Table of Contents

[Extreme programming 5](#_Toc416938102)

[History 5](#_Toc416938103)

[Origins 6](#_Toc416938104)

[Current state 7](#_Toc416938105)

[Concept 7](#_Toc416938106)

[Goals 7](#_Toc416938107)

[Activities 7](#_Toc416938108)

[Values 9](#_Toc416938109)

[Rules 10](#_Toc416938110)

[Principles 11](#_Toc416938111)

[Practices 12](#_Toc416938112)

[Fine-scale feedback 12](#_Toc416938113)

[Continuous process 12](#_Toc416938114)

[Shared understanding 12](#_Toc416938115)

[Programmer welfare 12](#_Toc416938116)

[Coding 12](#_Toc416938117)

[Testing 12](#_Toc416938118)

[Controversial aspects 13](#_Toc416938119)

[Scalability 13](#_Toc416938120)

[Severability and responses 14](#_Toc416938121)

[Criticism 14](#_Toc416938122)

[Agile software development 15](#_Toc416938123)

[History 15](#_Toc416938124)

[Predecessors 15](#_Toc416938125)

[The Agile Manifesto 16](#_Toc416938126)

[Evolutions 17](#_Toc416938127)

[Overview 18](#_Toc416938128)

[Iterative, incremental and evolutionary 18](#_Toc416938129)

[Efficient and face-to-face communication 18](#_Toc416938130)

[Very short feedback loop and adaptation cycle 18](#_Toc416938131)

[Quality focus 18](#_Toc416938132)

[Philosophy 19](#_Toc416938133)

[Adaptive vs. predictive 19](#_Toc416938134)

[Iterative vs. waterfall 20](#_Toc416938135)

[Code vs. documentation 20](#_Toc416938136)

[Agile methods 21](#_Toc416938137)

[Agile practices 22](#_Toc416938138)

[Method tailoring 22](#_Toc416938139)

[Comparison with other methods 23](#_Toc416938140)

[Large-scale and distributed agile development 23](#_Toc416938141)

[Experience and adoption 24](#_Toc416938142)

[Measuring agility 24](#_Toc416938143)

[While agility can be seen as a means to an end, a number of approaches have been proposed to quantify agility. *Agility Index Measurements* (AIM) 43  score projects against a number of agility factors to achieve a total. The similarly named *Agility Measurement Index*, 44  scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals. 45  Another study using fuzzy mathematics 46  has suggested that project velocity can be used as a metric of agility. There are agile self-assessments to determine whether a team is using agile practices (Nokia test, 47  Karlskrona test, 48  42 points test 49 ). 24](#_Toc416938144)

[Surveys 24](#_Toc416938145)

[Common agile pitfalls 24](#_Toc416938146)

[Criticism 28](#_Toc416938147)

[Application outside software development 28](#_Toc416938148)

[Scrum (software development) 29](#_Toc416938149)

[History 29](#_Toc416938150)

[Roles 30](#_Toc416938151)

[Product owner 30](#_Toc416938152)

[Development team 31](#_Toc416938153)

[Scrum master 31](#_Toc416938154)

[Events 32](#_Toc416938155)

[Sprint 32](#_Toc416938156)

[Events 32](#_Toc416938157)

[Extensions 34](#_Toc416938158)

[Artifacts 35](#_Toc416938159)

[Product backlog 35](#_Toc416938160)

[Sprint backlog 37](#_Toc416938161)

[Product increment 37](#_Toc416938162)

[Sprint burn-down chart 37](#_Toc416938163)

[Release burn-down chart 38](#_Toc416938164)

[Scrum-ban 38](#_Toc416938165)

[Tooling 39](#_Toc416938166)

[Limitations 39](#_Toc416938167)

[PRINCE2 40](#_Toc416938168)

[History 40](#_Toc416938169)

[Overview of the method 40](#_Toc416938170)

[Starting up a project (SU) 41](#_Toc416938171)

[Initiating a project (IP) 41](#_Toc416938172)

[Directing a project (DP) 41](#_Toc416938173)

[Controlling a stage (CS) 41](#_Toc416938174)

[Managing stage boundaries (SB) 42](#_Toc416938175)

[Managing product delivery (MP) 42](#_Toc416938176)

[Closing a project (CP) 43](#_Toc416938177)

[Management products 43](#_Toc416938178)

[Project brief 43](#_Toc416938179)

[Business case 43](#_Toc416938180)

[Risk register 43](#_Toc416938181)

[Quality register 43](#_Toc416938182)

[Issues register 43](#_Toc416938183)

[Lessons log 43](#_Toc416938184)

[Daily log 44](#_Toc416938185)

[Core principles 44](#_Toc416938186)

[Continued business justification 44](#_Toc416938187)

[Manage by exception 44](#_Toc416938188)

[Learn from experience 44](#_Toc416938189)

[Defined roles and responsibilities 44](#_Toc416938190)

[Manage by stages 44](#_Toc416938191)

[Focus on products 44](#_Toc416938192)

[Tailoring 45](#_Toc416938193)

[Integration with other techniques 45](#_Toc416938194)

[Quality review technique 45](#_Toc416938195)

[Certifications 46](#_Toc416938196)

[PRINCE2 Foundation 46](#_Toc416938197)

[PRINCE2 Practitioner 46](#_Toc416938198)

[PRINCE2 Professional 46](#_Toc416938199)

[Examinations and training 46](#_Toc416938200)

[Register 46](#_Toc416938201)

[Trainers 47](#_Toc416938202)

[Exemptions 47](#_Toc416938203)

[Scalability 47](#_Toc416938204)

[Adoption 47](#_Toc416938205)

[Advantages and pitfalls 47](#_Toc416938206)

[Rational Unified Process 48](#_Toc416938207)

[History 48](#_Toc416938208)

[Rational unified process topics 50](#_Toc416938209)

[RUP building blocks 50](#_Toc416938210)

[Four project life-cycle phases 50](#_Toc416938211)

[The IBM Rational Method Composer product 52](#_Toc416938212)

[Certification 52](#_Toc416938213)

[Six best practices 53](#_Toc416938214)

Extreme programming

**Extreme programming** (**XP**) is a [software development methodology](http://en.wikipedia.org/wiki/Software_development_methodology) which is intended to improve software quality and responsiveness to changing customer requirements. As a type of [agile software development](http://en.wikipedia.org/wiki/Agile_software_development), it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Other elements of extreme programming include: programming [in pairs](http://en.wikipedia.org/wiki/Pair_programming) or doing extensive [code review](http://en.wikipedia.org/wiki/Code_review), [unit testing](http://en.wikipedia.org/wiki/Unit_testing) of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers. [3](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-USFCA601-3)  [4](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-MASD-4) The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example, [code reviews](http://en.wikipedia.org/wiki/Code_review) are considered a beneficial practice; taken to the extreme, code can be reviewed *continuously*, i.e. the practice of [pair programming](http://en.wikipedia.org/wiki/Pair_programming).

Critics have noted several potential drawbacks, [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  including problems with unstable requirements, no documented compromises of user conflicts, and a lack of an overall design specification or document.

## **History**

Extreme Programming was created by [Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck) during his work on the [Chrysler Comprehensive Compensation System](http://en.wikipedia.org/wiki/Chrysler_Comprehensive_Compensation_System) (C3) payroll project. [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  Beck became the C3 [project leader](http://en.wikipedia.org/wiki/Project_management) in March 1996 and began to refine the development methodology used in the project and wrote a book on the methodology (in October 1999, *Extreme Programming Explained* was published). [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  Chrysler cancelled the C3 project in February 2000, after seven years, when the company was acquired by Daimler-Benz. [6](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-SR-6)

Although extreme programming itself is relatively new, many of its practices have been around for some time; the methodology, after all, takes "[best practices](http://en.wikipedia.org/wiki/Best_practices)" to extreme levels. For example, the "practice of test-first development, planning and writing tests before each micro-increment" was used as early as NASA's [Project Mercury](http://en.wikipedia.org/wiki/Project_Mercury), in the early 1960s ([Larman 2003](http://en.wikipedia.org/wiki/Extreme_programming" \l "CITEREFLarman2003)). To shorten the total development time, some formal test documents (such as for [acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing)) have been developed in parallel (or shortly before) the software is ready for testing. A NASA independent test group can write the test procedures, based on formal requirements and logical limits, before the software has been written and integrated with the hardware. In XP, this concept is taken to the extreme level by writing automated tests (perhaps inside of software modules) which validate the operation of even small sections of software coding, rather than only testing the larger features.

### Origins

Software development in the 1990s was shaped by two major influences: internally, [object-oriented programming](http://en.wikipedia.org/wiki/Object-oriented_programming) replaced [procedural programming](http://en.wikipedia.org/wiki/Procedural_programming) as the programming paradigm favored by some in the industry; externally, the rise of the Internet and the [dot-com boom](http://en.wikipedia.org/wiki/Dot-com_boom) emphasized speed-to-market and company growth as competitive business factors. Rapidly changing requirements demanded shorter [product life-cycles](http://en.wikipedia.org/wiki/Product_life_cycle_management), and were often incompatible with traditional methods of software development.

The [Chrysler Comprehensive Compensation System](http://en.wikipedia.org/wiki/Chrysler_Comprehensive_Compensation_System) (C3) was started in order to determine the best way to use object technologies, using the payroll systems at Chrysler as the object of research, with [Smalltalk](http://en.wikipedia.org/wiki/Smalltalk) as the language and [GemStone](http://en.wikipedia.org/wiki/Gemstone_Database_Management_System" \o "Gemstone Database Management System) as the [data access layer](http://en.wikipedia.org/wiki/Data_access_layer). They brought in [Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck), [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  a prominent Smalltalk practitioner, to do [performance tuning](http://en.wikipedia.org/wiki/Performance_tuning) on the system, but his role expanded as he noted several problems they were having with their development process. He took this opportunity to propose and implement some changes in their practices based on his work with his frequent collaborator, [Ward Cunningham](http://en.wikipedia.org/wiki/Ward_Cunningham). Beck describes the early conception of the methods: [7](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-7)

The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible, like testing and reviews. The second time there was a lot more on the line. I thought, "Damn the torpedoes, at least this will make a good article," and asked the team to crank up all the knobs to 10 on the things I thought were essential and leave out everything else.

Beck invited [Ron Jeffries](http://en.wikipedia.org/wiki/Ron_Jeffries) to the project to help develop and refine these methods. Jeffries thereafter acted as a coach to instill the practices as habits in the C3 team.

Information about the principles and practices behind XP was disseminated to the wider world through discussions on the original [wiki](http://en.wikipedia.org/wiki/Wiki), Cunningham's [WikiWikiWeb](http://en.wikipedia.org/wiki/WikiWikiWeb). Various contributors discussed and expanded upon the ideas, and some spin-off methodologies resulted (see [agile software development](http://en.wikipedia.org/wiki/Agile_software_development)). Also, XP concepts have been explained, for several years, using a[hypertext](http://en.wikipedia.org/wiki/Hypertext) system map on the XP website at "[http://www.extremeprogramming.org](http://www.extremeprogramming.org/)" circa 1999.

Beck edited a series of books on XP, beginning with his own *Extreme Programming Explained* (1999, [ISBN 0-201-61641-6](http://en.wikipedia.org/wiki/Special:BookSources/0201616416)), spreading his ideas to a much larger audience. Authors in the series went through various aspects attending XP and its practices. The series included a book that was critical of the practices.

### Current state

XP generated significant interest among software communities in the late 1990s and early 2000s, seeing adoption in a number of environments radically different from its origins.

The high discipline required by the original practices often went by the wayside, causing some of these practices, such as those thought too rigid, to be deprecated or reduced, or even left unfinished, on individual sites. For example, the practice of end-of-day [integration tests](http://en.wikipedia.org/wiki/Integration_test) for a particular project could be changed to an end-of-week schedule, or simply reduced to mutually agreed dates. Such a more relaxed schedule could avoid people feeling rushed to generate artificial stubs just to pass the end-of-day testing. A less-rigid schedule allows, instead, for some complex features to be more fully developed over a several-day period. However, some level of periodic integration testing can detect groups of people working in non-compatible, tangent efforts before too much work is invested in divergent, wrong directions.

