Characterizing people as non-linear, first-order components in software development

Alistair A.R. Cockburn
Humans and Technology

HaT Technical Report 1999.03, Oct 21, 1999
Presented at the 4th International Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, June, 2000.

Abstract

We methodologists and process designers have been designing complex systems without characterizing the active components of our systems, known to be highly non-linear and variable (people). This paper outlines theories and projects I reviewed on the way to making this stupendously obvious but notable discovery and four characteristics of people that most affect methodology design and project outcome. I find these characteristics of people to be better predictors of project behavior and methodology success than other methodological factors.

Keywords: methodology design, process, human factors, people factors

1 – Introduction

This is a report from experience, from reviewing roughly three dozen projects and methodologies over 20 years. The short of the story is that we have been designing complex systems whose active components are variable and highly non-linear components called people, without characterizing these components or their effect on the system being designed. Upon reflection, this seems absurd, but remarkably few people in our field have devoted serious energy to understanding how these things called people affect software development.

The notable exceptions are Gerald Weinberg [Wei] and Tom DeMarco [Dm], whose books are being reissued in anniversary(!) editions. Their work is well regarded, and consultants recognize humans as a dominant consideration in software development [B99], [Hi]. One would expect to find major research efforts extending their work. However, the software development community has largely ignored the characteristics of people as a topic of study. This is a significant oversight, akin to ignoring iron in walls and wondering why radio experiments are not giving the expected results.

Like most people, I referred to people on projects only superficially, until after several years as a research and practicing methodologist, I began wondering why my methodological recommendations were not matching my experiences on software projects. The problem wasn't what my various teams were *doing* (the projects enjoyed a high success rate). The problem was that what I wrote didn't match what we did.

Over the last five years, I have found – and still find – it very difficult to know "what it is I am looking at". It slowly became apparent that one thing not in my methodological equation, or, in fact, in anyone else's, as far as I can see, is the effect of "people" on methodologies.

Once I started accounting for this quantity, I found my methodological predictions and results started matching my experiences. I now consider the characteristics of people as *the dominant, first-order* project driver.

How does this differ from what DeMarco and Lister wrote in Peopleware?

DeMarco and Lister announced that people are important and gave some specific pointers. I am interested in how people's individual and group characteristics affect the design of software development practices (a.k.a. methodologies), for different groups working on different sorts of assignments.

Weinberg's chapter on "Teaming" in The Psychology of Computer Programming, particularly on 'task' versus 'maintenance' management, comes closest to what I am after: a characterization, and recommendations derived from the characterization. His characterizations and recommendations, based upon project interviews in the 1960's, are still accurate and significant 30 years later. That validates the stability and importance of these sorts of issues. It is about time we studied these issues as a core to the field of Software Engineering and stopped rediscovering their importance every 30 years.

This paper outlines the work I reviewed that made it clear to me that 'people' really are a first-order, and not incidental, project success driver, the characteristics I currently use as predictors, and their success as predictors. This paper is written in the first person rather than in formal, academic style, because it relates a quest for the obvious but unstated, and is best told in story form.

2 – What wasn't working

In the formal development of communications software, in 1987, I was given the motivation, "The *problem* with software development is that there is too much ambiguity in the problem statement and in the design description. *Things will go better* if we get people to work with mathematical formalisms." After some work in this area, I discovered:

- **Problem 1.** The people on the projects were not interested in learning our system.
- **Problem 2.** They were successfully able to ignore us, and were still delivering software, anyway.

I shifted out of formal development, hearing my formal development colleagues saying, "The *problem* lies in the training. *Things will go better* if we start training students in the necessary mathematics earlier, i.e., secondary school." Based on what I have learned about people, I expect this will be a frustrated desire. It is not that I doubt certain advantages of formal development, but I do doubt our ability to coax 10**6 people to master it. The question should become, "Under what circumstances, and with what assignment, should a formal specialist be called in?"

I shifted over to tool development, working in an ethnocentric fashion as far as possible. I watched the designers of communications protocols, and discussed with them what might be their problems. My colleagues and I decided that, "The *problem* is that the people are still drawing at the whiteboard. *Things will go better* if we give them special tools so that they can draw directly into the computer and give them early execution of their designs."

We spent several years developing an inference engine that would convert time-flow, interaction diagrams into a software architecture and rule system [Ci]. Many groups were and are working on a similar agenda, e.g., Harel's executable finite state machines [Ha].

