



# Agile Programming

**Software Development on the Fast Track**



**Technology Paper**



INTERNATIONAL  
DECISION SYSTEMS®



## **“Industrial” Programming**

Traditional software development is like building a car. You begin by envisioning the finished product down to the smallest detail. Based on that vision, you document every step of the manufacturing (or development) process. Then, and only then, do you begin stamping sheet metal or writing code.

For automobiles, the actual manufacturing process is so regimented that it can be done largely by robots. This approach works for cars because the automotive world simply doesn't change that much. Cars drive on roads, roll on rubber, burn gas, and carry passengers, just as they have since their creation.

There was a time when, for similar reasons, the same industrial approach worked for software development; strange as it may seem today, the IT world used to change relatively slowly. A programming team's customers could define their needs in detail in the expectation that those needs would be unchanged a year or more later. Software could be thoroughly documented, after which the customer signed off and went away and the programmers went to work. At the end of the process, customers got more or less what they had asked for.

If the software didn't work to spec, the developers were responsible for fixing it. If the customers had improperly or incompletely defined their needs, they either paid to have the code rewritten (at enormous cost) or lived with what they had ordered. But if the finished product generally resembled the original vision, the project was considered a success.

### **Then Everything Changed**

That world is, for all practical purposes, gone. Today's software exists in a more demanding and fluid environment. It's the rare piece of software that operates in isolation. More likely, it interfaces with other systems enterprise-wide and, via the Internet, with the world at large. It runs on a variety of platforms and performs a far wider range of functions. And those are just a few of the technical issues.

The real challenge to software today is the changing business environment. In competitive global markets, you have to meet customer expectations as never before – not just those of the mythical average customer, but of every individual customer. If you don't, there are competitors who will.

But keeping up with competition is just the beginning. The real challenge is differentiating yourself by providing better service, increasing your efficiency, and being more responsive to the market. And building software to specifications that were carved in stone a year or more ago is no way to lead.



## Out with the Old

Intense competition and a range of other pressures are forcing equipment finance companies to become much more focused on maximizing the value of each asset, moving away from the historical contract/agreement-focused model. It follows that new technology solutions must be fundamentally asset-centric, with the built-in flexibility to also deliver the information readily derived from *InfoLease*, with its focus on contracts. In a single solution, this makes it much easier for users to view key information from multiple perspectives.

Simply put, traditional software development has problems. These include;

- the assumption that a client can fully define the functionality of a piece of software at the very beginning of the development process
- the expectation that nothing in the external world will change during the development process
- the emphasis on top-down management of development
- the rigidity and overhead cost of up-front documentation and planning
- the isolation, even alienation, of individuals within a development team the relegation of testing to the very end of the process

A major argument for the traditional method has been the entire history of industrial design and engineering, where standard practice was to specify first and then build to the specification. Expected to provide on-target software, development managers and programmers felt fully justified in requiring that the target stay in one place. In the meantime, however, change in business, technology, and the world kept accelerating. This put reality and half a century of software design on a direct collision course.

## A New Philosophy

Not everyone was standing by waiting for the crash. Under a variety of names – Extreme Programming, Pragmatic Programming, Adaptive Software, Feature-Driven Development, Crystal, SCRUM, and DSDM – forward-thinking developers, as far back as the early 1990s, had been replacing the traditional approach with new, more flexible methodologies. In February of 2001, leaders of the movement met and gave the new process a name: Agile Software Development. And, despite differences in their methodologies, they issued a manifesto outlining their points of agreement. At its heart was a set of values, which prioritized:

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

Instead of relying on rigid process, it emphasized human ingenuity. It shortened the up-front planning process and replaced the overhead cost of documentation with early production of testable working code. Rather than exclude customers from the development process, it made them an integral part. And it recognized that the world does not stop when programming starts.

To return for a moment to the analogy of the automobile, Agile is, in significant ways, more like driving a car than building one. First of all, robots can't drive. While a talented machine might be able to traverse a room, driving across country or even across town requires a finer granularity of human control. Where manufacturing begins with reams of documentation, a trip typically starts with just a map. Stops, starts, acceleration, and detours are determined in real time as the journey progresses. And it is not unusual for an unplanned side trip to become an important part of the drive.





