Iterative and Incremental Development: A Brief History

Although many view iterative and incremental development as a modern practice, its application dates as far back as the mid-1950s. Prominent software-engineering thought leaders from each succeeding decade supported IID practices, and many large projects used them successfully.

Craig Larman
Valtech

Victor R. Basili
University of Maryland

As agile methods become more popular, some view iterative, evolutionary, and incremental software development—a cornerstone of these methods—as the "modern" replacement of the waterfall model, but its practiced and published roots go back decades. Of course, many software-engineering students are aware of this, yet surprisingly, some commercial and government organizations still are not.

This description of projects and individual contributions provides compelling evidence of iterative and incremental development's (IID's) long existence. Many examples come from the 1970s and 1980s—the most active but least known part of IID's history. We are mindful that the idea of IID came independently from countless unnamed projects and the contributions of thousands and that this list is merely representative. We do not mean this article to diminish the unsung importance of other IID contributors.

We chose a chronology of IID projects and approaches rather than a deep comparative analysis. The methods varied in such aspects as iteration length and the use of time boxing. Some attempted significant up-front specification work followed by incremental time-boxed development, while others were more classically evolutionary and feedback driven. Despite their differences, however, all the approaches had a common theme—to avoid a single-pass sequential, document-driven, gated-step approach.

Finally, a note about our terminology: Although some prefer to reserve the phrase "iterative development" merely for rework, in modern agile methods the term implies not just revisiting work, but also evolutionary advancement—a usage that dates from at least 1968.

PRE-1970

IID grew from the 1930s work of Walter Shewhart,[1] a quality expert at Bell Labs who proposed a series of short "plan-do-study-act" (PDSA) cycles for quality improvement. Starting in the 1940s, quality guru W. Edwards Deming began vigorously promoting PDSA, which he later described in 1982 in *Out of the Crisis*.[2] Tom Gilb[3] and Richard Zultner[4] also explored PDSA application to software development in later works.

The X-15 hypersonic jet was a milestone 1950s project applying IID,[5] and the practice was considered a major contribution to the X-15's success. Although the X-15 was not a software project, it is noteworthy because some personnel—and hence, IID experience—seeded NASA's early 1960s Project Mercury, which did apply IID in software. In addition, some Project Mercury personnel seeded the IBM Federal Systems Division (FSD), another early IID proponent.

Project Mercury ran with very short (half-day) iterations that were time boxed. The development team conducted a technical review of all changes, and, interestingly, applied the Extreme Programming practice of test-first development, planning and writing tests before each micro-increment. They also practiced top-down development with stubs.

Published by the IEEE Computer Society

The recollections of Gerald M. Weinberg, who worked on the project, provide a window into some practices during this period. In a personal communication, he wrote:

> We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's Service Bureau Corporation]. He was a colleague of John von Neumann, so perhaps he learned it there, or assumed it as totally natural. I do remember Herb Jacobs (primarily, though we all participated) developing a large simulation for Motorola, where the technique used was, as far as I can tell, indistinguishable from XP.
>
> When much of the same team was reassembled in Washington, DC in 1958 to develop Project Mercury, we had our own machine and the new Share Operating System, whose symbolic modification and assembly allowed us to build the system incrementally, which we did, with great success. Project Mercury was the seed bed out of which grew the IBM Federal Systems Division. Thus, that division started with a history and tradition of incremental development.
>
> All of us, as far as I can remember, thought waterfalling of a huge project was rather stupid, or at least ignorant of the realities… I think what the waterfall description did for us was make us realize that we were doing something else, something unnamed except for "software development."

The earliest reference we found that specifically focused on describing and recommending iterative development was a 1968 report from Brian Randell and F.W. Zurcher at the IBM T.J. Watson Research Center.[6] M.M. Lehman later described Randell and Zurcher's work and again promoted iterative development in his September 1969 internal report to IBM management on development recommendations:[7]

> The basic approach recognizes the futility of separating design, evaluation, and documentation processes in software-system design. The design process is structured by an expanding model seeded by a formal definition of the system, which provides a first, executable, functional model. It is tested and further expanded through a sequence of models, that develop an increasing amount of function and an increasing amount of detail as to how that function is to be executed. Ultimately, the model becomes the system.

Another 1960s reference comes from Robert Glass:[8]

> It is the opinion of the author that incremental development is worthwhile, [it] leads to a more thorough system shakedown, avoids implementer and management discouragement.