Meanwhile, other agile development practices have not stood still, and XP is still evolving, assimilating more lessons from experiences in the field, to use other practices. In the second edition of*Extreme Programming Explained* (November 2004), five years after the first edition, Beck added more values and practices and differentiated between primary and corollary practices.

## **Concept**

### Goals

*Extreme Programming Explained* describes extreme programming as a software-development discipline that organizes people to produce higher-quality software more productively.

XP attempts to reduce the cost of changes in requirements by having multiple short development cycles, rather than a long one. In this doctrine, changes are a natural, inescapable and desirable aspect of software-development projects, and should be planned for, instead of attempting to define a stable set of requirements.

Extreme programming also introduces a number of basic values, principles and practices on top of the agile programming framework.

### Activities

XP describes four basic activities that are performed within the software development process: coding, testing, listening, and designing. Each of those activities is described below.

#### Coding

The advocates of XP argue that the only truly important product of the system development process is code – software instructions that a computer can interpret. Without code, there is no working product.

Coding can also be used to figure out the most suitable solution. Coding can also help to communicate thoughts about programming problems. A programmer dealing with a complex programming problem, or finding it hard to explain the solution to fellow programmers, might code it in a simplified manner and use the code to demonstrate what he or she means. Code, say the proponents of this position, is always clear and concise and cannot be interpreted in more than one way. Other programmers can give feedback on this code by also coding their thoughts.

#### Testing

*Main article:*[*Test-driven development*](http://en.wikipedia.org/wiki/Test-driven_development)

Extreme programming's approach is that if a little testing can eliminate a few flaws, a lot of testing can eliminate many more flaws.

* [Unit tests](http://en.wikipedia.org/wiki/Unit_test) determine whether a given feature works as intended. A programmer writes as many automated tests as they can think of that might "break" the code; if all tests run successfully, then the coding is complete. Every piece of code that is written is tested before moving on to the next feature.
* [Acceptance tests](http://en.wikipedia.org/wiki/Acceptance_test) verify that the requirements as understood by the programmers satisfy the customer's actual requirements.

System-wide [integration testing](http://en.wikipedia.org/wiki/Integration_testing) was encouraged, initially, as a daily end-of-day activity, for early detection of incompatible interfaces, to reconnect before the separate sections diverged widely from coherent functionality. However, system-wide integration testing has been reduced, to weekly, or less often, depending on the stability of the overall interfaces in the system.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

#### Listening

Programmers must listen to what the customers need the system to do, what ["business logic"](http://en.wikipedia.org/wiki/Business_logic) is needed. They must understand these needs well enough to give the customer feedback about the technical aspects of how the problem might be solved, or cannot be solved. Communication between the customer and programmer is further addressed in the [*planning game*](http://en.wikipedia.org/wiki/Planning_game).

#### Designing

From the point of view of simplicity, of course one could say that system development doesn't need more than coding, testing and listening. If those activities are performed well, the result should always be a system that works. In practice, this will not work. One can come a long way without designing but at a given time one will get stuck. The system becomes too complex and the dependencies within the system cease to be clear. One can avoid this by creating a design structure that organizes the logic in the system. Good design will avoid lots of dependencies within a system; this means that changing one part of the system will not affect other parts of the system.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

### Values

Extreme programming initially recognized four values in 1999: communication, simplicity, feedback, and courage. A new value, respect, was added in the second edition of *Extreme Programming Explained*. Those five values are described below.

#### Communication

Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentation. Extreme programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team. The goal is to give all developers a shared view of the system which matches the view held by the users of the system. To this end, extreme programming favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

#### Simplicity

Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later. The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month. This is sometimes summed up as the "[You aren't gonna need it](http://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it)" (YAGNI) approach. [8](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-tr-8)  Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant. Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features. Related to the "communication" value, simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

#### Feedback

Within extreme programming, feedback relates to different dimensions of the system development:

* Feedback from the system: by writing [unit tests](http://en.wikipedia.org/wiki/Unit_test), [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
* Feedback from the customer: The functional tests (aka [acceptance tests](http://en.wikipedia.org/wiki/Acceptance_tests)) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
* Feedback from the team: When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.

Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break. The direct feedback from the system tells programmers to recode this part. A customer is able to test the system periodically according to the functional requirements, known as [*user stories*](http://en.wikipedia.org/wiki/User_story). [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  To quote [Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck), "Optimism is an occupational hazard of programming. Feedback is the treatment." [9](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Beck99-9)

#### Courage

Several practices embody courage. One is the commandment to always design and code for today and not for tomorrow. This is an effort to avoid getting bogged down in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with [refactoring](http://en.wikipedia.org/wiki/Refactoring) their code when necessary. [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  This means reviewing the existing system and modifying it so that future changes can be implemented more easily. Another example of courage is knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code. Also, courage means persistence: A programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, but only if they are persistent.

#### Respect

The respect value includes respect for others as well as self-respect. Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers. Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.

Adopting the four earlier values leads to respect gained from others in the team. Nobody on the team should feel unappreciated or ignored. This ensures a high level of motivation and encourages loyalty toward the team and toward the goal of the project. This value is very dependent upon the other values, and is very much oriented toward people in a team.

### Rules

The first version of rules for XP was published in 1999 by Don Wells [10](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-10)  at the XP website. 29 rules are given in the categories of planning, managing, designing, coding, and testing. Planning, managing and designing are called out explicitly to counter claims that XP doesn't support those activities.

Another version of XP rules was proposed by Ken Auer [11](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-11)  in XP/Agile Universe 2003. He felt XP was defined by its rules, not its practices (which are subject to more variation and ambiguity). He defined two categories: "Rules of Engagement" which dictate the environment in which software development can take place effectively, and "Rules of Play" which define the minute-by-minute activities and rules within the framework of the Rules of Engagement.

### Principles

The principles that form the basis of XP are based on the values just described and are intended to foster decisions in a system development project. The principles are intended to be more concrete than the values and more easily translated to guidance in a practical situation.

#### Feedback

Extreme programming sees feedback as most useful if it is done frequently and promptly. It stresses that minimal delay between an action and its feedback is critical to learning and making changes. Unlike traditional system development methods, contact with the customer occurs in more frequent iterations. The customer has clear insight into the system that is being developed, and can give feedback and steer the development as needed. With frequent feedback from the customer, a mistaken design decision made by the developer will be noticed and corrected quickly, before the developer spends much time implementing it.

Unit tests contribute to the rapid feedback principle. When writing code, running the unit test provides direct feedback as to how the system reacts to the changes made. This includes running not only the unit tests that test the developer's code, but running in addition all unit tests against all the software, using an automated process that can be initiated by a single command. That way, if the developer's changes cause a failure in some other portion of the system that the developer knows little or nothing about, the automated all-unit-test suite will reveal the failure immediately, alerting the developer of the incompatibility of his change with other parts of the system, and the necessity of removing or modifying his change. Under traditional development practices, the absence of an automated, comprehensive unit-test suite meant that such a code change, assumed harmless by the developer, would have been left in place, appearing only during integration testing – or worse, only in production; and determining which code change caused the problem, among all the changes made by all the developers during the weeks or even months previous to integration testing, was a formidable task.

#### Assuming simplicity

This is about treating every problem as if its solution were "extremely simple". Traditional system development methods say to plan for the future and to code for reusability. Extreme programming rejects these ideas.

The advocates of extreme programming say that making big changes all at once does not work. Extreme programming applies incremental changes: for example, a system might have small releases every three weeks. When many little steps are made, the customer has more control over the development process and the system that is being developed.

#### Embracing change

The principle of embracing change is about not working against changes but embracing them. For instance, if at one of the iterative meetings it appears that the customer's requirements have changed dramatically, programmers are to embrace this and plan the new requirements for the next iteration.

## **Practices**

*For more details on this topic, see*[*Extreme programming practices*](http://en.wikipedia.org/wiki/Extreme_programming_practices)*.*

Extreme programming has been described as having 12 practices, grouped into four areas:

### Fine-scale feedback

* [Pair programming](http://en.wikipedia.org/wiki/Pair_programming) [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)
* [Planning game](http://en.wikipedia.org/wiki/Planning_game)
* [Test-driven development](http://en.wikipedia.org/wiki/Test-driven_development)
* [Whole team](http://en.wikipedia.org/wiki/Extreme_programming_practices#Whole_team)

### Continuous process

* [Continuous integration](http://en.wikipedia.org/wiki/Continuous_integration)
* [Refactoring](http://en.wikipedia.org/wiki/Refactoring) or design improvement [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)
* [Small releases](http://en.wikipedia.org/wiki/Extreme_programming_practices#Small_releases)

### Shared understanding

* [Coding standards](http://en.wikipedia.org/wiki/Extreme_programming_practices#Coding_standard)
* [Collective code ownership](http://en.wikipedia.org/wiki/Extreme_programming_practices#Collective_code_ownership) [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)
* [Simple design](http://en.wikipedia.org/wiki/Extreme_programming_practices#Simple_design) [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)
* [System metaphor](http://en.wikipedia.org/wiki/Extreme_programming_practices#System_metaphor)

### Programmer welfare

* [Sustainable pace](http://en.wikipedia.org/wiki/Extreme_programming_practices#Sustainable_pace)

### Coding

* The customer is always available
* Code the [unit test](http://en.wikipedia.org/wiki/Unit_test) first
* Only one pair integrates code at a time
* Leave [optimization](http://en.wikipedia.org/wiki/Program_optimization) until last
* No [overtime](http://en.wikipedia.org/wiki/Overtime)

### Testing

* All code must have [unit tests](http://en.wikipedia.org/wiki/Unit_tests)
* All code must pass all [unit tests](http://en.wikipedia.org/wiki/Unit_tests) before it can be released.
* When a [bug](http://en.wikipedia.org/wiki/Software_bug) is found tests are created before the bug is addressed (a bug is not an error in logic, it is a test that was not written)
* [Acceptance tests](http://en.wikipedia.org/wiki/Acceptance_tests) are run often and the results are published

## **Controversial aspects**

The practices in XP have been heavily debated. [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  Proponents of extreme programming claim that by having the on-site customer [5](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-Cworld92-5)  request changes informally, the process becomes flexible, and saves the cost of formal overhead. Critics of XP claim this can lead to costly rework and project [scope creep](http://en.wikipedia.org/wiki/Scope_creep) beyond what was previously agreed or funded.

Change-control boards are a sign that there are potential conflicts in project objectives and constraints between multiple users. XP's expedited methods are somewhat dependent on programmers being able to assume a unified client viewpoint so the programmer can concentrate on coding, rather than documentation of compromise objectives and constraints. This also applies when multiple programming organizations are involved, particularly organizations which compete for shares of projects.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

Other potentially controversial aspects of extreme programming include:

* Requirements are expressed as automated acceptance tests rather than specification documents.
* Requirements are defined incrementally, rather than trying to get them all in advance.
* Software developers are usually required to work in pairs.
* There is no [Big Design Up Front](http://en.wikipedia.org/wiki/Big_Design_Up_Front). Most of the design activity takes place on the fly and incrementally, starting with "the simplest thing that could possibly work" and adding complexity only when it's required by failing tests. Critics compare this to "[debugging](http://en.wikipedia.org/wiki/Debugging) a system into appearance" and fear this will result in more re-design effort than only re-designing when requirements change.
* A [customer representative](http://en.wikipedia.org/wiki/Customer_representative) is attached to the project. This role can become a single-point-of-failure for the project, and some people have found it to be a source of stress. Also, there is the danger of [micro-management](http://en.wikipedia.org/wiki/Micro-management) by a non-technical representative trying to dictate the use of technical software features and architecture.
* Dependence upon all other aspects of XP: "XP is like a ring of poisonous snakes, daisy-chained together. All it takes is for one of them to wriggle loose, and you've got a very angry, poisonous snake heading your way." [12](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-12)

### Scalability

Historically, XP only works on teams of twelve or fewer people. One way to circumvent this limitation is to break up the project into smaller pieces and the team into smaller groups. It has been claimed that XP has been used successfully on teams of over a hundred developers.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed) [ThoughtWorks](http://en.wikipedia.org/wiki/ThoughtWorks" \o "ThoughtWorks) has claimed reasonable success on distributed XP projects with up to sixty people.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

In 2004, industrial extreme programming (IXP) [13](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-13)  was introduced as an evolution of XP. It is intended to bring the ability to work in large and distributed teams. It now has 23 practices and flexible values.

