1999.

Becker, S., Whittaker, J., *Cleanroom Software Engineering Practices*, Idea Group Publishing, Hershey, PA, 1996.

Boehm, B., "Industrial Software Metrics Top 10 List," *IEEE Software*, 4(5), Sept 1987, pp. 84-85.

Boehm, B., *Software Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.

Bordia, P., Prashant, "Face-to-Face Versus Computer-Mediated Communications: A Synthesis of the Literature," *The Journal of Business Communication*, 34(1), Jan 1997, pp. 99-120.

Checkland, P., *Systems Thinking, Systems Practice*, Wiley, Chichester, UK, 1981, pp. 149-161.

Coad, P., *Java Modeling in Color with UML*, Prentice Hall, Upper Saddle River, NJ, 1999.

Curtis, P., Dixon, M., Frederick, R., Nichols, D., "The Jupiter Audi/Video Architecture: Secure Multimedia in Network Places," in *Proceedings of ACM Multimedia `95*, San Francisco, CA, pp. 79-90.

DeMarco, T., Lister, T., *Peopleware, 2nd Edition*, New York, Dorset House, 1999.

DeWeese, P., Boutin, D., "Single Software Process Framework," Software Technology Conference, 1997. Available online at http://www.stc-online.org/cd-rom/1997/track1.pdf

Ehn, P., *Work-Oriented Development of Software Artifacts*, Arbetslivscentrum, Stockholm, 1988.

Ernst and Young, *Navigator Systems Series Overview Monograph*, Ernst and Young, 1990.

Extreme Programming, web notes, start from www.extremeprogramming.com.

Galliers, R., "Choosing Information System Research Approaches," in Galliers, R. (ed.), *Information System Research: Issues, Methods and Practical Guidelines*, Blackwell Scientific Publications, Oxford, 1992, pp. 144-162.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns*, Addison-Wesley, Boston, MA, 1995.

Glaser, B, Strauss, A., *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine de Gruyter, NY, 1967.

Glass, R., Vessey, I., "Contemporary Application-Domain Taxonomies," *IEEE Software*, 12(4), July 1995, pp. 63-76.

Goldratt, E., *The Goal*, North River Press, Great Barrington, MA, 1992.

Graham, I., Henderson-Sellers, B., Younessi, H., *The OPEN Process Specification*, Reading, MA, Addison-Wesley, 1997.

Gries, D., *The Science of Programming*, Springer Verlag, 1986.

Guindon, R., Curtis, B., "Insights from Empirical Studies of the Software Design Process," *Future Generation Computer Systems*, 7(2-3), April 1992, pp.139-149.

Harrison, N, "Organizational Patterns for Teams", in *Pattern Languages of Program Design 2*, Vlissides, Coplien, Kerth, eds., Addison Wesley, 1995, pp. 345-352.

Herring, R., Rees, M., "Internet-Based Collaborative Software Development Using Microsoft Tools," in *Proceedings of the 5th World Multiconference on Systemics, Cybernetics, and Informatics* (SCI'2001), 22-25 July 2001. Orlando, FL, online at http://erwin.dstc.edu.au/Herring/SoftwareEngineering0verInternet-SCI2001.pdf.

Highsmith, J., *Adaptive Software Development*, Dorset House, New York, 1999.

Hohmann, L. *Journey of the Software Professional*, Prentice Hall, Upper Saddle River, NJ, 1997.

Hovenden, F., "A Meeting and A Whiteboard," in *Proceedings of the 4th World Multiconference on Systemics, Cybernetics, and Informatics* (SCI'2000), Vol. I ("Information Systems"), 23-26 July 2000. Orlando, FL.

Hult, M., Lennung, S.-A., "Towards a Definition of Action Research: a Note and Bibliography," *Journal of Management Studies*,17(2), 1980, pp. 241-250.

Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley, Reading, MA, 1995.

Israel, J.,*The Language of Perspectives and the Perspectives of Language*. Munksgaard, Copenhagen, 1979.

Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, Boston, 1999.

Jones, T. Capers, *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, Boston, 2000.

Krutchen, P., *The Rational Unified Process*, Addison-Wesley, Boston, 1999.

Lave, J., Wenger, E., *Situated Learning: Legitimate Peripheral Participation*, Cambridge U. Press, 1991.