THE SEVENTIES

In his well-known 1970 article, "Managing the Development of Large Software Systems," Winston Royce shared his opinions on what would become known as the waterfall model, expressed within the constraints of government contracting at that time.[9] Many—incorrectly—view Royce's paper as the paragon of single-pass waterfall. In reality, he recommended an approach somewhat different than what has devolved into today's waterfall concept, with its strict sequence of requirements analysis, design, and development phases. Indeed, Royce's recommendation was to do it *twice*:

> If the computer program in question is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned.

Royce further suggested that a 30-month project might have a 10-month pilot model and justified its necessity when the project contains novel elements and unknown factors (hardly a unique case). Thus, we see hints of iterative development, feedback, and adaptation in Royce's article. This iterative feedback-based step has been lost in most descriptions of this model, although it is clearly not classic IID.

What did Royce think about the waterfall versus IID when he learned of the latter approach? In a personal communication, Walker Royce, his son and a contributor to popular IID methods in the 1990s, said this of his father and the paper:

> He was always a proponent of iterative, incremental, evolutionary development. His paper described the waterfall as the simplest description, but that it would not work for all but the most straightforward projects. The rest of his paper describes [iterative practices] within the context of the 60s/70s government-contracting models (a serious set of constraints).

"We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's Service Bureau Corporation]."

The first major documented IBM FSD IID application was the life-critical command and control system for the first US Trident submarine.

This was an ironic insight, given the influence this paper had as part of the bulwark promoting a strict sequential life cycle for large, complex projects.

The next earliest reference comes from Harlan Mills, a 1970s software-engineering thought leader who worked at the IBM FSD. In his well-known "Top-Down Programming in Large Systems," Mills promoted iterative development. In addition to his advice to begin developing from top-level control structures downward, perhaps less appreciated was the related life-cycle advice Mills gave for building the system via iterated expansions:[10]

> … it is possible to generate a sequence of intermediate systems of code and functional subspecifications so that at every step, each [intermediate] system can be verified to be correct…

Clearly, Mills suggested iterative refinement for the development phase, but he did not mention avoiding a large up-front specification step, did not specify iteration length, and did not emphasize feedback and adaptation-driven development from each iteration. He did, however, raise these points later in the decade. Given his employment at the IBM FSD, we suspect Mills's exposure to the more classic IID projects run there in the early 1970s influenced his thought, but we could not confirm this with colleagues.

Early practice of more modern IID (feedback-driven refinement with customer involvement and clearly delineated iterations) came under the leadership of Mike Dyer, Bob McHenry, and Don O'Neill and many others during their tenure at IBM FSD. The division's story is fascinating because of the extent and success of its IID use on large, life-critical US Department of Defense (DoD) space and avionics systems during this time.

The first major documented IBM FSD application of IID that we know of was in 1972. This was no toy application, but a high-visibility life-critical system of more than 1 million lines of code—the command and control system for the first US Trident submarine. O'Neill was project manager, and the project included Dyer and McHenry. O'Neill conceived and planned the use of IID (which FSD later called "integration engineering") on this project; it was a key success factor, and he was awarded an IBM Outstanding Contribution Award for the work. (Note that IBM leadership visibly approved of IID methods.)

The system had to be delivered by a certain date or FSD would face a $100,000 per day late penalty. The team organized the project into four time-boxed iterations of about six months each. There was still a significant up-front specification effort, and the iteration was longer than normally recommended today. Although some feedback-driven evolution occurred in the requirements, O'Neill noted that the IID approach was also a way to manage the complexity and risks of large-scale development.[11]

Also in 1972, an IBM FSD competitor, TRW, applied IID in a major project—the $100 million TRW/Army Site Defense software project for ballistic missile defense. The project began in February 1972, and the TRW team developed the system in five iterations. Iteration 1 tracked a single object, and by iteration 5, a few years later, the system was complete. The iterations were not strictly time boxed, and there was significant up-front specification work, but the team refined each iteration in response to the preceding iteration's feedback.[12]

As with IBM FSD, TRW (where Royce worked) was an early adopter of IID practices. Indeed, Barry Boehm, the originator of the IID spiral model in the mid-1980s, was chief scientist at TRW.

Another mid-1970s extremely large application of IID at FSD was the development of the Light Airborne Multipurpose System, part of the US Navy's helicopter-to-ship weapon system. A four-year 200-person-year effort involving millions of lines of code, LAMPS was incrementally delivered in 45 time-boxed iterations (one month per iteration). This is the earliest example we found of a project that used an iteration length in the range of one to six weeks, the length that current popular IID methods recommend. The project was quite successful: As Mills wrote, "Every one of those deliveries was on time and under budget."[13]

In 1975, Vic Basili and Joe Turner published a paper about iterative enhancement that clearly described classic IID:[14]

> The basic idea behind iterative enhancement is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented. At each iteration, design modifications are made along with adding new functional capabilities.