### Severability and responses

In 2003, [Matt Stephens](http://en.wikipedia.org/wiki/Matt_Stephens) and Doug Rosenberg published *Extreme Programming Refactored: The Case Against XP*, which questioned the value of the XP process and suggested ways in which it could be improved. [6](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-SR-6)  This triggered a lengthy debate in articles, Internet newsgroups, and web-site chat areas. The core argument of the book is that XP's practices are interdependent but that few practical organizations are willing/able to adopt all the practices; therefore the entire process fails. The book also makes other criticisms, and it draws a likeness of XP's "collective ownership" model to socialism in a negative manner.

Certain aspects of XP have changed since the publication of *Extreme Programming Refactored*; in particular, XP now accommodates modifications to the practices as long as the required objectives are still met. XP also uses increasingly generic terms for processes. Some argue that these changes invalidate previous criticisms; others claim that this is simply watering the process down.

Other authors have tried to reconcile XP with the older methodologies in order to form a unified methodology. Some of these XP sought to replace, such as the [waterfall methodology](http://en.wikipedia.org/wiki/Waterfall_model); example:[Project Lifecycles: Waterfall, Rapid Application Development, and All That](http://www.lux-seattle.com/resources/whitepapers/waterfall.htm). [JPMorgan Chase & Co.](http://en.wikipedia.org/wiki/JPMorgan_Chase_%26_Co.) tried combining XP with the computer programming methods of [capability maturity model integration](http://en.wikipedia.org/wiki/Capability_maturity_model_integration) (CMMI), and [Six Sigma](http://en.wikipedia.org/wiki/Six_Sigma). They found that the three systems reinforced each other well, leading to better development, and did not mutually contradict. [14](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-14)

## **Criticism**

Extreme programming's initial buzz and controversial tenets, such as [pair programming](http://en.wikipedia.org/wiki/Pair_programming) and [continuous design](http://en.wikipedia.org/wiki/Continuous_design), have attracted particular criticisms, such as the ones coming from McBreen [15](http://en.wikipedia.org/wiki/Extreme_programming" \l "cite_note-mcbreen-15) and Boehm and Turner., [16](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-boehm2004-16)  Matt Stephens and Doug Rosenberg. [17](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-stephens2004-17)  Many of the criticisms, however, are believed by Agile practitioners to be misunderstandings of agile development. [18](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-sdmagazine1811-18)

In particular, extreme programming has been reviewed and critiqued by Matt Stephens's and Doug Rosenberg's *Extreme Programming Refactored*. [6](http://en.wikipedia.org/wiki/Extreme_programming" \l "cite_note-SR-6)  [19](http://en.wikipedia.org/wiki/Extreme_programming#cite_note-ExtremeProgrammingRefactored-19)

Criticisms include:

* a methodology is only as effective as the people involved, Agile does not solve this
* often used as a means to bleed money from customers through lack of defining a deliverable product
* lack of structure and necessary documentation
* only works with senior-level developers
* incorporates insufficient software design
* requires meetings at frequent intervals at enormous expense to customers
* requires too much cultural change to adopt
* can lead to more difficult contractual negotiations
* can be very inefficient; if the requirements for one area of code change through various iterations, the same programming may need to be done several times over. Whereas if a plan were there to be followed, a single area of code is expected to be written once.
* impossible to develop realistic estimates of work effort needed to provide a quote, because at the beginning of the project no one knows the entire scope/requirements
* can increase the risk of [scope creep](http://en.wikipedia.org/wiki/Scope_creep) due to the lack of detailed requirements documentation
* Agile is feature-driven; non-functional quality attributes are hard to be placed as [user stories](http://en.wikipedia.org/wiki/User_story).

# Agile software development

**Agile software development** is a group of [software development methods](http://en.wikipedia.org/wiki/Software_development_methodologies) in which requirements and solutions evolve through collaboration between self-organizing, [cross-functional teams](http://en.wikipedia.org/wiki/Cross-functional_team). It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change. [1](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-1)

The *Manifesto for Agile Software Development*, also known as the *Agile Manifesto*, first introduced the term *agile* in the context of software development in 2001.

## **History**

### Predecessors

Incremental software development methods trace back to 1957. [3](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-craig2003-3)  In 1974, E. A. Edmonds wrote a paper that introduced an adaptive software development process. [4](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-edmonds1974-4)  [5](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-5)  Concurrently and independently, the same methods were developed and deployed by the New York Telephone Company's Systems Development Center under the direction of Dan Gielan. In the early 1970s,[Tom Gilb](http://en.wikipedia.org/wiki/Tom_Gilb" \o "Tom Gilb) started publishing the concepts of evolutionary project management (EVO), which has evolved into *competitive engineering*. [6](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-EvolutionaryProjectManagement-6)  During the mid- to late 1970s, Gielan lectured extensively throughout the U.S. on this methodology, its practices, and its benefits.

A collection of *lightweight* software development methods evolved in the mid-1990s in reaction to the perceived *heavyweight* [waterfall](http://en.wikipedia.org/wiki/Waterfall_model)-oriented methods, which critics called heavily regulated, regimented, and [micro-managed](http://en.wikipedia.org/wiki/Micromanagement); although some proponents of these lightweight methods contended that they were simply returning to earlier software practices. [3](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-craig2003-3)  These lightweight methods included: from 1994, [unified process](http://en.wikipedia.org/wiki/Unified_Process) and [dynamic systems development method](http://en.wikipedia.org/wiki/Dynamic_systems_development_method) (DSDM); from 1995, [scrum](http://en.wikipedia.org/wiki/Scrum_(software_development)); from 1996, [crystal clear](http://en.wikipedia.org/wiki/Crystal_Clear_(software_development)) and [extreme programming](http://en.wikipedia.org/wiki/Extreme_programming) (aka "XP"); and from 1997,[adaptive software development](http://en.wikipedia.org/wiki/Adaptive_software_development) and [feature-driven development](http://en.wikipedia.org/wiki/Feature-driven_development). Although these originated before the publication of the Agile Manifesto in 2001, they are now collectively referred to as agile methods; [7](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-rajeshkumar-7)  and often abbreviated loosely as *Agile*, with a capital *A*, although this is progressively becoming deprecated. [8](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-8)

### The Agile Manifesto

In February 2001, 17 software developers (see below) met at the [Snowbird](http://en.wikipedia.org/wiki/Snowbird,_Utah) resort in [Utah](http://en.wikipedia.org/wiki/Utah) to discuss lightweight development methods. They published the *Manifesto for Agile Software Development*:

*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

**Individuals and interactions** over Processes and tools

**Working software** over Comprehensive documentation

**Customer collaboration** over Contract negotiation

**Responding to change** over Following a plan

*That is, while there is value in the items on the right, we value the items on the left more.*

|  |  |  |
| --- | --- | --- |
| [Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck) | James Grenning | [Robert C. Martin](http://en.wikipedia.org/wiki/Robert_Cecil_Martin) |
| Mike Beedle | [Jim Highsmith](http://en.wikipedia.org/wiki/Jim_Highsmith) | [Steve Mellor](http://en.wikipedia.org/wiki/Stephen_J._Mellor) |
| Arie van Bennekum | [Andrew Hunt](http://en.wikipedia.org/wiki/Andy_Hunt_(author)) | [Ken Schwaber](http://en.wikipedia.org/wiki/Ken_Schwaber) |
| [Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn) | [Ron Jeffries](http://en.wikipedia.org/wiki/Ron_Jeffries) | [Jeff Sutherland](http://en.wikipedia.org/wiki/Jeff_Sutherland) |
| [Ward Cunningham](http://en.wikipedia.org/wiki/Ward_Cunningham) | Jon Kern | [Dave Thomas](http://en.wikipedia.org/wiki/Dave_Thomas_(programmer)) |
| [Martin Fowler](http://en.wikipedia.org/wiki/Martin_Fowler) | [Brian Marick](http://en.wikipedia.org/wiki/Brian_Marick) |  |

© 2001, the above authors. This declaration may be freely copied in any form, but only in its entirety through this notice.

The meanings of the manifesto items on the left are:

* *Individuals and interactions*: self-organization and motivation are important, as are interactions like [co-location](http://en.wikipedia.org/wiki/Colocation_(business)) and [pair programming](http://en.wikipedia.org/wiki/Pair_programming).
* *Working software*: working software is more useful and welcome than just presenting documents to clients in meetings.
* *Customer collaboration*: requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
* *Responding to change*: agile methods are focused on quick responses to change and continuous development. [9](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-abmmw-9)

Some of the authors formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's values and principles. Introducing the manifesto on behalf of the Agile Alliance, [Jim Highsmith](http://en.wikipedia.org/wiki/Jim_Highsmith) said,

The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker.

—Jim Highsmith, History: The Agile Manifesto [10](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-10)

#### Agile principles

The Agile Manifesto is based on 12 principles: [11](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-manifestoprinciples-11)

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstance

### Evolutions

Later, [Ken Schwaber](http://en.wikipedia.org/wiki/Ken_Schwaber) with others founded the Scrum Alliance and created the Certified Scrum Master programs and its derivatives. Schwaber left the Scrum Alliance in the fall of 2009, and founded Scrum.org.

In 2005, a group headed by [Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn) and [Jim Highsmith](http://en.wikipedia.org/wiki/Jim_Highsmith) wrote an addendum of [project management](http://en.wikipedia.org/wiki/Project_management) principles, the [Declaration of Interdependence](http://en.wikipedia.org/wiki/PM_Declaration_of_Interdependence), [12](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-12)  to guide software project management according to agile software development methods.

In 2009, a movement spearheaded by [Robert C Martin](http://en.wikipedia.org/wiki/Robert_Cecil_Martin) wrote an extension of [software development](http://en.wikipedia.org/wiki/Software_development) principles, the [Software Craftsmanship Manifesto](http://en.wikipedia.org/wiki/Software_craftsmanship), to guide agile software development according to [professional](http://en.wikipedia.org/wiki/Professional) conduct and mastery.

In 2011 the original Agile Alliance [13](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-13)  created the Guide to Agile Practices, an evolving open-source compendium of the working definitions of agile practices, terms, and elements, along with interpretations and experience guidelines from the world-wide community of agile practitioners.

The [PRINCE2](http://en.wikipedia.org/wiki/PRINCE2) project management methodology, used on many British Government projects, is being enhanced to manage projects that use Agile techniques. [14](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-14)

## **Overview**

There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

### Iterative, incremental and evolutionary

Most agile development methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames ([timeboxes](http://en.wikipedia.org/wiki/Timeboxing" \o "Timeboxing)) that typically last from one to four weeks. Each iteration involves a [cross-functional team](http://en.wikipedia.org/wiki/Cross-functional_team) working in all functions: planning,[requirements analysis](http://en.wikipedia.org/wiki/Requirements_analysis), [design](http://en.wikipedia.org/wiki/Software_design), [coding](http://en.wikipedia.org/wiki/Computer_programming), [unit testing](http://en.wikipedia.org/wiki/Unit_testing), and [acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing). At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal [bugs](http://en.wikipedia.org/wiki/Software_bug)) at the end of each iteration. [15](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-embracing_change-15)  Multiple iterations might be required to release a product or new features.