We completed the prototype after several years and showed it to our would-be user group, and were shattered to hear them say, "No, thanks, we actually like drawing on the whiteboards. We don't want to take the time to put the drawings into the computer. Um, may we use the drawing editor portion of your tool suite?" Listening to other tools vendors, we heard the same experiences, usually ending up with, "they use just the drawing editor portion." In other words,

- **Problem 1.** The people on the projects were not interested in learning our system.
- **Problem 2.** They were successfully able to ignore us, and were still delivering software, anyway.

Troubled by this, I shifted into software development methodologies, designing the OO methodology for the IBM Consulting Group (1992-94). Not wanting to make the same mistake a third time, I interviewed over a dozen object-technology projects around the world, and wrote down what they said. Studying those notes, I concluded several things.

- The successful teams used incremental development [Co95]
- Any design technique more complicated than "CRC Cards" [B87] seemed too complex to be used
- The design teams had the priviledge of ignoring any tool or technique they didn't like. All they had to say to their boss was, "It slows me down – I won't make the schedule if I use it," and their boss would let them ignore it. At the time I did not think this was significant to write down, but I did note it as a methodology "design constraint".

I designed the most palatable, low-ceremony methodology I thought possible, and went to use it on a project. Our experiences are documented as Project Winifred in [Co98]. The core design technique I recommended, taught, and mentored on that project was based on CRC cards.

A few months later I took off my consultant's hat, put on my ethnographer's, and studied the ways in which the team was actually behaving. What I saw staggered me:

- The moment-to-moment process the team followed was so intricate that I couldn't possibly write it all down, and if I could, no one else could possibly follow it [Co98p]. It matched my process only in the most superficial way.
- Not a single person out of the two-dozen OO designers I had mentored was using CRC cards.

In other words, although I had used what I thought was an ethnographic basis for designing the methodology,

- **Problem 1.** The people on the projects were not interested in learning our system.
- **Problem 2.** They were successfully able to ignore us, and were still delivering software, anyway.

This process repeated itself a few more times, until I became quite upset at my inability to “see what really is happening”. The best I could say was that there was some important thing in the project that we hadn’t named. To address this, I recently teamed with an ethnographer, just to have help naming what is happening [Ho]. A consultant plus an ethnographer is a good pairing, but we have barely started. The trouble is, we can’t say what we are seeing until we have names for what we are seeing. Evidently, our current vocabulary is inadequate.

I have now been on, debriefed, or read detailed accounts in the literature, of approximately three dozen projects (selected aspects of which are summarized in Table 1).

Table 1. Projects and methodologies reviewed. This table, which must necessarily be abbreviated, shows projects with year, nickname, and a note about each. Some of the projects are further documented, as the references show.

1980. “CT5”. Success. 26 people, 3 years (1 year late), company critical. Learned by apprenticeship, well-defined macro process, no micro-process.

1986. “Cleanroom” projects [Mi]. Success. IBM Federal Sector, large-team projects. Repeated success with heavy, high-discipline methodology.

1986. “Sherr’s projects” [Br] Success. Process: “make it work, but don’t work overtime” forced creative solutions, no defined process.

1980. “Broooklyn Union Gas” [Co98]. Success. New OO technology, 150 people, mission critical project.

1992. “Tracy” [Co98]. Failure. Small team blindly followed the methodology that said,”model the world, then code it up.” Only had access to occasional users and untrained staff.

1992. “BlackKnight”. Success. Small team happy with post-it notes connected by yarn

1992. “Manfred” [Co98]. Failure. Small team, low discipline, lightweight. “We’ll engineer it later” failure of continual prototyping.

1992. “CSPITF”. Success. Small team managed iterations carefully. Lightweight process, sitting close together.Good synergy between technical lead and project manager. Lead stayed on to restructure code internals for next team.

1992. “OTI” [Co98].. Success. Small teams. “Give good people good tools, and leave them alone”. Repeated success with light, people-centric methodology.

1993. “Reginald” [Co98].. Failure. 2-person team grew to 3 teams in two counties. One team blindly followed paper-heavy methodology, never produced code before project cancelled.

1993. “Ingrid” [Co98].. Success. 26-people, 2-years. Incremental macro process, no micro-process. First increment failed. Replaced all programmers, evolved lightweight, communication-centric methodology over time.