## The Flavors of Agile

While all Agile programming differs markedly from the traditional approach, its various manifestations approach the goal in individual ways.

### Advanced Software Development (ASD)

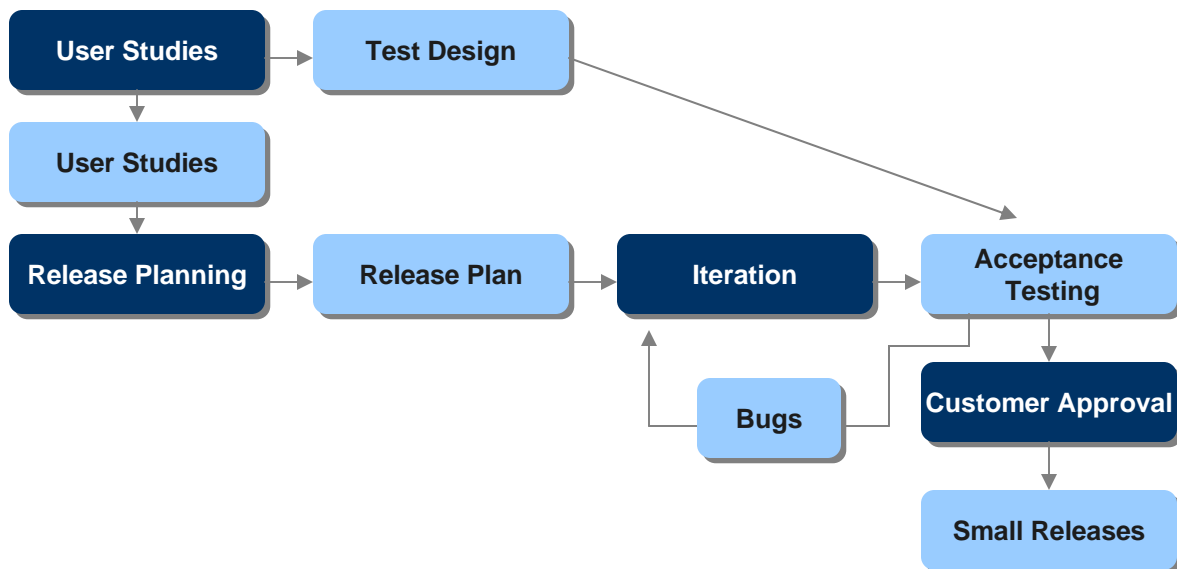
ASD isn't actually a methodology; rather, it is a philosophy. Based in chaos theory, it suggests that the number of variables impacting the environment in which software functions – developers, customers, vendors, competitors, and more – make prediction, and hence planning, virtually impossible. In its place, ASD advocates an ongoing process of estimation, feedback, and course correction in order to reach the goal. In place of the traditional process – plan, build, revise – ASD defines what it calls an adaptive life cycle – speculate, collaborate, learn, and then repeat. This concept can be found in most implementations of Agile methodology.

### Crystal

Crystal is a family of methodologies tailored to different levels of project criticality. Its users match the level of process rigor to the risk in the event of system failure. At the low end of the risk scale is loss of comfort, calling for the least rigorous Crystal methodology. At the high end the risk is loss of life, which calls for the most rigorous methodology. What all levels have in common is recognition of the importance of people, teamwork, and communication. Crystal processes all include incremental delivery, focus on software rather than documents, and emphasize user involvement, regression testing, and periodic course correction.

### Extreme Programming (XP)

XP is perhaps the best known and most widely used of Agile processes. Its methodology is defined as follows:





### **Collaboration**

- ◆ The XP process is collaborative throughout, involving product managers, business analysts, developers, and QA personnel.
- ◆ The process is organized so that business participants make the business decisions and technical participants make the technical decisions.
- ◆ A knowledgeable customer representative must be readily available for consultation throughout the entire development process.
- ◆ Short daily meetings keep all participants up to date.
- ◆ Other meetings are held on an as-needed basis and involve only the necessary participants.

### **Stories**

The development process begins with the gathering of a set of desired requirements. These are user-oriented, written in business terms and language, and are referred to as “stories.”