### Efficient and face-to-face communication

No matter what development disciplines are required, each agile team should include a [customer representative](http://en.wikipedia.org/wiki/Customer_representative) (*product owner* in scrum). This person is appointed by stakeholders to act on their behalf [16](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-16)  and makes a personal commitment to being available for developers to answer mid-iteration questions. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the [return on investment](http://en.wikipedia.org/wiki/Rate_of_return) (ROI) and ensuring alignment with customer needs and company goals.

In agile software development, an *information radiator* is a (normally large) physical display located prominently in an office, where passers-by can see it. It presents an up-to-date summary of the status of a software project or other product. [17](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Cockburn.2C_Information_radiator-17)  [18](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Ambler-18)  The name was coined by [Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn), and described in his 2002 book *Agile Software Development*. [18](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Ambler-18)  A [build light indicator](http://en.wikipedia.org/wiki/Build_light_indicator) may be used to inform a team about the current status of their project.

### Very short feedback loop and adaptation cycle

A common characteristic in agile development is the daily "[stand-up](http://en.wikipedia.org/wiki/Stand-up_meeting)", also known as the *daily scrum*. In a brief session, team members report to each other what they did the previous day toward their team's sprint goal, what they intend to do today toward their team's sprint goal, and any roadblocks or impediments they can see to their team's sprint goal. [19](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-19)

### Quality focus

Specific tools and techniques, such as [continuous integration](http://en.wikipedia.org/wiki/Continuous_integration), automated [unit testing](http://en.wikipedia.org/wiki/Unit_testing), [pair programming](http://en.wikipedia.org/wiki/Pair_programming), [test-driven development](http://en.wikipedia.org/wiki/Test-driven_development), [design patterns](http://en.wikipedia.org/wiki/Software_design_pattern), [domain-driven design](http://en.wikipedia.org/wiki/Domain-driven_design), [code refactoring](http://en.wikipedia.org/wiki/Code_refactoring) and other techniques are often used to improve quality and enhance project agility.

## **Philosophy**

Compared to traditional software engineering, agile software development mainly targets complex systems and projects with dynamic, undeterministic and non-linear characteristics, where accurate estimates, stable plans, and predictions are often hard to get in early stages—and big up-front designs and arrangements would probably cause a lot of waste, i.e., are not economically sound. These basic arguments and previous industry experiences, learned from years of successes and failures, have helped shape agile development's favor of adaptive, iterative and evolutionary development. [20](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Larman2004-20)

### Adaptive vs. predictive

Development methods exist on a continuum from *adaptive* to *predictive*. [21](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-boehm2004App-21)  Agile methods lie on the *adaptive* side of this continuum. One key of adaptive development methods is a "[Rolling Wave](http://en.wikipedia.org/wiki/Rolling_Wave_planning)" approach to schedule planning, which identifies milestones but leaves flexibility in the path to reach them, and also allows for the milestones themselves to change. [22](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-22)  Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team has difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method is about what will happen on that date. An adaptive team cannot report exactly what tasks they will do next week, but only which features they plan for next month. When asked about a release six months from now, an adaptive team might be able to report only the mission statement for the release, or a statement of expected value vs. cost.

*Predictive* method, in contrast, focus on analysing and planning the future in detail and cater for known risks. In the extremes, a predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction. Predictive teams often institute a [change control board](http://en.wikipedia.org/wiki/Change_control_board) to ensure they consider only the most valuable changes.

[Risk analysis](http://en.wikipedia.org/wiki/Risk_analysis_(engineering)) can be used to choose between adaptive (*agile* or *value-driven*) and predictive (*plan-driven*) methods. [23](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-PMBridgeToAgility-23)  [Barry Boehm](http://en.wikipedia.org/wiki/Barry_Boehm) and [Richard Turner](http://en.wikipedia.org/wiki/Richard_Turner_(software)) suggest that each side of the continuum has its own *home ground*, as follows: [24](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-boehm2004-24)

|  |  |  |
| --- | --- | --- |
| **Home grounds of different development methods** | | |
| **Agile methods** | **Plan-driven methods** | **Formal methods** |
| Low criticality | High criticality | Extreme criticality |
| Senior developers | Junior developers(?) | Senior developers |
| Requirements change often | Requirements do not change often | Limited requirements, limited features see [Wirth's law](http://en.wikipedia.org/wiki/Wirth%27s_law) |
| Small number of developers | Large number of developers | Requirements that can be modeled |
| Culture that responds to change | Culture that demands order | Extreme quality |

### Iterative vs. waterfall

One of the differences between agile and waterfall is the approach to quality and testing. In the [waterfall model](http://en.wikipedia.org/wiki/Waterfall_model), there is always a separate *testing phase* after a *build phase*; however, in agile development testing is usually done concurrently with, or at least in same iteration as, programming.

Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value.

After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—[scrum](http://en.wikipedia.org/wiki/Scrum_(software_development)) typically has iterations of just two weeks—helps the team continuously adapt its plans so as to maximize the value it delivers.

This iterative practice also introduces a *product* mindset rather than the waterfall model's *project* mindset. Software can be seen as an living organism, which actively changes due to environmental change. As long as the software is being used, especially when it has competition, iterations in agile software development drive the change.

Because of the short iteration style of agile software development, it also has strong connections with the [lean startup](http://en.wikipedia.org/wiki/Lean_startup) concept.

### Code vs. documentation

In a letter to IEEE *Computer*, Steven Rakitin expressed cynicism about agile development, calling it "yet another attempt to undermine the discipline of software engineering" and translating "Working software over comprehensive documentation" as "We want to spend all our time coding. Remember, real programmers don't write documentation." [25](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-rakitin2001-25)

This is disputed by proponents of agile software development, who state that developers should write documentation if that's the best way to achieve the relevant goals, but that there are often better ways to achieve those goals than writing static documentation. [26](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-agiledoc-26)  [Scott Ambler](http://en.wikipedia.org/wiki/Scott_Ambler) states that documentation should be "Just Barely Good Enough" (JBGE), [27](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-27)  that too much or comprehensive documentation would usually cause waste, and developers rarely trust detailed documentation because it's usually out of sync with code, [26](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-agiledoc-26)  while too little documentation may also cause problems for maintenance, communication, learning and knowledge sharing. [Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn) wrote of the *Crystal Clear* method:

Crystal considers development a series of co-operative games, and intends that the documentation is enough to help the next win at the next game. The work products for Crystal include use cases, risk list, iteration plan, core domain models, and design notes to inform on choices...however there are no templates for these documents and descriptions are necessarily vague, but the objective is clear, **just enough documentation** for the next game. I always tend to characterize this to my team as: what would you want to know if you joined the team tomorrow. [28](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-28)

—Alistair Cockburn[*attribution verification needed*](http://en.wikipedia.org/wiki/Wikipedia:Verifiability)

## **Agile methods**

Well-known agile software development methods and/or process frameworks include:

* [Adaptive software development](http://en.wikipedia.org/wiki/Adaptive_software_development) (ASD)
* [Agile modeling](http://en.wikipedia.org/wiki/Agile_modeling)
* [Agile Unified Process](http://en.wikipedia.org/wiki/Agile_Unified_Process) (AUP)
* [Crystal Clear Methods](http://en.wikipedia.org/wiki/Crystal_Clear_(software_development))
* [Disciplined agile delivery](http://en.wikipedia.org/wiki/Disciplined_agile_delivery)
* [Dynamic systems development method](http://en.wikipedia.org/wiki/Dynamic_systems_development_method) (DSDM)
* [Extreme programming](http://en.wikipedia.org/wiki/Extreme_programming) (XP)
* [Feature-driven development](http://en.wikipedia.org/wiki/Feature-driven_development) (FDD)
* [Lean software development](http://en.wikipedia.org/wiki/Lean_software_development)
* [Kanban (development)](http://en.wikipedia.org/wiki/Kanban_(development))
* [Scrum](http://en.wikipedia.org/wiki/Scrum_(software_development))
* [Scrum ban](http://en.wikipedia.org/wiki/Scrum_ban)

Agile methods are focused on different aspects of the [software development life cycle](http://en.wikipedia.org/wiki/Software_development_life_cycle). Some focus on the practices (e.g. XP, pragmatic programming, agile modeling), while others focus on managing the software projects (e.g. scrum). Yet, there are approaches providing full coverage over the development life cycle (e.g. DSDM, [RUP](http://en.wikipedia.org/wiki/Rational_Unified_Process)), while most of them are suitable from the requirements specification phase on (FDD, for example). Thus, there is a clear difference between the various agile methods in this regard. [29](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Abrahamsson2002-29)

### Agile practices

Agile development is supported by a bundle of concrete practices, covering areas like requirements, design, modeling, coding, testing, project management, process, quality, etc. Some notable agile practices include:

* [Acceptance test-driven development](http://en.wikipedia.org/wiki/Acceptance_test-driven_development) (ATDD)
* [Agile modeling](http://en.wikipedia.org/wiki/Agile_modeling)
* [Backlogs](http://en.wikipedia.org/wiki/Scrum_(development)#Product_backlog) (Product and Sprint)
* [Behavior-driven development](http://en.wikipedia.org/wiki/Behavior-driven_development) (BDD)
* [Cross-functional team](http://en.wikipedia.org/wiki/Cross-functional_team)
* [Continuous integration](http://en.wikipedia.org/wiki/Continuous_integration) (CI)
* [Domain-driven design](http://en.wikipedia.org/wiki/Domain-driven_design) (DDD)
* Information radiators (scrum board, task board, visual management board, [burndown chart](http://en.wikipedia.org/wiki/Burndown_chart" \o "Burndown chart))
* [Iterative and incremental development](http://en.wikipedia.org/wiki/Iterative_and_incremental_development) (IID)
* [Pair programming](http://en.wikipedia.org/wiki/Pair_programming)
* [Planning poker](http://en.wikipedia.org/wiki/Planning_poker)
* [Refactoring](http://en.wikipedia.org/wiki/Refactoring)
* [Scrum events](http://en.wikipedia.org/wiki/Scrum_(development)) (sprint planning, daily scrum, sprint review and retrospective)
* [Test-driven development](http://en.wikipedia.org/wiki/Test-driven_development) (TDD)
* [Agile testing](http://en.wikipedia.org/wiki/Agile_testing)
* [Timeboxing](http://en.wikipedia.org/wiki/Timeboxing)
* [Use case](http://en.wikipedia.org/wiki/Use_case)
* [User story](http://en.wikipedia.org/wiki/User_story)
* [Story-driven modeling](http://en.wikipedia.org/wiki/Story-driven_modeling)
* [Retrospective](http://en.wikipedia.org/wiki/Retrospective)
* [Velocity tracking](http://en.wikipedia.org/wiki/Velocity_tracking)

The Agile Alliance has provided a comprehensive online collection with a map guide to the applying agile practices. [30](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Agile_Practices_Guide-30)

### Method tailoring

In the literature, different terms refer to the notion of method adaptation, including 'method tailoring', 'method fragment adaptation' and 'situational method engineering'. Method tailoring is defined as:

A process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments. [31](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Aydin2004-31)

Potentially, almost all agile methods are suitable for method tailoring. Even the [DSDM](http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method) method is being used for this purpose and has been successfully tailored in a [CMM](http://en.wikipedia.org/wiki/Capability_Maturity_Model) context. [32](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Abrahamsson2003-32)  Situation-appropriateness can be considered as a distinguishing characteristic between agile methods and traditional software development methods, with the latter being relatively much more rigid and prescriptive. The practical implication is that agile methods allow project teams to adapt working practices according to the needs of individual projects. Practices are concrete activities and products that are part of a method framework. At a more extreme level, the philosophy behind the method, consisting of a number of principles, could be adapted (Aydin, 2004). [31](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Aydin2004-31)