1993. “Synon in NZ”. Success. Project leader claimed success was due to “4 people in one room, use of fast iterative tool”, would not take on project in which people could not talk to each other easily.

1994. “Udall” [Co98]. Success. Failed initially with a large team. Success due to “starting over, extracting a good small team from fumbling large team.”

1995. “Winifred” [Co98]. Success. 45 people, 20 months. Success dues to “increments, good communications, a few good people ”. Macro process, no micro process. Successful medium-sized, communication-centric metodology

1996. “Reel”. Failure. 150-people told to update every document with every design change. Project canceled. Project participant summarized: “Diligent use of bad practices is still bad.”

1997. “Caliper”. Failure. 90-people, company critical. Still missing first major delivery after 6 years. High expectations, new technology, no increments, inadequately skilled staff in all roles.

1997. “NorgesBank”. Interviews with 6 project teams. Common phrase in all interviews: “success when good communications in team and with users.”

1998. “C3” [C3]. Success. 8 people replaced 26 after initial failure. Extreme Programming [EP]. Successful small-team, high-discipline, communication-centric methodology.

1998. “NB Banking”. Success. 3 people, 2-month project grew suddenly to 10 people, 14-months. Disliked video link. Macro but no micro process. Success due to “increments, adequate people & communication”.

1998. “M.A.D.” [Ch]. Success. Small team, studying end user context, used prototypes. Successful use of communication-centric, prototyping methodology.

1998. “Insman”. Success. 6 people using “Crystal(Clear)” [Co00]. Success due to “focusing on close communication, group morale, 3-month increments with team learning.”.

1999. “Cinch”. Ongoing. 40 people sitting close, but still requiring formal deliverables. Recognize cost of writing, but unable to shift to personal mode (personalities, habit, or culture?).

1999. “Hill AFB TSP1” [Web]. Success. 7 people, CMM level 5 group using PSP/TSP. Small team success with high-discipline, process-centric methodology.

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 did not find any theory to account for the high success rate of lightweight, low-ceremony methodologies, the low success of very-high-ceremony methodologies, with occasional exceptions as in Cleanroom and PSP/TSP. Obviously, poor management is a non-methodological factor of greatest significance, but even normalizing for that does not give meaningful predictions.

I finally concluded that there is something there, in front of us all the time, which we are not seeing: people. People’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?”

3 – Four characteristics

People, as active devices, have success modes and failure modes. The following are the main ones that I have named and used to date:

1. People are communicating beings, doing best face-to-face, in person, with real-time question and answer.
2. People have trouble acting consistently over time.
3. People are highly variable, varying from day to day and place to place.
4. People generally want to be good citizens, are good at looking around, taking initiative, and doing “whatever is needed” to get the project to work.

There are other characteristics that I won’t expand here:

- People need both think time and communicating opportunities (see [Co98], [Cs], [Dm]).
- People work well from examples (a topic for more study, however, see [J-L]).
- People prefer to fail conservatively than to risk succeeding differently [Pi]; prefer to invent than to research, can only keep a small amount in their heads, and do make mistakes, and find it hard to change their habits.
- Individual personalities easily dominate a project.
- A person’s personality profile strongly affects their ability to perform specific assignments.

People are communicating beings

The most significant single factor is “communication”. Figure 1 is the informal curve I now use to inform my methodological considerations. The figure shows communication effectiveness dropping as modalities and timing are removed. There is some research around this topic, see [Pl] and [Si], and it also matches the project experiences Weinberg documented 30 years ago [Wei].

The most effective communication is person-to-person, face-to-face, as with two people at the whiteboard. As we remove the characteristics of two people at the whiteboard, we see a drop in communication effectiveness. The characteristics that get lost are:

- Physical proximity. I am at a loss to explain why, but being physically close to the other person affects the communication. Whether it is three-dimensionality, timing, smell, or small visual cues, physical proximity matters.
- Multiple modalities. People communicate through gestures as well as words, often making a point by gesturing, raising an eyebrow or pointing while speaking.
- Vocal inflection and timing. By speeding up, slowing down, pausing, or changing tones, the speaker emphasizes the importance of a sentence, or perhaps its surprise value.
- Real-time question-and-answer. Questions reveal the ambiguity in the speaker’s explanation, or the way in which the explanation misses the listener’s background. The timing of the questions sets up a pattern of communication between the parties.