- ◆ Each story is typically just a few sentences in length, small enough to fit on a standard index card. Its purpose is to provide enough detail for a reasonable estimate of necessary development time.
- ◆ A story that is deemed overly large is typically subdivided into smaller segments that can be developed separately.

### **Releases**

- ◆ The development process is divided into a series of small releases. This ensures the early and steady generation of usable software.
- ◆ A release plan prioritizes stories and assigns them to planned releases along with release dates.

### **Iterations (the deliver of quality, production-ready code)**

- ◆ Development is divided into a series of cycles referred to as “iterations.” Iterations are of fixed length, typically two-weeks each. The fixed duration of iterations helps enforce the discipline of continuous production.
- ◆ During each iteration, those stories deemed to have highest priority to the customer are coded.
- ◆ Stories are drawn from the pool defined in the release plan and may include stories that were not satisfactorily completed in preceding iterations.
- ◆ Iterations overlap. Each takes place during the testing phase for the iteration that preceded it and the planning phase for the one to follow.





### **Iteration Planning**

- ◆ Each iteration is preceded by a short planning phase. The iteration plan defines the stories to be developed during that iteration. The customer makes the selection based on business priorities.
- ◆ During this phase acceptance criteria are defined for each requirement. This becomes the automated test a piece of code must pass to be considered successfully implemented and become part of the release.

### **Coding**

- ◆ In consultation with the customer, the story is expanded to understand the required functionality.
- ◆ The team then defines engineering tasks that must be completed to provide the required functionality.
- ◆ Some coding is done by pairs of programmers sharing a single machine. Pair programming has been shown to maintain quantity of output while actually increasing quality.
- ◆ Before the start of each iteration, coding is done for acceptance tests.
- ◆ Each iteration addresses only those stories assigned to it. Functions not assigned to that iteration are specifically excluded from consideration. This exclusion applies to defined requirements that are expected to be assigned to future iterations.

### **Testing**

- ◆ Testing is done at the end of each iteration using the tests developed before the start of that iteration.
- ◆ Requirements that pass are cleared for release.
- ◆ Requirements that do not pass are returned to the pool for reassignment to future iterations or, depending upon priority, fixed immediately.
- ◆ Integration
- ◆ Integration of code is a virtually continuous process, done as often as several times a day.
- ◆ This creates a shared repository of current code and eliminates some of the conflicts that burden traditional programming.
- ◆ The integrated pool of completed code is the basis for releases and provides milestones for the measurement of progress as the project proceeds.
- ◆ Automated testing is done with each build to ensure the quality of production-ready code.





INTERNATIONAL  
DECISION SYSTEMS®

#### **UNITED STATES**

International Decision Systems  
1500 IDS Center  
80 South 8th Street  
Minneapolis, MN 55402 USA  
Tel: +1.612.851.3200  
Fax: +1.612.851.3207  
Inquiries: [info@idsgrp.com](mailto:info@idsgrp.com)

#### **EUROPE**

IDS Limited  
Norton House  
1 Stewart Road  
Basingstoke  
Hampshire RG24 8NF  
United Kingdom  
Tel: +44.1256.302000  
Fax: +44.1256.302005  
Inquiries: [info@idsgrp.com](mailto:info@idsgrp.com)

#### **AUSTRALIA**

IDS Pty Ltd  
Suite, 17/ 83  
Clarence Street,  
Sydney, NSW 2000  
Australia  
Tel: +61.02.9247.9166  
Fax: +61.02.9247.9212  
Inquiries: [info@idsgrp.com](mailto:info@idsgrp.com)

#### **ASIA/PACIFIC**

International Decision Systems  
9 Temasek Boulevard  
#15-02A Suntec Tower Two  
Singapore 038989  
Tel: +65.6333.9866  
Fax: +65.6333.9877  
Inquiries: [info@idsgrp.com](mailto:info@idsgrp.com)

IDS Software Solutions (India) Private Ltd.  
6th Floor, Tower D, Corporate Block  
Diamond District, Airport Road  
Bangalore, 560 008 India  
Tel: +91.80.55150910  
Inquiries: [idsindia@idsgrp.com](mailto:idsindia@idsgrp.com)