[Extreme programming](http://en.wikipedia.org/wiki/Extreme_programming) (XP) makes the need for method adaptation explicit. One of the fundamental ideas of XP is that no one process fits every project, but rather that practices should be tailored to the needs of individual projects. Partial adoption of XP practices, as suggested by [Beck](http://en.wikipedia.org/wiki/Kent_Beck), has been reported on several occasions. [29](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Abrahamsson2002-29)  [Mehdi Mirakhorli](http://portal.acm.org/citation.cfm?id=1370143.1370149&coll=ACM&dl=ACM&CFID=69442744&CFTOKEN=96226775,) proposes a tailoring practice that provides a sufficient road-map and guidelines for adapting all the practices. [RDP Practice](http://en.wikipedia.org/wiki/Extreme_Programming) is designed for customizing XP. This practice, first proposed as a long research paper in the APSO workshop at the ICSE 2008 conference, is currently the only proposed and applicable method for customizing XP. Although it is specifically a solution for XP, this practice has the capability of extending to other methodologies. At first glance, this practice seems to be in the category of static method adaptation but experiences with RDP Practice says that it can be treated like dynamic method adaptation. The distinction between static method adaptation and dynamic method adaptation is subtle. [33](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Aydin2005-33)

### Comparison with other methods

#### RAD

Agile methods have much in common with the [Rapid Application Development](http://en.wikipedia.org/wiki/Rapid_Application_Development) techniques from the 1980/90s as espoused by [James Martin](http://en.wikipedia.org/wiki/James_Martin_(author)) and others.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed) In addition to technology-focused methods, customer-and-design-centered methods, such as Visualization-Driven Rapid Prototyping developed by [Brian Willison](http://en.wikipedia.org/wiki/Brian_Willison), work to engage customers and end users to facilitate agile software development.[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

#### CMMI

In 2008 the [Software Engineering Institute](http://en.wikipedia.org/wiki/Software_Engineering_Institute) (SEI) published the technical report "CMMI or Agile: Why Not Embrace Both" [34](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-34)  to make clear that the [Capability Maturity Model Integration](http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration) and Agile can co-exist. Modern CMMI-compatible development processes are also iterative. The [CMMI Version 1.3](http://en.wikipedia.org/wiki/CMMI_Version_1.3) includes tips for implementing CMMI and agile process improvement together. [35](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-35)

## **Large-scale and distributed agile development**

Agile development has been widely seen as highly suited to certain types of environments, including small teams of experts working on [greenfield projects](http://en.wikipedia.org/wiki/Greenfield_project), [24](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-boehm2004-24)  [36](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-beck1999-36) :157 and the challenges and limitations encountered in the adoption of agile methods in a large organization with [legacy infrastructure](http://en.wikipedia.org/wiki/Legacy_system) are well-documented and understood. [37](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-37)

In response, a range of strategies and patterns has evolved for overcoming challenges with large-scale development efforts (>20 developers) [38](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-ambler2006-38)  [39](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-sstc2007-39)  or distributed (non-colocated) development teams, amongst other challenges; and there are now several recognised frameworks that seek to mitigate or avoid these challenges, including:

* Scaled agile framework (SAFe), Dean Leffingwell *inter alia*
* [Disciplined agile delivery](http://en.wikipedia.org/wiki/Disciplined_agile_delivery) (DAD), [Scott Ambler](http://en.wikipedia.org/wiki/Scott_Ambler) *inter alia*
* [Large-scale scrum (LeSS)](http://en.wikipedia.org/wiki/Large-Scale_Scrum_(LeSS)), [Craig Larman](http://en.wikipedia.org/wiki/Craig_Larman) and Bas Vodde
* Scaled professional scrum, [Jeff Sutherland](http://en.wikipedia.org/wiki/Jeff_Sutherland) and [Ken Schwaber](http://en.wikipedia.org/wiki/Ken_Schwaber)
* Enterprise scrum, Mike Beedle

There are many conflicting viewpoints on whether all of these are effective or indeed fit the definition of agile development, and this remains an active and ongoing area of research. [38](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-ambler2006-38)  [42](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-oopsla2002-42)

## **Experience and adoption**

### Measuring agility

### While agility can be seen as a means to an end, a number of approaches have been proposed to quantify agility. *Agility Index Measurements* (AIM) [43](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-43)  score projects against a number of agility factors to achieve a total. The similarly named *Agility Measurement Index*, [44](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-44)  scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals. [45](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-45)  Another study using fuzzy mathematics [46](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Kurian_2006-46)  has suggested that project velocity can be used as a metric of agility. There are agile self-assessments to determine whether a team is using agile practices (Nokia test, [47](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-47)  Karlskrona test, [48](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-48)  42 points test [49](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-49) ).

While such approaches have been proposed to measure agility, the practical application of such metrics is still debated. There is agile software development ROI data available from the CSIAC ROI Dashboard. [50](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-CSIAC_ROI-50)

### Surveys

One of the early studies reporting gains in quality, productivity, and business satisfaction by using Agile methods was a survey conducted by Shine Technologies from November 2002 to January 2003. [51](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-51)  A similar survey, the State of Agile, is conducted every year starting in 2006 with thousands of participants from around the software development community. The State of Agile survey tracks trends on the benefits of agile, trends, lessons learned, preferred practices and agile methodologies. From the 2013 results released in January 2014, the survey concludes that 73% of respondents say agile software helps them complete software projects faster; 92% say agile improves their ability to manage changing customer priorities; and 87% say agile improves their development team's productivity. [52](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-52)  Yet another survey conducted in 2006 by Scott Ambler, the Practice Leader for Agile Development with IBM Rational's Methods Group reported similar benefits. [53](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-53)  Others claim that agile development methods are still too young to require extensive academic proof of their success. [54](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-54)

### Common agile pitfalls

Organizations and teams implementing agile development often face difficulties transitioning from more traditional methods such as [waterfall development](http://en.wikipedia.org/wiki/Waterfall_development), such as teams having an agile process forced on them. [55](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-The_Art_of_Agile_Development-55)  These are often termed *agile anti-patterns* or more commonly *agile smells*. Below are some common examples:

#### Lack of overall project design

This can lead to lack of scalability and design flaws—like building a house with no blueprint and hoping it remains standing.

#### Adding stories to a sprint in progress

Adding stories to a sprint in progress is detrimental to the flow established by Agile. From Ilan Goldstein's *Sprint issues – when sprints turn into crawls*, "'Isn't the ability to change course on the fly what Scrum is all about?' Well not quite. Scrum certainly provides provision to change product backlog priorities mid-project however this needs to occur between sprints and not during them." [56](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-axisagile.com.au-56)

If an issue arises that requires additions to a sprint, Ilan recommends an abnormal sprint termination. [56](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-axisagile.com.au-56)  This does not mean that a user story cannot expand. Teams must deal with new information, which may produce additional tasks for a user story. If the new information prevents the user story from being production-ready during the sprint, then it should carry over to the next sprint. However, during the next sprint planning, the user story should be prioritized over all remaining user stories. The information that requires new tasks may have altered the user story priority.

#### Lack of sponsor support

Agile is often implemented as a grassroots efforts in organizations by software development teams trying to optimize their development processes. By not having sponsor support, teams may face difficulties and resistance from business partners, other development teams and management. "Carrying on without an effective sponsor raises the probability of project failure..." [57](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-57)

Additionally, the project sponsor is responsible for ensuring the team has appropriate funding and resources. [58](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-58)

#### Insufficient training

A survey performed by Version One found respondents cited insufficient training as the most significant cause for failed agile projects [59](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-59)  Teams have fallen into the trap of assuming the reduced processes of agile development compared to other methodologies such as waterfall mean there are no actual rules for agile development. Agile development is a set of prescribed methodologies, and training/practice is a requirement.

#### Product owner role is not properly filled

The [product owner](http://en.wikipedia.org/wiki/Product_owner) is responsible for representing the business in the development activity. In The Elements of Scrum the product owner "... is usually the most demanding role on a scrum team." [60](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-60)

A common mistake is to have the product owner role filled by someone from the development team. According to Johanna Rothman this is a mistake, "When the business is unaccountable, the agile ecosystem breaks down." [61](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-61)  Having the development team fill this role results in the team making its own decisions on prioritization without real feedback from the business. Additionally, the team either tries to solve business issues internally or delay as they reach outside the core group for input. This can cause finger-pointing and divert from the collaborative process directed.

#### Teams are not focused

The agile process requires teams who focus on the project to meet project commitments. During a sprint, a resource who has the capacity is expected to take-up tasks potentially outside their subject area. If team members have multiple projects, it is difficult to make spare capacity available to help complete the sprint. "While having information developers working on multiple scrum teams is not ideal, it can be done with some proper planning and judicious evaluation of which meetings you should attend." [62](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-62)

#### Excessive preparation/planning

Teams may fall into the trap of spending too much time preparing or planning. This is a common trap for teams less familiar with the agile process where the teams feel obligated to have a complete understanding of all user stories or a detailed design. Teams should leverage the ability for Sprints to act as a method discovery and moving forward with the information they do know. As more information is gained it should be applied to the next Sprint.

#### Problem-solving in the daily scrum

The daily scrum should be a focused, timely meeting where all team members disseminate information. If problem-solving occurs, it often can only involve certain team members and potentially is not the best use of the entire team's time. If during the daily scrum the team starts diving into problem-solving, it should be tabled until a sub-team can discuss immediately after the daily scrum completes. [63](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-63)

#### Assigning Tasks

One of the intended benefits of agile development is to empower the team to make choices, as they are closest to the problem. Additionally, they should make choices as close to implementation as possible—compared to a waterfall approach—to use more timely information in the decision. If team members are assigned tasks by others or too early in the process, the benefits of localized and timely decision making can be lost. [64](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Effective_Sprint_Planning-64)

Another tendency is for assigners to box team members into certain roles (for example, team member A must always do the database work), which hinders cross-training. [64](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Effective_Sprint_Planning-64)  Team members themselves can choose to take on tasks that stretch their abilities and provide cross-training opportunities.

#### Scrum master as a contributor

Another common pitfall is for the scrum master to act as a contributor. While not prohibited by the agile methodology, the scrum master needs to ensure they have the capacity to act in the role of scrum master first and not working on tasks for the project. A scrum master's role is to facilitate the scrum process. "Facilitating meetings such as a daily scrum, sprint planning, sprint reviews and sprint retrospectives is part of this. A technical contributor's role is to work with the other team members to figure out how to get the work done and to do it." [65](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-agileconnection.com-65)

Having the scrum master also multitasking may result in too many context switches to be productive. Additionally, as a scrum master is to remove roadblocks for the team, the benefit gained by individual tasks moving forward may not outweigh roadblocks that are deferred due to lack of capacity. [65](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-agileconnection.com-65)

#### Lacking test automation

Due to the iterative nature of agile development, multiple rounds of testing for a project are often needed. "Having an automated testing framework, which takes care of both system and integration tests, adds a lot of firepower to such a team. It not only acts as a safety net against regressions caused by new development, but more importantly frees up a lot of precious developer and tester time - allowing them to focus on the things they do best." [66](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-66)

Test automation also supports continued [refactoring](http://en.wikipedia.org/wiki/Refactoring) required by iterative software development. Allowing a developer to quickly run tests to confirm refactoring has not modified the functionality of the application may reduce the workload and increase confidence that cleanup efforts have not introduced new defects.