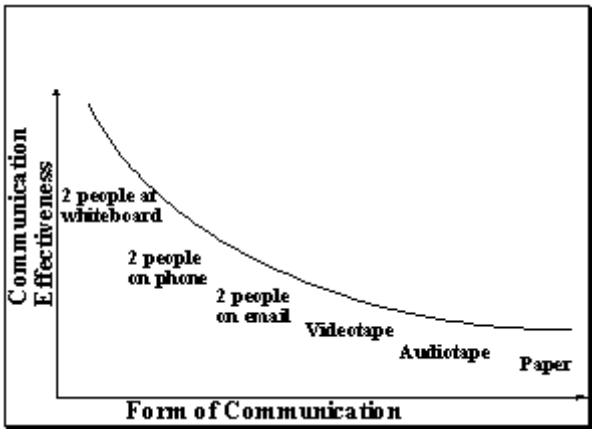

Figure 1. Modes of communication

What happens when we remove those characteristics, one by one?

- Remove only physical proximity. Put people at opposite ends of a video link. In principle, the link has the same characteristics as physical presence. However, when we try it, the effect is not the same. Teammates working in Oslo and Lillehammer found that they only made design progress when they took the train trip to sit together. Even walking from the train station together was a more effective design environment than talking over a video link.
- Remove visual gestures with visual timing, but keep vocal inflection and timing (e.g., use a telephone). Most people speak *while drawing*. While drawing the line that connects two boxes, the person will say what is important to note. This visual / auditory timing information anchors information content. Putting two people over the phone eliminates this timing, along with facial expressions, gestures, and pointing.
- Next, remove vocal timing and inflection, but keep the ability to ask questions (email). Without vocal timing, we can't pause for effect, check for interruptions, speed up or slow down to make a point. Without vocal inflection, we can't raise our tone or volume to indicate the surprise, boredom, or obviousness of the transmitted idea.
- Next, remove the ability to ask questions (but possibly reinstate one of the above factors). Without the questions, the presenter must guess what the receiver knows, doesn't know, would like to ask, and what an appropriate answer to the guessed question might be – without any feedback. In this set of communication media, we can still allow visual cues (videotape), or voice (audiotape).
- Finally, remove visual, vocals, timing, and questions, and we get… paper documentation. In the above model, paper documentation comes out as the least effective communication medium available. The writer must guess at the audience, with no feedback, and does not get to use timing or emphatic signals, vocal or gestural inflections.

If this model is useful, it should inform us as to how better to work. Indeed it does, and we find the busiest projects making use of those suggestions.

> "Put all the people into one room." "Don't give me more than 4 people, that's all I can get into one room and talking together." These are standard recommendations of successful project leaders. They plan on using the highest communication mode, people face-to-face.
>
> "Make sure there are whiteboards and coffee corners all over the building." Hewlett-Packard and IBM were early to observe the effectiveness of informal meeting places, and by now it has become a standard idiom in our industry that an effective design environment actively encourage and permit *ad hoc* meetings of small groups. Weinberg documented specific instances where casual conversations had significant effect on the overall group output [Wei]. Much progress comes when people "just talk together.".

Three recent methodologies contain, as a core element, putting people into the same room or very close together (Adaptive Software Engineering [Hi], Extreme Programming [B99], [EP], Crystal(Clear) [Co00]).

The above model also allows us to make a recommendation for archival documentation:

Have the designer give a short (5-20 minute) description of the design to one or two colleagues who are not familiar with the work. They will act as ombudsmen for the viewers of the videotape. Let the people simply have a discussion of the design, with the colleagues asking questions as they need. Video the discussion. At the end, produce drawings of the examples used in the discussion, or the design drawings used, to act as mnemonic anchors of the discussion.

I was pleased to hear from Lizette Velasquez of Lucent Technologies that not only had she profitably already used that technique, but that I had forgotten to mention something important. She said it is also important to mark and index places where "something interesting happened". While much of the discussion proceeds at a relatively slow pace, occasionally a question triggers a flurry of significant discussion, and the viewers will want to refer back to those sections.

These days, it is possible to put the talk online with hyperlinked media.

For those who still think a book is best, consider the excellent but difficult book Design Patterns. Imagine that instead of trying to extract the meaning of the "Decorator" pattern from the paper, you could click on the page and see one of the authors explaining the pattern in a video clip. They would, of course, rely on tonal inflections, gestures, and timing to get the idea across.