Lyytinen, K., "A Taxonomic Perspective of Information Systems Development. Theoretical Constructs and Recommendations," in R. J. Boland et al., *Critical Issues in Information Systems Research*, John Wiley, Chichester, UK, 1987.

Markus, M.L, "Asynchronous Technologies in Small Face to Face Groups," *Information, Technology & People*, 6(1), Apr 1992, p.29.

Martin, J., Odell, J., *Object-Oriented Methods, Pragmatic Considerations*, Prentice Hall, Upper Saddle River, NJ, 1996.

Mathiassen L., "Collaborative Practice Research," in Baskerville, R., Stage, J., DeGross, J., *Organizational and Social Perspectives on Information Technology*, Kluwer Academic Publishers, Norwell, MA, 2000, pp. 127-148. Online at http://www.cs.auc.dk/~pan/pdf/mathiassen-aalborg2000.pdf.

Mathiassen, L., "Reflective Systems Development," *Scandinavian Journal of Information Systems*,10, (1&2), 1998, pp.67-117. Online at http://www.cs.auc.dk/~larsm/rsd.html.

McCarthy, J., *Dynamics of Software Development*, Microsoft Press, 1995.

McCarthy, J., Monk, A., "Channels, Conversation, Cooperation and Relevance: All You Wanted to Know About Communication but Were Afraid to Ask," in *Collaborative Computing*,1(1), March 1994, pp. 35-61.

McTaggart, J., McTaggart, E., "Studies in the Hegelian Perspective," 1896, readable on the web at www.its.uidaho.edu/mickelsen/ToC/McTagart.htm.

Monarchi, D., Puhr, G., A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM*, 35(9), 1992, pp. 35-47.

Ngwenyama, O., "Groupware, Social Action and Organizational Emergence: On the process dynamics of computer mediated distrubuted work," in *Accounting, Management and Information Technologies*, Vol. 8, No. 2-3, 1998, pp. 127-146.

Naur, P., "Programming as Theory Building," pp.37-48 in *Computing: A Human Activity*, ACM Press, 1992.

Nygaard, K., "Program Development as a Social Activity", in *Information Processing 86*, H.-J. Kugler (ed.), Elsevier Science Publishers B.V. (North Holland), IFIP, 1986 (Proceedings from the IFIP 10th World Computer Congress, Dublin, Ireland, September 1-5, 1986), pp. 189-198.

Olson, G.M., and Olson, J.S. "Distance Matters," *Human-Computer Interaction*, Vol. 15, 2001, pp. 139-179.

OMG, http://www.omg.org.

Paulk, M., ed., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, MA, 1995.

Royce, W. "Improving Software Economic Development Economics Part I," *The Rational Edge*, April 2001. Online at http://www.therationaledge.com/content/apr_01/f_econ_wr.html.

Probasco, L., "The Ten Essentials of RUP," *The Rational Edge*, Dec 2000. Online at http://therationaledge.com/content/dec_00/f_rup.html.

Schwaber, K., Beedle, M., *Scrum: Agile Software Development*, Prentice Hall, Upper Saddle River, NJ, 2001 (see also http://www.controlchaos.com).

Soloway, E., Spohrer, J. (eds.), *Empiricial Studies of Novice Programmers*, Erlbaum and Associates Press, 1988.

Stapleton, J., *DSDM Dynamic Systems Development Method: The Method in Practice*, Addison-Wesley, Boston, 1997.

STC: www.stc-online.org

Schön, D., *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, Inc., 1983.

Straus, A., Corbin, J, *Basics of Qualitative Research: Grounded theory procedures and techniques*, Sage Publications, Inc., 1990.

Thoresen, K., *Computer Use*, Dr Philos thesis, University of Oslo, March 1999.

Vessey, I., Glass, R., "Strong vs. Weak Approaches to Systems Development," *Communications of the ACM* , 41(4), April 1998, pp.99-102.

Weinberg, J., *The Psychology of Computer Programming*, Silver Anniversary Edition, New York, Dorset House, 1998.

Weisband, S., Schneider, S., Connolly, T., "Computer-Mediated Communication and Social Information: Status Salience and Status Differences", *Academy of Management Journal*, 38(4), Aug 95, pp. 1124-1143.