#### Allowing technical debt to build up

Focusing on delivering new functionality may result in increased [technical debt](http://en.wikipedia.org/wiki/Technical_debt). The team must allow themselves time for defect remediation and refactoring. Technical debt hinders planning abilities by increasing the amount of unscheduled work as production defects distract the team from further project progress. [67](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Technical_Debt_.2B_Red_October-67)

As the system evolves it is important to [refactor](http://en.wikipedia.org/wiki/Code_refactoring) as entropy of the system naturally increases. [68](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-68)  Over time the lack of constant maintenance causes increasing defects and development costs. [67](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Technical_Debt_.2B_Red_October-67)

#### Attempting to take on too much in a sprint

A common misconception is that agile development allows continuous change, however a sprint backlog is an agreement of what work can be completed during the sprint. [69](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-69)  Additionally having too much [work-in-progress(WIP)](http://en.wikipedia.org/wiki/Work_in_process) can result in inefficiencies due "to avoid the penalties of wasted time, effort and resources." [70](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-70)

A possible issue is the team being pressured into taking on additional work. "An important point to reiterate here is that it's the team that selects how much work they can do in the coming sprint. The product owner does not get to say, 'We have four sprints left so you need to do one-fourth of everything I need.' We can hope the team does that much (or more), but it's up to the team to determine how much they can do in the sprint." [71](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-71)

#### Fixed time, resources, scope and quality

Agile development fixes time (sprint duration) and resources while the scope and quality remain variable. [72](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-adeptechllc.com-72)  The customer or product owner often pushes for a fixed scope for a sprint. However, teams should be reluctant to commit to locked time, resources and scope (commonly known as the [project management triangle](http://en.wikipedia.org/wiki/Project_management_triangle)). Efforts to add scope to the fixed time and resources of agile development may result in decreased quality. [72](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-adeptechllc.com-72)

### Criticism

Agile methodologies can be inefficient in large organizations and certain types of projects. Agile methods seem best for developmental and non-sequential projects. Many organizations believe that agile methodologies are too extreme and adopt a hybrid approach that mixes elements of agile and plan-driven approaches. [73](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-73)

The term "agile" has also been criticized as being a [management fad](http://en.wikipedia.org/wiki/Management_fad) that simply describes existing good practices under new jargon, promotes a "one size fits all" mindset towards development strategies, and wrongly emphasizes method over results. [74](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-74)

[Alistair Cockburn](http://en.wikipedia.org/wiki/Alistair_Cockburn) organized a celebration of the 10th anniversary of the [Agile Manifesto](http://en.wikipedia.org/wiki/Agile_software_development#The_Agile_Manifesto) in Snowbird, Utah on February 12, 2011, gathering some 30+ people who had been involved at the original meeting and since. A list of about 20 [elephants in the room](http://en.wikipedia.org/wiki/Elephant_in_the_room) (“undiscussable” agile topics/issues) were collected, including aspects: the alliances, failures and limitations of agile practices and context (possible causes: commercial interests, decontextualization, no obvious way to make progress based on failure, limited objective evidence, cognitive biases and reasoning fallacies), politics and culture. [75](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-75)  As [Philippe Kruchten](http://en.wikipedia.org/wiki/Philippe_Kruchten) wrote:

*The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and also therefore more effective.*

### Application outside software development

Agile methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies. [76](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-Smith2007-76)  However, these techniques can be applied to the development of non-software products, such as computers, motor vehicles, medical devices, food, clothing, and music; [77](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-77)  see [Flexible product development](http://en.wikipedia.org/wiki/Flexible_product_development).

Agile development paradigms can be used in other areas of life such as raising children. Its success in child development might be founded on some basic management principles; communication, adaptation and awareness. Bruce Feiler has shown that the basic Agile Development paradigms can be applied to household management and raising children. In his [TED Talk, "Agile programming -- for your family"](http://www.ted.com/talks/bruce_feiler_agile_programming_for_your_family.html), these paradigms brought significant changes to his household environment, such as the kids doing dishes, taking out the trash, and decreasing his children's emotional outbreaks, which inadvertently increased their emotional stability. In some ways, agile development is more than a bunch of software development rules: it can be something more simple and broad, like a problem solving guide.

# Scrum (software development)

**Scrum** is an iterative and incremental [agile software development](http://en.wikipedia.org/wiki/Agile_software_development) methodology for managing product development. It defines "a flexible, [holistic](http://en.wikipedia.org/wiki/Holism" \o "Holism)product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

A key principle of scrum is its recognition that during a project the customers can change their minds about what they want and need (often called "requirements churn"), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. As such, scrum adopts an [empirical](http://en.wikipedia.org/wiki/Empirical) approach—accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements.

## **History**

Scrum was first defined as "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach" in 1986 by [Hirotaka Takeuchi](http://en.wikipedia.org/wiki/Hirotaka_Takeuchi" \o "Hirotaka Takeuchi) and [Ikujiro Nonaka](http://en.wikipedia.org/wiki/Ikujiro_Nonaka" \o "Ikujiro Nonaka) in the *New New Product Development Game*. Takeuchi and Nonaka later argued in *The Knowledge Creating Company* that it is a form of "organizational knowledge creation, ... especially good at bringing about innovation continuously, incrementally and spirally".

The authors described a new approach to commercial [product development](http://en.wikipedia.org/wiki/New_product_development) that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. [3](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-Takeuchi-3)  They called this the [*holistic*](http://en.wikipedia.org/wiki/Holistic) or [*rugby*](http://en.wikipedia.org/wiki/Rugby_football)*approach*, as the whole process is performed by one [cross-functional team](http://en.wikipedia.org/wiki/Cross-functional_team) across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth". [3](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-Takeuchi-3)  (In [rugby football](http://en.wikipedia.org/wiki/Rugby_football), a [scrum](http://en.wikipedia.org/wiki/Scrum_(rugby)) refers to a tight-packed formation of players with their heads down who attempt to gain possession of the ball. [4](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-4) )

In the early 1990s, [Ken Schwaber](http://en.wikipedia.org/wiki/Ken_Schwaber) used what would become scrum at his company, Advanced Development Methods, and [Jeff Sutherland](http://en.wikipedia.org/wiki/Jeff_Sutherland), with John Scumniotales and Jeff McKenna, developed a similar approach at Easel Corporation, and were the first to refer to it using the single word *scrum*. [5](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-autogenerated1-5)  In 1995, Sutherland and Schwaber jointly presented a paper describing the *scrum methodology* at the Business Object Design and Implementation Workshop held as part of [Object-Oriented Programming, Systems, Languages & Applications '95](http://en.wikipedia.org/wiki/OOPSLA) (OOPSLA '95) in Austin, Texas, its first public presentation. [6](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-6)  Schwaber and Sutherland collaborated during the following years to merge the above writings, their experiences, and industry best practices into what is now known as scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book *Agile Software Development with Scrum*. [7](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-7)  Its approach to planning and managing projects is to bring [decision-making](http://en.wikipedia.org/wiki/Decision-making) authority to the level of operation properties and certainties. [8](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-schwaber-8)  Although the word is not an [acronym](http://en.wikipedia.org/wiki/Acronym), some companies implementing the process have been known to spell it with capital letters as SCRUM. This may be due to one of Ken Schwaber's early papers, which capitalized SCRUM in the title. [8](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-schwaber-8)

Later, Schwaber with others founded the Scrum Alliance and created the Certified Scrum Master programs and its derivatives. Schwaber left the Scrum Alliance in the fall of 2009, and founded Scrum.org to further improve the quality and effectiveness of scrum.

## **Roles**

There are three core roles [9](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-9)  with a range of ancillary roles. These core roles are those committed to the project in the scrum process—they are the ones producing the product (objective of the project). They represent the *scrum team*. Although other roles may be encountered in real projects, scrum does not define any team roles other than those described below. [10](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumguide-10)

### Product owner

The product owner represents the [stakeholders](http://en.wikipedia.org/wiki/Stakeholder_(corporate)) and is the [voice of the customer](http://en.wikipedia.org/wiki/Voice_of_the_customer). He or she is accountable for ensuring that the team delivers value to the business. The product owner writes (or has the team write) customer-centric items (typically [user stories](http://en.wikipedia.org/wiki/User_story)), ranks and prioritizes them, and adds them to the [product backlog](http://en.wikipedia.org/wiki/Scrum_(software_development)#Product_backlog). Scrum teams should have one product owner, and while they may also be a member of the development team, this role should not be combined with that of the scrum master. This role is equivalent to the [customer representative](http://en.wikipedia.org/wiki/Customer_representative) role in some other agile frameworks.

#### Role of product owner in defining and communicating product requirements

Communication is a main function of the product owner. The ability to convey priorities and empathize with team members and stakeholders are vital to steer the project in the right direction. Product owners bridge the communication gap between the team and their stakeholders

As the face of the team to the stakeholders, the following are some of the communication tasks of the product owner to the stakeholders:

* demonstrates the solution to key stakeholders who were not present in a normal iteration demo
* announces releases
* communicates team status
* organizes milestone reviews
* educates stakeholders in the development process
* negotiates priorities, scope, funding, and schedule
* ensures that the product backlog is visible, transparent, and clear

Empathy is a key attribute for a product owner to have – the ability to put one’s self in another’s shoes. A product owner will be conversing with different stakeholders in the project – different people, with a variety of backgrounds, job roles, and objectives. A product owner needs to be able to see from these different points of view. To be effective, it would also be wise for a product owner to know the level of detail the audience needs from him or her. The development team would need thorough feedback and specifications so they build a product up to expectation, while an executive sponsor may just need summaries of progress. Providing more information than necessary may lose stakeholder interest and waste time. There is also significant evidence that face-to-face communication around a shared sketching environment is the most effective way to communicate information instead of documentation. A direct means of communication is then most preferred by seasoned agile product owners. [12](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-12)

A product owner’s ability to communicate effectively is also enhanced by being skilled in techniques that identify stakeholder needs, negotiate priorities between stakeholder interests, and collaborate with developers to ensure effective implementation of requirements.

### Development team

The development team is responsible for delivering potentially shippable increments (PSIs) of product at the end of each sprint (the sprint goal). A team is made up of 3–9 individuals with cross-functional skills who do the actual work (analyse, design, develop, test, technical communication, document, etc.). The development team in scrum is self-organizing, even though there may be some level of interface with project management offices (PMOs).

### Scrum master

Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional[team lead](http://en.wikipedia.org/wiki/Team_leader) or [project manager](http://en.wikipedia.org/wiki/Project_manager), but acts as a buffer between the team and any distracting influences. The scrum master ensures that the scrum process is used as intended. The scrum master helps ensure the team follows the agreed scrum processes, often facilitates key sessions, and encourages the team to improve. The role has also been referred to as a *team facilitator* [13](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-13)  or[*servant-leader*](http://en.wikipedia.org/wiki/Servant_leadership) to reinforce these dual perspectives.

The core responsibilities of a scrum master include (but are not limited to): [14](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-14)

* Helping the product owner maintain the product backlog in a way that ensures the project is well defined and the team can continually advance forward on the project at any given time
* Helping the team to determine the *definition of done* for the product, with input from key stakeholders
* Coaching the team, within the scrum principles, in order to deliver quality valuable features for their product
* Promoting self-organization within the team
* Helping the scrum team to avoid or remove impediments to their progress, whether internal or external to the team
* Facilitating team events to ensure regular progress
* Educating key stakeholders in the product on scrum principles

One of the ways in which the scrum master role differs from that of a project manager is that the latter may have [people management](http://en.wikipedia.org/wiki/Management) responsibilities while the scrum master does not. Scrum does not formally recognise the role of project manager, as traditional command and control tendencies would cause difficulties, [15](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumprimer-15)  however, it is possible for scrum teams to work effectively with agile project managers, especially on large-scale programs. [16](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-16)

## **Events**

### Sprint

A sprint (or iteration) is the basic unit of development in scrum. The sprint is *[timeboxed](http://en.wikipedia.org/wiki/Timeboxing" \o "Timeboxing)* effort; that is, it is restricted to a specific duration. [17](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-Sprint_2009-17)  The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common. [8](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-schwaber-8)

Each sprint starts with a sprint planning event, the aim of which is to define a sprint backlog, where the work for the sprint is identified and an estimated commitment for the sprint goal is made. Each sprint ends with a sprint review and a sprint retrospective, [5](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-autogenerated1-5)  where the progress is reviewed and shown to stakeholders and improving lessons for the next sprints are identified.