The lesson of this human characteristic is that we should try to move team communications up the curve as far as possible, given the situation at hand.

People tend to inconsistency

We have to be careful in talking about human failure modes. “If you give a dog a bad name, you might as well shoot him,” is an old English saying. Indeed, as two examples below show, simply changing the management style and local culture can change the apparent behavior dramatically. And yet, there is a thread running through my project experiences, how difficult it is to expect consistency of action. As Jim Highsmith writes [Hi]:

> ”...there are people, not cogs, inside the box. People can do similar things repeatedly, but never the same thing. With a step-by-step process we expect the same set of outputs from identical inputs, but people’s reaction to inputs may vary considerably from day-to-day based on a wide variety of conditions, many of them unrelated to the task at hand.”

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

> “How can I manage the flood of paper that comes into my office?” asked one person. Another replied, “It’s easy. Keep your desk entirely clean, with four baskets on one side and a set of folders in the top drawer….” The speaker never got further. “Keep the desk entirely clean!?” cried the listeners in unison. “I can’t do that!”

Note that the clear desk suggestion asks them both to change their habits *and* apply an action consistently.

If people could just act consistently, they could keep their desks clean, avoid cavities, lose weight, give up smoking, play a musical instrument, and possibly even produce software on a regular and timely basis.

As Karl Wiegers quipped, “We are not short on *practices*, we are short on *practice*.” David Gries gives detailed instructions for deriving correct programs in “The Science of Programming” [Gr]. CRC cards are a good design exploration medium [B87]. Extreme Programming [EP], with its pair programming and automated regression unit tests [Je], uses known and effective practices. The practices of the Cleanroom methodology are well documented [Mi]. Watts Humphrey gives detailed instructions on how programmers can become more effective in the Personal Software Process [Hu]. Consistent application of almost any of the above ideas would improve any of the projects I have visited.

The trouble is the word “consistent”. PSP and Extreme Programming lose their meaning when applied sporadically. A half-derived code fragment is not an error-free code fragment. Just as the clear-desk technique, they must be applied completely, consistently, daily.

Lack of consistency is a common *failure mode* of humans. Methodologies that require consistency of action, I call “high-discipline” methodologies. The project interviews indicate high-discipline methodologies as being fragile, although they have been brought to work in some projects. The following words from a CMM level 5 organization, trained in PSP, are instructive [Web]:

During the summer of 1996, TIS introduced the PSP to a small group of software engineers. Although the training was generally well received, use 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.”

To stay in place, a high-discipline methodology must contain some form of support for the people to act consistently. Cleanroom has a “no compilation” rule backed by management practices. Extreme Programming calls for a coach to keep the practices in use. PSP has no such explicit support, and so it is not surprising that in this CMM level 5 group the PSP fell into disuse for lack of support structures. The adjunct TSP is supposed to provide those factors [Web].

All of the above comments notwithstanding, there really are people who stay highly disciplined and consistent on a daily basis (illustrating the variability across people). Sometimes simply changing the manager of a group of people can change the consistency of their actions. I thank Trygve Reenskaug for the following anecdote, with which he illustrated that management style and personal chemistry matter a lot:

> There was a small-goods and buttons shop nearby that was always in a terrible shape. The place was a mess, and the girls were either doing their nails or on the phone, and didn't have much time for the customers. That business closed, and another small-goods and buttons shop opened in its place. This place was wonderful! It was clean, tidy, and the girls were attentive to their customers. The only thing was … it was the same two girls!

Indeed, it is well known that personal management style has an enormous effect. The Chrysler Comprehensive Compensation project went through this around 1997 [C3]. Initially the team placed value on "thinking ahead", "subtle but extensible designs" and "my code is private". The team, together but largely under the impetus of Kent Beck, rebuilt itself with the core values "make it simple and clear, we can add that subtlety later", "all code is public, any pair of people sitting together may change anything." The same people adopted different core values and went from disarray, non-communication and non-delivery to consistent application of a highly disciplined, different set of practices, and regular delivery over a three-year period.

Good Citizenship and Good at Looking Around

Three success modes counter the consistency problem:

- people are generally interested in being "good citizens",
- people take initiative,
- people are good at looking around.