The paper detailed successful IID application to the development of extendable compilers for a family of application-specific programming languages on a variety of hardware architectures. The project team developed the base system in 17 iterations over 20 months. They analyzed each iteration from both the user's and developer's points of view and used the feedback to modify both the language requirements and design changes in future iterations. Finally, they tracked measures, such as coupling and cohesion, over the multiple iterations.

In 1976, Tom Gilb published *Software Metrics* (coining the term), in which he discussed his IID practice—evolutionary project management—and introduced the terms "evolution" and "evolutionary" to the process lexicon. This is the earliest book we could find that had a clear IID discussion and promotion, especially of evolutionary delivery:[3]

> "Evolution" is a technique for producing the appearance of stability. A complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a "retreat" possibility to a previous successful step upon failure. You have the opportunity of receiving some feedback from the real world before throwing in all resources intended for a system, and you can correct possible design errors…

The book marked the arrival of a long-standing and passionate voice for evolutionary and iterative development. Gilb is one of the earliest and most active IID practitioners and promoters. He began the practice in the early 1960s and went on to establish several IID milestones. His material was probably the first with a clear flavor of agile, light, and adaptive iteration with quick results, similar to that of newer IID methods.

By 1976, Mills had strengthened his IID message:[15]

> Software development should be done incrementally, in stages with continuous user participation and replanning and with design-to-cost programming within each stage.

Using a three-year inventory system project as a backdrop, he challenged the idea and value of upfront requirements or design specification:

> ...there are dangers, too, particularly in the conduct of these [waterfall] stages in sequence, and not in iteration-i.e., that development is done in an open loop, rather than a closed loop with user feedback between iterations. The danger in the sequence [waterfall approach] is that the project moves from being grand to being grandiose, and exceeds our human intellectual capabilities for management and control.

Tom Gilb introduced the terms "evolution" and "evolutionary" to the process lexicon.

And perhaps reflecting several years of seeing IID in action at FSD, Mills asked, "...why do enterprises tolerate the frustrations and difficulties of such [waterfall] development?"

In 1977, FSD incorporated the Trident IID approach, which included integrating all software components at the end of each iteration into its software-engineering practices—an approach McHenry dubbed "integration engineering." Some Trident team members and Mills were key advisers in this incorporation effort.[16] Integration engineering spread to the 2,500 FSD software engineers, and the idea of IID as an alternative to the waterfall stimulated substantial interest within IBM's commercial divisions and senior customer ranks and among its competitors.

Although unknown to most software professionals, another early and striking example of a major IID success is the very heart of NASA's space shuttle software—the primary avionics software system, which FSD built from 1977 to 1980. The team applied IID in a series of 17 iterations over 31 months, averaging around eight weeks per iteration.[17] Their motivation for avoiding the waterfall life cycle was that the shuttle program's requirements changed during the software development process. Ironically (in hindsight), the authors sound almost apologetic about having to forego the "ideal" waterfall model for an IID approach:

> Due to the size, complexity, and evolutionary [changing requirements] nature of the program, it was recognized early that the ideal software development life cycle [the waterfall model] could not be strictly applied...However, an implementation approach (based on small incremental releases) was devised for STS-1 which met the objectives by applying the ideal cycle to small elements of the overall software package on an iterative basis.

The shuttle project also exhibited classic IID practices: time-boxed iterations in the eight-week range, feedback-driven refinement of specifications, and so on.

The first IID discussion in the popular press that we could find was in 1978, when Tom Gilb began publishing a column in the UK's *Computer Weekly*.

The IID practice of evolutionary prototyping was commonly used in 1980s efforts to create artificial intelligence systems.

The column regularly promoted IID, as well as evolutionary project management and delivery. In his 6 April 1978 column, Gilb wrote,

> Management does not require firm estimates of completion, time, and money for the entire project. Each [small iterative] step must meet one of the following criteria (priority order): either (a) give planned return on investment payback, or, if impossible, then (b) give breakeven (no loss); or, at least, (c) some positive user benefit measurably; or, at least (d) some user environment feedback and learning.