Scrum emphasizes working product at the end of the sprint that is really *done*; in the case of software, this would likely include that the software has been integrated, fully tested, end-user documented, and is potentially shippable. [15](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumprimer-15)

### Events

#### Sprint planning

At the beginning of a sprint, the team hold a *sprint planning event*: [17](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-Sprint_2009-17)

* Select what work is to be done
* Prepare the sprint backlog that details the time it will take to do that work, with the entire team
* Identify and communicate how much of the work is likely to be done during that sprint
* Four-hour time limit for a two-week sprint (pro rata for other sprint durations)  [10](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumguide-10) 
  + During the first half, the whole team (development team, scrum master, and product owner) agree what product backlog items they will consider for that sprint
  + During the second half, the development team establish the work (tasks) required to deliver those backlog items, resulting in the *sprint backlog*

#### Daily scrum

Each day during a sprint, the team hold a *daily scrum* (or [*stand-up*](http://en.wikipedia.org/wiki/Stand-up_meeting)) with specific guidelines:

* All members of the development team come prepared.
* The daily scrum starts precisely on time even if some development team members are missing.
* The daily scrum should happen at the same location and same time every day.
* The daily scrum length is constrained ([timeboxed](http://en.wikipedia.org/wiki/Timeboxing" \o "Timeboxing)) to fifteen minutes.
* Anyone is welcome, although normally only the scrum team roles contribute.

During the daily scrum, each team member answers three questions:

* What did I do yesterday that helped the development team meet the sprint goal?
* What will I do today to help the development team meet the sprint goal?
* Do I see any impediment that prevents me or the development team from meeting the sprint goal?

Any impediment (stumbling block, risk, or issue) identified in the daily scrum should be captured by the scrum master, displayed on the team's scrum board, and someone agreed to be responsible for working toward a resolution outside of the daily scrum. No detailed discussions should happen during the daily scrum.

#### Sprint review and sprint retrospective

At the end of a sprint, the team holds two events: the *sprint review* and the *sprint retrospective*.

At the **sprint review**, the team:

* Reviews the work that was completed and the planned work that was not completed
* Presents the completed work to the stakeholders (a.k.a. the *demo*)

Guidelines for sprint reviews:

* Incomplete work cannot be demonstrated
* The recommended duration is two hours for a two-week sprint (pro-rata for other sprint durations)

At the **sprint retrospective**, the team:

* Reflects on the past sprint
* Identifies and agrees continuous process improvement actions

Guidelines for sprint retrospectives:

* Two main questions are asked in the sprint retrospective: What went well during the sprint? What could be improved in the next sprint?
* The recommended duration is one-and-a-half hours for a two-week sprint (pro-rata for other sprint durations)
* This event is facilitated by the scrum master

### Extensions

#### Backlog refinement

Backlog refinement (which used to be called *backlog grooming*) is the ongoing process of reviewing product backlog items and checking that they are appropriately prioritised and prepared in a way that makes them clear and executable for teams once they enter sprints via the sprint planning activity. Product backlog items may be broken into multiple smaller ones; acceptance criteria may be clarified; and dependencies, investigation, and preparatory work may be identified and agreed as *technical spikes*.

Backlog refinement is not a core scrum practice but has been adopted as a way of managing the quality of backlog items entering a sprint, with a recommended duration of up to 10% of the sprint capacity. [18](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-18)

#### Scrum of scrums

The *scrum of scrums* is a technique to scale scrum up for multiple teams working on the same product, allowing teams to discuss progress on their interdependencies, focusing on how to coordinate delivering software, [19](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumofscrums-19)  especially on areas of overlap and integration. Depending on the cadence (timing) of the scrum of scrums, the relevant daily scrum for each scrum team ends by designating one member as an *ambassador* to participate in the scrum of scrums with ambassadors from other teams. Depending on the context, the ambassadors may be technical contributors or each team’s scrum master. [19](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumofscrums-19)

This should run similar to a daily scrum, with each ambassador answering the following four questions:[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)

* What impediments have my team resolved since we last met?
* What impediments will my team resolve before we meet again?
* Are there any impediments slowing my team down or getting in our way?
* Are we about to put something in another team's way?

Resolution of impediments is expected to focus on the challenges of coordination between the teams, and may entail agreeing to interfaces between teams, negotiating responsibility boundaries, etc.

The scrum of scrums will track these working items via a backlog of its own, such as a ROAM board, where each item contributes to improving cross-team coordination. [19](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumofscrums-19)

As [Jeff Sutherland](http://en.wikipedia.org/wiki/Jeff_Sutherland) commented, [19](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumofscrums-19)

*Since I originally defined the Scrum of Scrums (Ken Schwaber was at IDX working with me), I can definitively say the Scrum of Scrums is not a "meta Scrum". The Scrum of Scrums as I have used it is responsible for delivering the working software of all teams to the Definition of Done at the end of the Sprint, or for releases during the sprint. PatientKeeper delivered to production four times per Sprint. Ancestry.com delivers to production 220 times per two week sprint. Hubspot delivers live software 100-300 times a day. The Scrum of Scrums Master is held accountable for making this work. So the Scrum of Scrums is an operational delivery mechanism.*

## **Artifacts**

### Product backlog

The *product backlog* is an ordered list of [*requirements*](http://en.wikipedia.org/wiki/Requirement) that is maintained for a [product](http://en.wikipedia.org/wiki/New_product_development). It consists of [features](http://en.wikipedia.org/wiki/Software_feature), [bug fixes](http://en.wikipedia.org/wiki/Patch_(computing)), [non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirement), etc.—whatever needs to be done in order to successfully deliver a viable product. The *product backlog items* (PBIs) are ordered by the product owner based on considerations like risk, business value, dependencies, date needed, etc.

Items added to a backlog are commonly written in story format. The product backlog is *what* will be delivered, ordered into the sequence in which it should be delivered. It is open and editable by anyone, but the product owner is ultimately responsible for ordering the items on the backlog for the development team to choose.

The product backlog contains the product owner's assessment of business value and the development team's assessment of development effort, which are often, but not always, stated in [story points](http://en.wikipedia.org/wiki/Planning_poker) using a rounded [Fibonacci](http://en.wikipedia.org/wiki/Fibonacci_number) sequence. These estimates help the product owner to gauge the timeline and may influence ordering of backlog items; for example, if the "add spellcheck" and "add table support" features have the same business value, the product owner may schedule earlier delivery of the one with the lower development effort (because the [return on investment](http://en.wikipedia.org/wiki/Return_on_investment) is higher) or the one with higher development effort (because it is more complex or riskier, and they want to retire that risk earlier). [21](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-21)

The product backlog and the business value of each backlog item is the responsibility of the product owner. The size (i.e. estimated complexity or effort) of each backlog item is, however, determined by the development team, who contributes by sizing items, either in story points or in estimated hours.

There is a common misunderstanding that only [user stories](http://en.wikipedia.org/wiki/User_story) are allowed in a product backlog. By contrast, scrum is neutral on requirement techniques. As the *Scrum Primer* [15](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumprimer-15)  states,

*Product Backlog items are articulated in any way that is clear and sustainable. Contrary to popular misunderstanding, the Product Backlog does not contain "user stories"; it simply contains items. Those items can be expressed as user stories,*[*use cases*](http://en.wikipedia.org/wiki/Use_case)*, or any other requirements approach that the group finds useful. But whatever the approach, most items should focus on delivering value to customers.*

Scrum advocates that the role of product owner be assigned. The product owner is responsible for maximizing the value of the product and the work of the development team. The product owner gathers input, takes feedback and is lobbied by many people, but will ultimately make the call on what gets built. They are also solely responsible for the management of the backlog.

The product backlog is used to:

* capture requests for modifying a product. This can include adding new features, replacing old features, removing features and fixing issues
* ensure the development team is given work which maximizes the business benefit to the owner of the product

Typically, the product owner and the scrum team come together and write down everything that needs to be prioritized and this becomes content for the first sprint, which is a block of time meant for focused work on selected items that can be accommodated within a timeframe. The product backlog is permitted to evolve as new information surfaces about the product and its customers, and so new work are tackled for next sprints.

The following items typically comprise a scrum backlog: features, bugs, technical work, and knowledge acquisition. In the web development sphere, there is confusion as to the difference between a feature and a bug, technically a feature is “wanted”, while a bug is a feature that is “unintended” or “unwanted” (but may not be necessarily a defective thing). An example of technical work would be, “run virus check on all developers workstations”. An example of knowledge acquisition could be a scrum backlog item about researching Wordpress plugin libraries and making a selection.

#### Managing the product backlog between product owner and scrum team

A backlog, in its simplest form, is merely a list of items to be worked on. Having well established rules about how work is added, removed and ordered helps the whole team make better decisions about how to change the product.

The product owner prioritizes which of the items in the product backlog are most needed. The team then chooses which items can be completed in the coming sprint. On the scrum board, the team moves items from the product backlog to the sprint backlog, which is the list of items they will build. Conceptually, it is ideal for the team to only select what they think they can accomplish from the top of the list, but it is not unusual to see in practice that teams are able to take lower priority items from the list along with the top ones selected. This normally happens because there is time left within the sprint to accommodate more work. Items at the top of the backlog, the items that are going to be worked on first, should be broken down into stories that are suitable for the development team to work on. The further down the backlog goes, the less refined the items should be. As Schwaber and Beedle put it “The lower the priority, the less detail, until you can barely make out the backlog item.” [8](http://en.wikipedia.org/wiki/Scrum_(software_development)" \l "cite_note-schwaber-8)

As the team works through the backlog, it needs to be assumed that “changes in the world can happen”—the team can learn about new market opportunities to take advantage of, competitor threats that arise, and feedback from customers that can change the way the product was meant to work. All of these new ideas tend to trigger the team to adapt the backlog to incorporate new knowledge. This is part of the fundamental mindset of an agile team. The world changes, the backlog is never finished. [11](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-Pichler.2C_Roman_2010-11)

### Sprint backlog

The *sprint backlog* is the list of work the development team must address during the next sprint. The list is derived by selecting product backlog items from the top of the product backlog until the development team feels it has enough work to fill the sprint. This is done by the development team asking "Can we also do this?" and adding product backlog items to the sprint backlog. The development team should keep in mind its past performance assessing its capacity for the new sprint, and use this as a guide line of how much "effort" they can complete.

The product backlog items are broken down into tasks by the development team. Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed according to the set priority and the development team member skills. This promotes self-organization of the development team, and developer buy-in.

The sprint backlog is the property of the development team, and all included estimates are provided by the development team. Often an accompanying*task board* is used to see and change the state of the tasks of the current sprint, like "to do", "in progress" and "done".

Once a sprint backlog is committed, no additional functionality can be added to the sprint backlog except by the team. Once a sprint has been delivered, the product backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next sprint.

### Product increment

The *increment* (or *potentially shippable increment*, PSI) is the sum of all the product backlog items completed during a sprint and all previous sprints. At the end of a sprint, the increment must be complete, according to the scrum team's Definition of Done (DoD), and in a usable condition regardless of whether the product owner decides to actually release it.

### Sprint burn-down chart

*Main article:*[*Burn down chart*](http://en.wikipedia.org/wiki/Burn_down_chart)

The sprint burndown chart is a public displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. During sprint planning the ideal burndown chart is plotted. Then, during the sprint, each member picks up tasks from the sprint backlog and works on them. At the end of the day, they update the remaining hours for tasks to be completed. In such way the actual burndown chart is updated day by day.

It should not be confused with an [earned value chart](http://en.wikipedia.org/wiki/Earned_value_management).

### Release burn-down chart

The release burndown chart is the way for the team to track progress and provide visibility. The release burndown chart is updated at the end of each sprint by the scrum master. The horizontal axis of the release burndown chart shows the sprints; the vertical axis shows the amount of work remaining at the start of each sprint. The release burndown chart makes it easy to drill down into a sprint and understand what is the remaining work, what work has been added, what work has been done, what work will have to be done. You can see what the team has completed as well as how scope changed.