When I interview a project, I always ask what caused them to succeed in the end. The single most common answer I receive is, "A few good people stepped in at key moments and did whatever was needed to get the job done." A typical such comment was carefully written up in NASA's "Deorbit flight software lessons learned" [NASA]:

> "Perhaps most important for the long term, during the course of the project, a capable core team for rapid development of GN&C systems evolved. This included finding talented team members; training in and gaining experience with the tools, processes and methodology, and integrating into a cohesive team.
> After working together in the RDL for a year, team members have acquired expertise in methods, tools and domain. A *helpful and cooperative atmosphere has encouraged* and enabled cross training. A willingness on the part of team members *to address any and all* project issues has proven invaluable on many occasion…"
> (my italics added for emphasis)
> ...this team can be a long term asset to the division and to the agency."

What causes people to do that? One plausible answer is, "good citizenship".

Perhaps we can increase the rate of project success simply by increasing the "sense of community and good citizenship" on the team. I actually don't propose that as a top recommendation, because I find that the sense of good citizenship is already generally high, and improper management is already taking too much advantage of it (see, for example, Death March [Yo]).

However, the idea does reveal a rarely mentioned element of methodology design:

"Community and citizenship"

should be among a project's central activities and measures, at least on a par with "code review and testing". I am happy to say that several of the projects I have visited have this as an explicit activity. They use it to keep the person-to-person communication channels in good order (needless to say, these projects also maximize face-to-face communication). However, I have yet to see it included in a written methodology or process.

The second success mode is that people take initiative. It works in concert with good citizenship and being *good at looking around*, to produce the common success factor, "a few good people stepped in at key moments."

"Good at looking around" is a vague phrase with strong effects. People sorting papers often create small, unsorted stacks, using the shellsort technique. The surprising part is that quite often they never sort the smaller stacks. Close is good enough, because they know they can skim through a pile on demand, and often will remember "approximately" where a given item is (future studies should address mnemonic anchoring techniques).

We try to create good, accurate and up-to-date design documentation. Since consistency of action is a common failure mode, we can safely predict that the documentation will not be up to date. In fact, I have not yet interviewed a successful project that had accurate documentation, unless either the code or the documentation was generated automatically. I did, however see a project fail because the developers were told to update all design documentation every time they made a change. Cost of development was just too high, and progress too slow, and the project was soon cancelled.

I ask maintenance people how they manage to make program updates, in the face of out-of-date documentation. Their answer is that they “just look around”, they don’t trust the documentation in any case – they just read the code.

I have come to rely on this “good at looking around” success mode of humans on projects and in methodology design.

Traceability documents are very expensive to create and keep correct, particularly given the weakness of humans with regard to consistency, I therefore recommend getting the traceability and design documentation “good enough” so that the investigating person can learn approximately where to look. Their eyes and intelligence will take care of the rest. Even this level of traceability and documentation is unlikely to be maintained on most projects (and it somehow doesn’t matter so much, because people work much more from talking to colleagues who know than from reading the documents).

Trygve Reenskaug gave another example. He proposed a computer-aided design system to an engineer designing offshore oil platforms. Trygve proposed that the system could add value by tracking the update activity being performed on any part of the platform. The engineer replied, “Just have it store the phone numbers of the people working on each part. I’ll call them and find out.”

The official, methodological term I use is “low precision” [Co98]. I find that most projects run just fine on (accurate) low-precision descriptions: low-precision project plans are easier to read, adjust, and negotiate. Low-precision architecture diagrams are easier to remember, low precision requirements tables are easier to prioritize and evaluate early in a project, low-precision design documentation is better at giving the reader “the idea” of the design – and then letting them look around.

Low precision artifacts use the strengths of people to lower development costs. They draw on “good at looking around” and face-to-face communication, while forgiving inconsistency of update. I have used them to good effect in projects since 1994, and now recommend them as a core methodological element.

People vary

Some people like to make lists, some don’t. Some work best nights, some work best in the morning. Some like deadlines, some don’t. Groups vary similarly. Some cultures prize public self-scrutiny, others shelter people from embarrassment, and so on.

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.

In the table of projects above, I observe that the two successful examples of high-discipline processes were in IBM’s Federal Sector Division and the Air Force. I have not yet seen high-discipline processes succeed in commercial settings. My tentative conclusion is that there are factors in the government / military settings that permit the heavier methodology to survive. My few interviews in those groups are littered with the phrase, “we’d like to work in a lighter, more efficient manner, but…”, followed by a reference either to a military process standard or to the difficulty of controlling subcontractors. It would be good to know whether those factors are intrinsic, or whether those cultures simply “tolerate” the heaviness.