Another discussion of incremental development, although published in 1984, refers to a System Development Corp. project to build an air defense system, which began in 1977 and finished in 1980. The project combined significant up-front specifications with incremental development and builds. Ostensibly, the project was meant to fit within DoD single-pass waterfall standards, with testing and integration in the last phase. Carolyn Wong comments on the unrealism of this approach and the team's need to use incremental development:[18]

> The [waterfall] model was adopted because software development was guided by DoD standards…In reality, software development is a complex, continuous, iterative, and repetitive process. The [waterfall model] does not reflect this complexity.

THE EIGHTIES

In 1980 Weinberg wrote about IID in "Adaptive Programming: The New Religion," published in Australasian *Computerworld*. Summarizing the article, he said, "The fundamental idea was to build in small increments, with feedback cycles involving the customer for each." A year later, Tom Gilb wrote in more detail about evolutionary development.[19]

In the same year, Daniel McCracken and Michael Jackson promoted IID and argued against the "stultifying waterfall" in a chapter within a software engineering and design text edited by William Cotterman. The chapter's title, "A Minority Dissenting Position," underscored the subordinate position of IID to the waterfall model at the time.[20] Their arguments continued in "Life-Cycle Concept Considered Harmful,"[21] a 1982 twist on Edsger Dijkstra's late 1960s classic "Go To Statement Considered Harmful."[22] (The use of "life cycle" as a synonym for waterfall during this period suggests its unquestioned dominance. Contrast this to its qualified use in the 1990s, "sequential life cycle" or "iterative life cycle.")

In 1982, William Swartout and Robert Balzer argued that specification and design have a necessary interplay, and they promoted an iterative and evolutionary approach to requirements engineering and development.[23] The same year also provided the earliest reference to a very large application successfully built using evolutionary prototyping, an IID approach that does not usually include time-boxed iterations. The $100 million military command and control project was based on IBM's Customer Information Control System technology.[24]

In 1983, Grady Booch published *Software Engineering with Ada*,[25] in which he described an iterative process for growing an object-oriented system. The book was influential primarily in the DoD development community, but more for the object-oriented design method than for its iterative advice. However, Booch's later 1990s books that covered IID found a large general audience, and many first considered or tried iterative development through their influence.

The early 1980s was an active period for the (attempted) creation of artificial intelligence systems, expert systems, and so on, especially using Lisp machines. A common approach in this community was the IID practice of evolutionary prototyping.[26]

In another mid-1980s questioning of the sequential life cycle, Gilb wrote "Evolutionary Delivery versus the 'Waterfall Model.'" In this paper, Gilb promoted a more aggressive strategy than other IID discussions of the time, recommending frequent (such as every few weeks) delivery of useful results to stakeholders.[27]

A 1985 landmark in IID publications was Barry Boehm's "A Spiral Model of Software Development and Enhancement," (although the more frequent citation date is 1986).[28] The spiral model was arguably not the first case in which a team prioritized development cycles by risk: Gilb and IBM FSD had previously applied or advocated variations of this idea, for example. However, the spiral model did formalize and make prominent the risk-driven-iterations concept and the need to use a discrete step of risk assessment in each iteration.

In 1986, Frederick Brooks, a prominent software-engineering thought leader of the 1970s and 1980s, published the classic "No Silver Bullet" extolling the advantages of IID:[29]

> Nothing in the past decade has so radically changed my own practice, or its effectiveness [as incremental development].

Commenting on adopting a waterfall process, Brooks wrote

> Much of present-day software acquisition procedure rests upon the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. I think this assumption is fundamentally wrong, and that many software acquisition problems spring from that fallacy.

Perhaps summing up a decade of IID-promoting messages to military standards bodies and other organizations, Brooks made his point very clear in his keynote speech at the 1995 International Conference on Software Engineering: "The waterfall model is wrong!"

In 1986, David Parnas and Paul Clements published "A Rational Design Process: How and Why to Fake It."[30] In it, they stated that, although they believe in the ideal of the waterfall model (thorough, correct, and clear specifications before development), it is impractical. They listed many reasons, including (paraphrased)

- A system's users seldom know exactly what they want and cannot articulate all they know.
- Even if we could state all requirements, there are many details that we can only discover once we are well into implementation.
- Even if we knew all these details, as humans, we can master only so much complexity.
- Even if we could master all this complexity, external forces lead to changes in requirements, some of which may invalidate earlier decisions.

and commented that for all these reasons, "the picture of the software designer deriving his design in a rational, error-free way from a statement of requirements is quite unrealistic."

