Introduction

In the 1960s, customer involvement and collaboration weren't problems. Then, the computers were less powerful, the users were fewer, the applications were simpler, and the idea of milestones was still unknown. Developers used short iterations of one or two days. They did meet with the customer and together sketch out on paper what he or she wanted. They'd discuss the problem until developers understood it. Then developers would go to their desk, design and code the solution, punch up the cards, and compile the program. Once the compile and link were clean, they did run some test data against the program. Then they'd return to the customer and ask, "Is this what you wanted?" People didn't realize it at the time, but this was heaven.

As the applications and technology became more complex and the number of stakeholders in a project increased, practices were inserted to coordinate the communication among the increased numbers of participants. For instance, because many stakeholders were involved, we began to collect all of their requirements prior to starting development. We felt that the system should implement the sum of their respective requirements. Because documentation was such an inadequate medium of communicating, we started to use pictures to communicate, supporting these pictures with text. And because pictures were imprecise, we developed modeling techniques to formally represent the pictures. Each step drove a wedge between the stakeholders and the developers. We went from face-to-face communication to documentation. We went from quick turnaround to lengthy requirements-gathering phases. We went from simple language to artifacts that were arcane and highly specialized.

In retrospect, the more we improved the practice of software engineering, the further we widened the gap between stakeholders and developers. The last step in the estrangement was the introduction of *waterfall methodology*, which embodies all the flaws of sequential development. Waterfall methodology gathers all the requirements, then creates the design, then writes the code, then develops and runs tests, and finally implements the system. Between each of these steps, or phases, were review meetings. Stakeholders were invited to these meetings to review the progress to date. At these meetings, developers and managers would ask customers "Do these requirements that we've gathered and the models that demonstrate them constitute a full and accurate representation of what you want? Because once you say yes, it will be increasingly expensive for you to change your mind!" As you can

see, this wording implies a contract between the customers and the developers. By this point, there was little in-person collaboration; in its place were contracts that said, “If you agree that what I showed you is the complete description of your requirements, we will proceed. If you don’t agree, we’ll continue to develop requirements until you give up!”

Einstein’s definition of insanity: doing the same thing over and over and expecting different results. Surprisingly, this approach is common. If a project being managed by a defined approach (Waterfall) fails, people often assume that the project failed because the defined approach wasn’t adhered to rigorously enough. They conclude that all that is needed for the project to succeed is increased control and project definition.

Iterative Development

Software development is a cooperative game of invention and communication.
Humans working together, building something they don't quite understand. Done well, the result is breathtaking; done poorly, dross.

Purely people factors predict project success, overriding choice of process or technology.
There is still a lot of resistance in our industry to this idea.
A well-functioning team of adequate people will complete a project almost regardless of the process or technology they are asked to use (although the process and technology might help or hinder them along the way).
Dave A. Thomas, founder of Object Technology International, a company with a long record of successful projects, summarized his success formula: "Some people deliver software; some don't. I hire those that have delivered."

The very reason for incremental and iterative strategies is to allow for people's inevitable mistakes to be discovered relatively early and repaired in a tidy manner.
That people make mistakes should really not be any surprise to us. And yet, some managers seem genuinely surprised when the development team announces a plan to work according to an incremental or iterative process. I have heard of managers saying things like

"What do you mean; you don't know how long it will take?"

Or

> "What do you mean; you plan to do it wrong the first time? I can go out and hire someone who will promise to do it right the first time."

In other words, the manager is saying that he expects the development team not to make any major mistakes or to learn anything new on the project.

One can find people who promise to get things right the first time, but one is unlikely to find people who actually get things right the first time. People make mistakes in estimation, requirements, design, typing, proofreading, installing, testing . . . and everything else they do. There is no escape. *We must accept that mistakes will be made and use processes that adjust to the fact of mistakes.*

Given how obvious it is that people make mistakes, the really surprising thing is that managers still refuse to use incremental and iterative strategies. This is because

- Humans prefer to fail conservatively rather than risk succeeding differently
- Humans have difficulty changing working habits.

Consider a manager faced with changing from waterfall to incremental or iterative scheduling. The waterfall strategy is accepted as a normal, conservative way of doing business, even though many people think it is faulty. The manager has used waterfall strategy several times, with varying success. Now, one of his junior people comes to him with a radically different approach. He sees some significant dangers in the new approach. His reputation is riding on this next project. Does he use the normal, conservative strategy or try out the risky new strategy?
Odds are that he will use the normal, conservative strategy, a "guaranteed" standard outcome, rather than one that might work but might blow up in strange ways.

This characteristic, "preferring to fail conservatively rather than to risk succeeding differently," gets coupled with people's fear of rejection and the difficulty they have in building new work habits. So managers continue to use the long-abused one-pass waterfall development process. Based on this line of thinking, I expect that people will continue to use the waterfall process even in the presence of mounting evidence against it and increasing evidence supporting incremental and iterative development. Use of the waterfall process is anchored in a failure mode.