R.2 Cockburn Papers (Chronologically)

(1993 IBMSJ) "The Impact of Object-Orientation on Application Development," *IBM Systems Journal*, 32(3), March 1993, pp. 420-444, reprinted in the 50-year anniversary issue, *IBM Systems Journal* 38(2-3), 1999. Available online through http: / / www.research.ibm.com / journal / sj38-23.html (html http: / / www.research.ibm.com / journal / sj / 382 / cockburn.pdf).

(1994 OM) "In Search of Methodology," *Object Magazine*, July 1994, pp.52-56,76. Online at http: / / members.aol.com / humansandt / papers / insearchofmethy.htm

(1995 OM) "Unraveling Incremental Development," *Object Magazine*, Jan 1995, pp. 49-51.

(1995 PLOPD) "Prioritizing Forces in Software Architecture," in *Pattern Languages of Programs 2*, Addison-Wesley, Reading, MA, 1996, pp. 319-333.

(1995 SPE) "Using Formalized Temporal Message-Flow Diagrams," with Citrin, von Kaenel, and Hauser, *Software Practice and Experience*, 25(12), 1995, pp. 1367-1401.

(1996 CACM) "The Interaction of Social Issues and Software Architecture," *Communications of the ACM*, 39(10), Oct 1996, pp. 40-46, online at http: / / members.aol.com / acockburn / papers / softorg.htm and http: / / www.acm.org / pubs / articles / journals / cacm / 1996-39-10 / p40-cockburn / p40-cockburn.pdf

(1996 TR) "d(Ad) / d(Hf) : Growth of Human Factors in Application Development," *Humans and Technology Technical Report*, online at http: / / members.aol.com / acockburn / papers / ad-change.htm

(1997 DOC) "PARTS: Precision, Accuracy, Relevance, Tolerance, Scale in Object Design" in *Distributed Object Computing*, 1(2), 1997, pp. 47-49. Online at http: / / members.aol.com / acockburn / papers / precisio.htm

(1997 TR) "Software Development As Community Poetry Writing," *Humans and Technology Technical Report*, online at http: / / members.aol.com / acockburn / papers / swaspoem / swpo.htm

(1997 VW) "Using "V-W" staging to clarify spiral development," OOPSLA'97 Practitioner's Report.

(1998 SOOP) *Surviving Object-Oriented Projects*, Addison-Wesley, Reading, MA,1998.

(1998 TR) "Exploring the Methodology Space," *Humans and Technology Technical Report*, 1998, http: / / members.aol.com / acockburn / papers / methyspace / methyspace.htm

(2000 Cutter) "Balancing Lightness with Sufficiency," *Cutter IT Journal*, 13(11), Nov 2000, pp. 26-33.

(2000 LW) "The Costs and Benefits of Pair Programming," with Laurie Williams, presented at the Extreme Programming and Flexible Processes conference, Sardinia, June 2000, online at http: / / members.aol.com / humansandt / papers / pairprogrammingcostbene / pairprogrammingcost-bene.htm

(2000 SCI) "Characterizing People as Non-Linear, First-Order Components in Software Development," presented at the 4th International Multi-Conference on Systems, Cybernetics, and Informatics, Orlando, FL, June 2000.

(2000 Software) "Selecting a Project's Methodology", *IEEE Software*, 17(4), July/Aug 2000, pp.64-71.

(2000 XPFP) "Just-in-Time Methodology Construction" presented at the Extreme Programming and Flexible Processes conference, Sardinia, June 2000, online at http: / / members.aol.com / humansandt / papers / jitmethy / jitmethy.htm

(2001 JH1) "Agile Software Development: The Business of Innovation," with Jim Highsmith, *IEEE Computer*, 18(9), Sept 2001, pp. 120-122.

(2001 JH2) "Agile Software Development: The People Factor," with Jim Highsmith, *IEEE Computer*, 18(11), Nov 2001, pp. 131-133.

(2002 ASD) *Agile Software Development*, Addison-Wesley, Boston, 2002.

(2002 Cutter) "Agile Software Development Joins the 'Would-Be' Crowd," *Cutter IT Journal*, 14(1), Jan 2002.

(2003 Clear) Cockburn A., *Crystal Clear: A Human-Powered Software Development Methodology for Small Teams*, Addison-Wesley, in preparation. Online at http://members.aol.com/humansandt/crystal/ clear.

Appendix: The Papers