In 1987, TRW launched a four-year project to build the Command Center Processing and Display System Replacement (CCPDS-R), a command and control system, using IID methods. Walker Royce described the effort in 60 pages of detail.[31] The team time-boxed six iterations, averaging around six months each. The approach was consistent with what would later become the Rational Unified Process (to which Royce contributed): attention to high risks and the core architecture in the early iterations.

Bill Curtis and colleagues published a particularly agile-relevant paper during this decade,[32] reporting results on research into the processes that influenced 19 large projects. The authors identified that the prescriptive waterfall model attempted to satisfy management accountability goals, but they did not describe how projects successfully ran. The paper also noted that successful development emphasizes a cyclic learning process with high attention to people's skills, common vision, and communication issues, rather than viewing the effort as a sequential "manufacturing process." As the authors state,

> The conclusion that stands out most clearly from our field study observations is that the process of developing large software systems must be treated, at least in part, as a learning and communication process.

In 1987, as part of the IBM FSD Software Engineering Practices program, Mills, Dyer, and Rick Linger continued the evolution of IID with the Cleanroom method, which incorporated evolutionary development with more formal methods of specification and proof, reflecting Mills's strong mathematical influences.[33]

The Cleanroom method incorporated evolutionary development with more formal methods of specification and proof.

By the late 1980s, the DoD was experiencing significant failure in acquiring software based on the strict, document-driven, single-pass waterfall model that DoD-Std-2167 required. A 1999 review of failure rates in a sample of earlier DoD projects drew grave conclusions: "Of a total $37 billion for the sample set, 75% of the projects failed or were never used, and only 2% were used without extensive modification."[34] Consequently, at the end of 1987, the DoD changed the waterfall-based standards to allow IID, on the basis of recommendations in an October 1987 report from the Defense Science Board Task Force on Military Software, chaired by Brooks. The report recommended replacing the waterfall, a failing approach on many large DoD projects, with iterative development:

> DoD-Std-2167 likewise needs a radical overhaul to reflect modern best practice. Draft 2167A is a step, but it does not go nearly far enough. As drafted, it continues to reinforce exactly the document-driven, specify-then-build approach that lies at the heart of so many DoD software problems....

Tom Gilb's *Principles of Software Engineering Management* was the first book with substantial chapters dedicated to IID discussion and promotion.

> In the decade since the waterfall model was developed, our discipline has come to recognize that [development] requires iteration between the designers and users.

Finally, in a section titled "Professional Humility and Evolutionary Development" (humility to accept that the 2167's goals—get the specifications accurate without incremental implementation and feedback—was not possible), the report stated:

> Experience with confidently specifying and painfully building mammoths has shown it to be simplest, safest, and even fastest to develop a complex software system by building a minimal version, putting it into actual use, and then adding functions [and other qualities] according to the priorities that emerge from actual use.
>
> Evolutionary development is best technically, and it saves time and money.

Both DoD overseers and contractors often view the updated DoD-Std-2167A, released in February 1988, as the epitome of a waterfall specification. Yet, its authors actually wanted it to be an amendment (hence the A) for life-cycle neutrality that allowed IID alternatives to the waterfall:

> This standard is not intended to specify or discourage the use of any particular software development method. The contractor is responsible for selecting software development methods (for example, rapid prototyping) that best support the achievement of contract requirements.

Despite this intent, many (justifiably) interpreted the new standard as containing an implied preference for the waterfall model because of its continued document-driven milestone approach.

Ironically, in a conversation nearly a decade later, the principal creator of DoD-Std-2167 expressed regret for creating the strict waterfall-based standard. He said that at the time he knew of the single-pass document-driven waterfall model, and others he questioned advised it was excellent, as did the literature he examined, but he had not heard of iterative development. In hindsight, he said he would have made a strong recommendation for IID rather than the waterfall model.

In 1988, Gilb published *Principles of Software Engineering Management*, the first book with substantial chapters dedicated to IID discussion and promotion.[35] In it he reiterated and expanded on the IID material from *Software Metrics*. Gilb described the Evo method, distinguished by frequent evolutionary delivery and an emphasis on defining quantified measurable goals and then measuring the actual results from each time-boxed short iteration.

1990 TO THE PRESENT

By the 1990s, especially the latter half, public awareness of IID in software development was significantly accelerating. Hundreds of books and papers were promoting IID as their main or secondary theme. Dozens more IID methods sprang forth, which shared an increasing trend to time-boxed iterations of one to six weeks.