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.

Cultural variations are probably even more difficult for methodologists to incorporate than the daily variation of individuals. At this time, I know of no methodology that takes cultural issues into account (including my own), although I know people who are sensitive to the local culture while formulating project recommendations..

Other characteristics

There are other characteristics of people I rely on, but will not elaborate on. These are the ones on my short list:

Apprenticeship. People learn from watching and doing. This is a cognitive and social principle, well known in certain circles [La], but not yet properly used in software development. I have come to look for opportunities to apply it, but have not yet created a proper methodological structure for it.

Flow. “Flow” in the context of programming refers to quiet and effective thinking time [Dm], [Cs], [Co98]. Flow time must be balanced with communication time. How to set that balance on a busy project is beyond my current understanding. However, many interviewees marked it as a notable project success factor.

Working from examples. Some cognitive psychologists convincingly argue that our deductive mechanisms are built around constructing specific examples of problems [J-L]. CRC cards and use cases are two software development mechanisms centered on examples, and are repeatedly cited by practitioners as effective. “Instance diagrams” are often preferred by newcomers to object-oriented design, and still are used by experienced designers. Example-based documentation does not yet get adequate support from methodologists, and is a topic for future work.

Kinesthetic or multi-sensory thinking. CRC cards, role-playing, designing at the whiteboard, paper-prototyping of user interfaces, all take advantage of a person talking, moving, and acting while thinking. We are missing research on this, but I find the use of these techniques significant in software design.

Personality profiles. We have seen the non-communicative lead designer who alienates his or her team, and yet these superb programmers repeatedly get put into positions requiring good communication skills. We encounter project managers who cannot make decisions. The fit of a particular person’s personality profile to the role given them has a large effect on the outcome of the project. This fit should be adjusted for the person’s personal growth interests, but often that position is the only career improvement path for the individual.

Failing conservatively. It has been fairly well established that people prefer to fail conservatively than to succeed by taking a risk [Pi]. This may explain why waterfall development is still in use after decades of trouble and the advent of spiral, incremental and iterative staging techniques.

Changing habits. Getting people to change their habits is the single hardest thing I know. And yet, people do change their habits almost spontaneously, given a shift in value systems. I have seen this done deliberately and consciously perhaps twice in 20 years, and it is impressive. We could do well to understand this phenomenon better.

“Small heads”. Even expert designers repeatedly say “I can only keep a small amount of information in my head”. Several documentation techniques have tried to leverage this weakness, but none have combined this with use of low-precision work products and high interpersonal communication.

I am certain there are other characteristics of people that strongly affect the way projects operate and the recommendations that we should be making. These are the ones I currently use.

4 – Conclusion

The fundamental characteristics of “people” have a first-order effect on software development, not a lower-order effect. Consequently, understanding this first-order effect should become a first-order research agenda item, and not neglected as a second-order item. I suggest that this field of study become a primary area in the field “software engineering” for the next 20-50 years.

I presented several characteristics of people that have recognizable effects on methodology design.

The first is that we are sensitive to communication timing and modalities. The prediction is that physical proximity and ease of communication has dominant effect.

The second is that people tend to inconsistency. The prediction is that methodologies requiring disciplined consistency are fragile in practice.

> The third is that people vary, not just daily, but from group to group. Methodologies don't currently, but do need to deal with this cultural variation.
>
> The fourth is that people like to be good citizens, are good at looking around and taking initiative. These combine to form that common success factor, "a few good people stepped in at key moments."

Being good at communicating and looking around counter inconsistency, leading to the prediction that methodologies can make good use of low-precision artifacts whose gaps are covered by personal communication. Project histories also support this prediction, subject to the normalization of an adequately skilled staff, including management.

In the title, I refer to people as "components". That is how people are treated in the process / methodology design literature. The mistake in this approach is that "people" are highly variable and non-linear, with unique success and failure modes. Those factors are first-order, not negligible factors. Failure of process and methodology designers to account for them contributes to the sorts of unplanned project trajectories we so often see.