In the 1970s and 1980s, some IID projects still incorporated a preliminary major specification stage, although their teams developed them in iterations with minor feedback. In the 1990s, in contrast, methods tended to avoid this model, preferring less early specification work and a stronger evolutionary analysis approach.

The DoD was still experiencing many failures with "waterfall-mentality" projects. To correct this and to reemphasize the need to replace the waterfall model with IID, the Defense Science Board Task Force on Acquiring Defense Software Commercially, chaired by Paul Kaminski, issued a report in June 1994 that stated simply, "DoD must manage programs using iterative development. Apply evolutionary development with rapid deployment of initial functional capability."

Consequently, in December 1994, Mil-Std-498 replaced 2167A. An article by Maj. George Newberry summarizing the changes included a section titled "Removing the Waterfall Bias," in which he described the goal of encouraging evolutionary acquisition and IID:[36]

> Mil-Std-498 describes software development in one or more incremental builds. Each build implements a specified subset of the planned capabilities. The process steps are repeated for each build, and within each build, steps may be overlapping and iterative.

Mil-Std-498 itself clearly states the core IID practices of evolving requirements and design incrementally with implementation:

> If a system is developed in multiple builds, its requirements may not be fully defined until the final build.... If a system is designed in multiple builds, its design may not be fully defined until the final build.

Meanwhile, in the commercial realm, Jeff Sutherland and Ken Schwaber at Easel Corp. had started to apply what would become known as the Scrum method, which employed time-boxed 30-day iterations. The method took inspiration from a Japanese IID approach used for nonsoftware products at Honda, Canon, and Fujitsu in the 1980s; from Shashimi ("slices" or iterations); and from a version of Scrum described in 1986.[37] A 1999 article described their later refinements to Scrum.[38]

In January 1994, a group of 16 rapid application development (RAD) practitioners met in the UK to discuss the definition of a standard iterative process to support RAD development. The group drew inspiration from James Martin's RAD teachings. Martin, in turn, had taken his inspiration from the time-boxing work at Dupont, led by Scott Shultz in the mid-1980s. The RAD group's process definition would eventually become the Dynamic Systems Development Method (DSDM), an IID method that predictably had more early advocates in Europe and has since spread.[39]

In the early 1990s, a consortium of companies began a project to build a new-generation Canadian Automated Air Traffic Control System (CAATS) using a risk-driven IID method. The project, under the process leadership of Philippe Kruchten, used a series of six-month iterations, relatively long by today's standards. The project was a success, despite its prior near-failure applying a waterfall approach.[40]

In the mid-1990s, many contributors within Rational Corp. (including Kruchten and Walker Royce) and its clients created the Rational Unified Process, now a popular IID method. A 1995 milestone was the public promotion of the daily build and smoke test, a widely influential IID practice institutionalized by Microsoft that featured a one-day micro-iteration.[41]

In 1996, Kent Beck joined the Chrysler C3 payroll project. It was in this context that the full set of XP practices matured, with some collaboration by Ron Jeffries and inspiration from earlier 1980s work at Tektronix with Ward Cunningham. XP went on to garner significant public attention because of its emphasis on communication, simplicity, and testing, its sustainable developer-oriented practices, and its interesting name.[42]

In 1997, a project to build a large logistics system in Singapore, which had been running as a waterfall project, was facing failure. With the collaboration of Peter Coad and Jeff De Luca, the team resurrected it and ran it as a successful IID project. DeLuca created an overall iterative process description, Feature-Driven Development (FDD), that also incorporated ideas from Coad.[43]

In 1998, the Standish Group issued its widely cited "CHAOS: Charting the Seas of Information Technology," a report that analyzed 23,000 projects to determine failure factors. The top reasons for project failure, according to the report, were associated with waterfall practices. It also concluded that IID practices tended to ameliorate the failures. One of the report's key conclusions was to adopt IID:

> Research also indicates that smaller time frames, with delivery of software components early and often, will increase the success rate. Shorter time frames result in an iterative process of design, prototype, develop, test, and deploy small elements.

In 2000, DoD replaced Mil-Std-498 with another software acquisition standard, DoD 5000.2, which again recommended adopting evolutionary acquisition and the use of IID:

> There are two approaches, evolutionary and single step [waterfall], to full capability. An evolutionary approach is preferred. … [In this] approach, the ultimate capability delivered to the user is divided into two or more blocks, with increasing increments of capability...software development shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development.

In 2001, Alan MacCormack reported a study of key success factors in recent projects; first among these was adopting an IID life cycle:[44]

> Now there is proof that the evolutionary approach to software development results in a speedier process and higher-quality products. […] The iterative process is best captured in the evolutionary delivery model proposed by Tom Gilb.

In February 2001, a group of 17 process experts—representing DSDM, XP, Scrum, FDD, and others—interested in promoting modern, simple IID methods and principles met in Utah to discuss common ground. From this meeting came the Agile Alliance (www.agilealliance.org) and the now

XP garnered significant public attention because of its emphasis on communication, simplicity, and testing, and its sustainable developer-oriented practices.

popular catch phrase "agile methods," all of which apply IID. And in 2002, Alistair Cockburn, one of the participants, published the first book under the new appellation.[45]

In a typical quip, H.L. Mencken said, "For every complex problem, there is a solution that is simple, neat, and wrong." In the history of science, it is the norm that simplistic but inferior ideas first hold the dominant position, even without supporting results. Medicine's four humors and related astrological diagnosis and prescription dominated Europe for more than a millennium, for example.

Software development is a very young field, and it is thus no surprise that the simplified single-pass and document-driven waterfall model of "requirements, design, implementation" held sway during the first attempts to create the ideal development process. Other reasons for the waterfall idea's early adoption or continued promotion include:

- It's simple to explain and recall. "Do the requirements, then design, and then implement." IID is more complex to understand and describe. Even Winston Royce's original two-iteration waterfall immediately devolved into a single sequential step as other adopters used it and writers described it.
- It gives the illusion of an orderly, accountable, and measurable process, with simple document-driven milestones (such as "requirements complete").
- It was promoted in many software engineering, requirements engineering, and management texts, courses, and consulting organizations. It was labeled appropriate or ideal, seemingly unaware of this history or of the statistically significant research evidence in favor of IID.

This brief history shows that IID concepts have been and are a recommended practice by prominent software-engineering thought leaders of each decade, associated with many successful large projects, and recommended by standards boards.

Yet, even though the value of IID is well known among literate, experienced software engineers, some commercial organizations, consulting companies, and standards bodies still promote a document-driven single-pass sequential life cycle as the ideal. We conclude with this recommendation: In the interest of promoting greater project success and saving taxpayer or investor dollars, let's continue efforts to educate and promote the use of IID methods. ■

References

1. W. Shewhart, *Statistical Method from the Viewpoint of Quality Control*, Dover, 1986 (reprint from 1939).
2. W.E. Deming, *Out of the Crisis*, SPC Press, 1982; reprinted in paperback by MIT Press, 2003.
3. T. Gilb, *Software Metrics*, Little, Brown, and Co., 1976 (out of print).
4. R. Zultner, "The Deming Approach to Quality Software Engineering," *Quality Progress*, vol. 21, no. 11, 1988, pp. 58-64.
5. W.H. Dana, *The X-15 Lessons Learned*, tech. report, NASA Dryden Research Facility, 1993.
6. B. Randell and F.W. Zurcher, "Iterative Multi-Level Modeling: A Methodology for Computer System Design," *Proc. IFIP*, IEEE CS Press, 1968, pp. 867-871.
7. M.M. Lehman, "The Programming Process," internal IBM report, 1969; reprinted in *Program Evolution—Processes of Software Change*, Academic Press, 1985.
8. R. Glass, "Elementary Level Discussion of Compiler/Interpreter Writing," *ACM Computing Surveys*, Mar. 1969, pp. 64-68.
9. W. Royce, "Managing the Development of Large Software Systems," *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.
10. H. Mills, "Debugging Techniques in Large Systems," *Software Productivity*, Dorset House, 1988.
11. D. O'Neill, "Integration Engineering Perspective," *J. Systems and Software*, no. 3, 1983, pp. 77-83.
12. R.D. Williams, "Managing the Development of Reliable Software," *Proc. Int'l Conf. Reliable Software*, ACM Press, 1975, pp. 3-8.
13. H. Mills, "Principles of Software Engineering," *IBM Systems J.*, vol. 19, no. 4, 1980, pp. 289-295.
14. V. Basili and J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Trans. Software Eng.*, Dec. 1975, pp. 390-396.
15. H. Mills, "Software Development," *IEEE Trans. Software Eng.*, Dec. 1976, pp. 265-273.
16. D. O'Neill, "The Management of Software Engineering," *IBM Systems J.*, vol. 19, no. 4, 1980, pp. 421-431.
17. W. Madden and K. Rone, "Design, Development, Integration: Space Shuttle Flight Software System," *Comm. ACM*, Sept. 1984, pp. 914-925.
18. C. Wong, "A Successful Software Development," *IEEE Trans. Software Eng.*, no. 3, 1984, pp. 714-727.
19. T. Gilb, "Evolutionary Development," *ACM Software Eng. Notes*, Apr. 1981, p. 17.
20. W.W. Cotterman et al., eds., *Systems Analysis and*