Finally, it should be clear that we could do with much more research on this topic. I hope that many more researchers will pick up the challenge now than have since Weinberg wrote his version of this article 30 years ago. In particular, I look forward to psychological and ethnographic research to discover other first-order effects not mentioned here.

References

[B87] Beck, K. and Cunningham, W., "A laboratory for teaching object-oriented thinking", Proceedings of the OOPSLA Conference, 1987, ACM Sigplan Oct., 1987, pp.1-7.

[B99] Beck, K., Extreme Programming Explained: Embrace Change, Addison Wesley Longman, 1999.

[Br] Britcher, R., The Limits of Software, Addison Wesley, 1999.

[Ch] Christensen, M., *et al.*, "The M.A.D. experience: multiperspective application development in evolutionary prototyping", in the ECOOP'98 proceedings, Lecture Notes in Computer Science, Vol. 1445, Goos, G., Hartmanis, J, van Leeuwen, J., eds., 1998, pp. 13-41.

[Ci] Citrin, W., Cockburn A., von Kaenel, J., Hauser, R., "Using formalized temporal message†'flow diagrams," Software Practice and Experience, 1995.

[Co94] Cockburn A., "In search of methodology," Object Magazine, July 1994.

[Co95] Cockburn A., "Unraveling incremental development," Object Magazine, Jan. 1995.

[Co98] Cockburn A., Surviving Object-Oriented Projects, Addison Wesley, 1998.

[Co98p] Cockburn A., Position statement for "Software development and process" panel, ECOOP '98, online at http://members.aol.com/acockburn/papers/Ecoop98panel.htm

[Co00]Cockburn A., Crystal(Clear): A human-powered software development methodology for small teams, Addison Wesley, in prep. Online at http://members.aol.com/humansandt/crystal/ clear.

[Cs] Csikszentmihalyi, M., Flow: The Psychology of Optimal Experience, Harper Perennial, 1990

[C3] The C3 Team, "Chrysler goes to 'Extremes'", in Distributed Object Computing, October, 1998, pp. 24-28.

[Dm] DeMarco, T., Lister, T., Peopleware 2nd Edition, Dorset House, 1999.

[EP] Extreme Programming, web notes, start from www.extremeprogramming.com.

[Gr] Gries, D, The Science of Programming, Springer Verlag, 1987.

[Ha] Harel, D., Politi, M., Modeling Reactive Systems with Statecharts, McGraw-Hill, 1998.

[Hi] Highsmith, J., Adaptive Software Development, Dorset House, 1999.

[Ho] Hovenden, F., "An ethnographic account of writing use cases on an IT project", Humans and Technology Technical Memo, 1999.10.30, online at http://members.aol.com/humansandt/papers/ethno1.htm.

[Hu] Humphrey, W., A Discipline for Softwrae Engineering, Addison Wesley, 1995.

[Je] Jeffries, R., “Extreme testing”, in Software Testing and Quality Engineering, March/April, 1999, pp. 23-26.

[J-L] Johnson-Laird, P. and Byrne, R. Deduction, Lawrence Erlbaum Associates, 1991.

[La] Lave, J., Situation Learning: Legitimate Peripheral Participation, Cambridge Press, 1991.

[Mi] Mills, H., Dyer, M., Linger, R., “Cleanroom Software Engineering,” IEEE Software Vol 2 (1984) No. 9, 19-24.

[NASA] JSC38609, “Deorbit flight software lessons learned”, NASA Johnson Space Center, Jan 19, 1998 online at

[Pi] Piatelli-Palmarini, M., Inevitable Illusions : How Mistakes of Reason Rule Our Minds, Wiley & Sons, 1996.

[Pl] Plowman, L., “The interfunctionality of talk and text”, Computer Support of Cooperative Work, vol. 3, 1995, pp.229-246.

[Si] Sillince, J.A., “A model of social, emotional and symbolic aspects of computer-mediated communication within organizations”, Computer Support of Cooperative Work vol. 4, 1996, pp. 1-31.

[Web] Webb, D., Humphrey, W., “Using TSP on the TaskView Project”, in *CrossTalk, The Journal of Defense Software Engineering*, Feb 1999, pp. 3-10, online at http://www.stsc.hill.af.mil/crosstalk/1999/feb/webb.asp

[Wei] Weinberg, J., The Psychology of Computer Programming, Silver Edition, Dorset House, 1998.

[Yo] Yourdon, E., Death March Projects, Prentice Hall, 1997.