Design: A Foundation for the 1980's, North-Holland, 1981.
21. D. McCracken and M. Jackson, "Life-Cycle Concept Considered Harmful," *ACM Software Eng. Notes*, Apr. 1982, pp. 29-32.
22. E. Dijkstra, "Go To Statement Considered Harmful," *Comm. ACM*, Mar. 1968, pp. 147-148.
23. W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation," *Comm. ACM,* July 1982, pp. 438-440.
24. D. Tamanaha, "An Integrated Rapid Prototyping Methodology for Command and Control Systems: Experience and Insight," *ACM Software Eng. Notes*, Dec. 1982, pp. 387-396.
25. G. Booch, *Software Engineering with Ada*, Benjamin-Cummings, 1983.
26. R. Budde et al., eds., *Approaches to Prototyping*, Springer Verlag, 1984.
27. T. Gilb, "Evolutionary Delivery versus the 'Waterfall Model'," *ACM Software Requirements Eng. Notes*, July 1985.
28. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Proc. Int'l Workshop Software Process and Software Environments*, ACM Press, 1985; also in *ACM Software Eng. Notes*, Aug. 1986, pp. 22-42.
29. F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Proc. IFIP,* IEEE CS Press, 1987, pp. 1069-1076; reprinted in *Computer*, Apr. 1987, pp. 10-19.
30. D. Parnas and P. Clements, "A Rational Design Process: How and Why to Fake It," *IEEE Trans. Software Eng.*, Feb. 1986, pp. 251-257.
31. W. Royce, *Software Project Management*, Addison-Wesley, 1998.
32. W. Curtis et al., "On Building Software Process Models under the Lamppost," *Proc. Int'l Conf. Software Eng.,* IEEE CS Press, 1987, pp. 96-103.
33. H. Mills et al., "Cleanroom Software Engineering," *IEEE Software*, Sept. 1987, pp. 19-25.
34. S. Jarzombek, *Proc. Joint Aerospace Weapons Systems Support, Sensors and Simulation Symp.*, Gov't Printing Office Press, 1999.
35. T. Gilb, *Principles of Software Engineering Management*, Addison Wesley Longman, 1989.
36. G.A. Newberry, "Changes from DOD-STD-2167A to MIL-STD-498," *Crosstalk: J. Defense Software Eng.*, Apr. 1995, www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1995/04/Changes.asp.
37. H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Rev.*, Jan. 1986, pp. 137-146.
38. M. Beedle et al., "SCRUM: An Extension Pattern Language for Hyperproductive Software Development," *Pattern Languages of Program Design*, vol. 4, 1999, pp. 637-651.
39. J. Stapleton, *DSDM: Dynamic Systems Development Method*, Addison-Wesley, 1997.
40. P. Kruchten, "Rational Development Process," *Crosstalk: J. Defense Software Eng.*, July 1996, www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/07/rational.asp.
41. J. McCarthy, *Dynamics of Software Development*, Microsoft Press, 1995.
42. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
43. P. Coad et al., "Feature-Driven Development," in *Java Modeling in Color with UML,* Prentice Hall, 1999.
44. A. MacCormack, "Product-Development Practices That Work," *MIT Sloan Management Rev.,* vol. 42, no. 2, 2001, pp. 75-84.
45. A. Cockburn, *Agile Software Development*, Addison-Wesley, 2002.

Craig Larman *is chief scientist for Valtech, an international consulting company, and he speaks and consults worldwide. He is also the author of* Agile and Iterative Development: A Manager's Guide *(Addison-Wesley, 2003), which examines both historical and other forms of evidence demonstrating the advantages of iterative methods. Larman is a member of the ACM and the IEEE. Contact him at craig@craiglarman.com.*

Victor R. Basili *is a professor of computer science at the University of Maryland and executive director of the Fraunhofer Center-Maryland, where he works on measuring, evaluating, and improving the software development process and product. He is an IEEE and ACM fellow and co-editor-in-chief of Kluwer's* Empirical Software Engineering: An International Journal. *Contact him at basili@cs.umd.edu.*