## **Scrum-ban**

*Main article:*[*Scrum ban*](http://en.wikipedia.org/wiki/Scrum_ban)

Scrum-ban is a software production model based on scrum and [kanban](http://en.wikipedia.org/wiki/Kanban_(development)" \o "Kanban (development)). Scrum-ban is especially suited for maintenance projects or (system) projects with frequent and unexpected work items or programming errors. [27](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-27)  In such cases the time-limited sprints of the scrum model are of no appreciable use, but scrum's daily events and other practices can be applied, depending on the team and the situation at hand. Visualization of the work stages and limitations for simultaneous unfinished work and defects are familiar from the Kanban model. Using these methods, the team's[workflow](http://en.wikipedia.org/wiki/Workflow) is directed in a way that allows for minimum completion time for each work item or programming error, and on the other hand ensures each team member is constantly employed. [28](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-28)

To illustrate each stage of work, teams working in the same space often use post-it notes or a large whiteboard. [29](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-29)  In the case of decentralized teams, stage-illustration software such as[Assembla](http://en.wikipedia.org/wiki/Assembla), [TargetProcess](http://en.wikipedia.org/wiki/TargetProcess" \o "TargetProcess), [Eylean Board](http://en.wikipedia.org/wiki/Eylean_Board" \o "Eylean Board), [JIRA](http://en.wikipedia.org/wiki/JIRA) or [Agilo for Scrum](http://en.wikipedia.org/wiki/Agilo_for_Scrum" \o "Agilo for Scrum).

In their simplest, the tasks are categorized into the work stages:

* Unstarted
* Ongoing
* Completed

If desired, though, the teams can add more stages of work (such as "defined", "designed", "tested" or "delivered"). These additional phases can be of assistance if a certain part of the work becomes a bottleneck and the limiting values of the unfinished work cannot be raised. A more specific task division also makes it possible for employees to specialize in a certain phase of work. [30](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumban-30)

There are no set limiting values for unfinished work. Instead, each team has to define them individually by trial and error; a value too small results in workers standing idle for lack of work, whereas values too high tend to accumulate large amounts of unfinished work, which in turn hinders completion times. [31](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-crisp-31)  A rule of thumb worth bearing in mind is that no team member should have more than two simultaneous selected tasks, and that on the other hand not all team members should have two tasks simultaneously. [30](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-scrumban-30)

The major differences between scrum and kanban are derived from the fact that, in scrum, work is divided into sprints that last a certain amount of time, whereas in Kanban the workflow is continuous. This is visible in work stage tables, which in scrum are emptied after each sprint. In Kanban all tasks are marked on the same table. Scrum focuses on teams with multifaceted know-how, whereas Kanban makes specialized, functional teams possible. [31](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-crisp-31)

Since scrum-ban is such a new development model, there is not much reference material. Kanban, on the other hand, has been applied by Microsoft and Corbis. [32](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-32)

## **Tooling**

Like other agile methods, scrum can be implemented through a wide range of tools.

Many companies use universal tools, such as spreadsheets to build and maintain artifacts such as the sprint backlog. There are also open-source and proprietary packages dedicated to management of products under the scrum process. Other organizations implement scrum without the use of any software tools, and maintain their artifacts in hard-copy forms such as paper, whiteboards, and sticky notes. [33](http://en.wikipedia.org/wiki/Scrum_(software_development)#cite_note-33)

## **Limitations**

The hybridization of scrum is common as scrum does not cover the whole [product development lifecycle](http://en.wikipedia.org/wiki/Software_lifecycle); therefore, organizations find the need to add in additional processes to create a more comprehensive implementation. For example, at the start of the project, organizations commonly add process guidance on requirements gathering and prioritization, initial high-level design, and budget and schedule forecasting. However, the scrum framework does not explicitly allow for extension points of such a kind; consequently, achieving a more comprehensive software life cycle requires extending the framework rather than instantiating it.

# PRINCE2

**PRINCE2** (an acronym for **PRojects IN Controlled Environments**, version ) is a [project management](http://en.wikipedia.org/wiki/Project_management) [methodology](http://en.wikipedia.org/wiki/Methodology). The methodology encompasses the high level management, control and organisation of a project, but not lower level activities such as scheduling. PRINCE2 is also used to refer to the training and accreditation of authorised practitioners of the methodology who must undertake accredited qualifications to obtain certification.

## **History**

PRINCE2 derives from an earlier method called PROMPT and from the PRINCE project-management method, which was initially developed in 1989 by the Central Computer and Telecommunications Agency (CCTA) as a UK Government standard for information systems (IT) project management. PRINCE is an acronym for "PRojects IN Controlled Environments". However, it soon became regularly applied outside the purely IT environment, both in UK government and in the private sector around the world. PRINCE2 was released in 1996 as a generic project management method. PRINCE2 has become increasingly popular and is now a *de facto* standard for project management in many UK government departments and across the [United Nations system](http://en.wikipedia.org/wiki/United_Nations_system). [3](http://en.wikipedia.org/wiki/PRINCE2#cite_note-3)

In July 2013, ownership of the rights to PRINCE2 was transferred from HM Cabinet Office to AXELOS Ltd, a joint venture between the Cabinet Office and Capita plc.

Since 2006, the method has been revised. It launched as "PRINCE2:2009 Refresh" in 2009. The name "PRINCE2" (instead of "PRINCE3" or similar) remains to indicate that the method remains faithful to its principles. Nevertheless, it is a fundamental revision of the method from 1996 to adapt it to the changed business environment, to make the method simpler and lighter, to address current weaknesses or misunderstandings, and to better integrate it with other AXELOS Global Best Practice methods ([ITIL](http://en.wikipedia.org/wiki/ITIL), P3O, [P3M3](http://en.wikipedia.org/wiki/P3M3), MSP, M\_o\_R etc.).

The main difference between the 2009 version and earlier versions is that there are two manuals:

1. 'Managing Successful Projects with PRINCE2 - 2009 Edition', [4](http://en.wikipedia.org/wiki/PRINCE2#cite_note-4)
2. 'Directing Successful Projects with PRINCE2 - 2009 Edition'

Both the Foundation and Practitioner Examinations are based on the new 'Managing Projects' manual and do not include material from the new 'Directing Successful Projects' book. [5](http://en.wikipedia.org/wiki/PRINCE2#cite_note-5)

## **Overview of the method**

PRINCE2 is a process-driven project management method. [6](http://en.wikipedia.org/wiki/PRINCE2#cite_note-6)

PRINCE2 is based on seven principles (continued business justification, learn from experience, defined roles and responsibilities, manage by stages, manage by exception, focus on products and tailored to suit the project environment), seven themes (business case, organization, quality, plans, risk, change and progress) and seven processes. The principles and themes come into play in the seven processes: [7](http://en.wikipedia.org/wiki/PRINCE2#cite_note-7)

### Starting up a project (SU)

In this process the project team is appointed and a project brief is produced.

In addition the overall approach to be taken is decided and the next stage (initiation) of the project is planned. Once this work is done, the project board is asked to authorize that stage.

Key activities include: Forming project board; appointing an executive and a [project manager](http://en.wikipedia.org/wiki/Project_manager); designing and appointing a project management team; preparing a project brief; defining the project approach; consulting the Lessons Logs of previous projects; and planning the next stage (initiation).

### Initiating a project (IP)

This process builds on the work of the start up process, and the project brief is augmented to form a [business case](http://en.wikipedia.org/wiki/Business_case). The approach taken to ensure quality throughout the project is agreed together with the overall approach to controlling the project itself (project controls). Project files are also created as is an overall plan for the project. A plan for the next stage of the project is also created. The resultant information can be put before the project board for them to authorize the project itself.

Key activities include: planning quality; planning a project; refining the business case and risks; setting up project controls; setting up project files; and assembling a [Project Initiation Documentation](http://en.wikipedia.org/wiki/Project_Initiation_Documentation).

### Directing a project (DP)

This process dictates how the Project Board (which comprises such roles as the [executive sponsor](http://en.wikipedia.org/wiki/Executive_sponsor) or project sponsor) should control the overall project. As mentioned above, the project board can authorise an initiation stage and can also authorize a project. Directing a Project also dictates how the project board should authorize a stage plan, including any stage plan that replaces an existing stage plan due to slippage or other unforeseen circumstances. Also covered is the way in which the board can give ad hoc direction to a project and the way in which a project should be closed down.

Key activities include: authorising initiation; authorising a project; authorising a stage or exception plan; giving ad hoc direction; and confirming project closure.

### Controlling a stage (CS)

PRINCE2 suggests that projects should be broken down into stages and these sub-processes dictate how each individual stage should be controlled. Most fundamentally this includes the way in which work packages are authorised and received. It also specifies the way in which progress should be monitored and how the highlights of the progress should be reported to the project board. A means for capturing and assessing project issues is suggested together with the way in which corrective action should be taken. It also lays down the method by which certain project issues should be escalated to the project board.

Key activities include: authorising work package; assessing progress; capturing and examining project issues; reviewing stage status; reporting highlights; taking corrective action; escalating project issues; and receiving a completed work package.

### Managing stage boundaries (SB)

*Main article:*[*Managing stage boundaries*](http://en.wikipedia.org/wiki/Managing_stage_boundaries)

Whereas the *Controlling a Stage* process dictates what should be done within a stage, *Managing Stage Boundaries* (SB) dictates what should be done towards the end of a stage. Most obviously, the next stage should be planned and the overall project plan, risk register and business case amended as necessary. The process also covers what should be done for a stage that has gone outside its tolerance levels. Finally, the process dictates how the end of the stage should be reported.

Key activities include: planning a stage; updating a project plan; updating a project business case; updating the risk register; reporting stage end; and producing an exception plan.

Best practice includes the project board including users reviewing progress and approving changes to the workplan at the boundary. This review does not include team managers or team workers because it wastes their time; and the project manager should be competent enough to present their work directly to the board.

### Managing product delivery (MP)

The *Managing product delivery* process has the purpose of controlling the link between the Project Manager and the Team Manager(s) by placing formal requirements on accepting, executing and delivering project work. [8](http://en.wikipedia.org/wiki/PRINCE2#cite_note-8)  The Objectives of the Managing Product Delivery process are:

* To ensure that work on products allocated to the team is authorised and agreed,
* Team Manager(s), team members and suppliers are clear as to what is to be produced and what is the expected effort, cost and timescales,
* The planned products are delivered to expectations and within tolerance,
* Accurate progress information is provided to the Project Manager at an agreed frequency to ensure that expectations are managed.

The key activities are: Accept a work package, execute a work package and deliver a work package.

### Closing a project (CP)

This covers the things that should be done at the end of a project. The project should be formally de-commissioned (and resources freed up for allocation to other activities), follow on actions should be identified and the project itself be formally evaluated.

Key activities include: decommissioning a project; identifying follow-on actions; and project evaluation review.

## **Management products**

PRINCE2 maintains several management products throughout the projects, which may be formal paper documents, word processor files, data in specialised PRINCE2 software, informal notes by managers, or even communicated orally. "Registers" are intended to be more formal than "logs". The following are examples of the PRINCE2 management products. [9](http://en.wikipedia.org/wiki/PRINCE2#cite_note-9)

### Project brief

A short explanation of the need for the project, the management team, the structure and goals.

### Business case

A detailed description of the need for the project and its expected benefits (to all stakeholders in the project, including its owner, users and suppliers and sometimes the general public (e.g. in public-funded projects, especially concerning pure research or the environment)).

### Risk register

*See also:*[*Risk register*](http://en.wikipedia.org/wiki/Risk_register)

The risk register is a table which contains the risks that may threaten the goal of the project. Risks are categorized, named and assigned a risk number. Then, each risk gets a probability score and an impact score. The multiplication of these two results in the risk score. The risks are sorted on descending risk score in order to show the priority of a risk. Each risk is also assigned a precaution and a response-action in case the risk still occurs.

### Quality register

*See also:*[*Quality control*](http://en.wikipedia.org/wiki/Quality_control)

This register will be updated to show the current status of all quality checking activities. It ensures that the delivered products are complete, have met their quality criteria and the agreed processes have been observed.

### Issues register

A set of notes about problems, complaints and concerns sent by all project members.

### Lessons log

A set of notes of lessons learned (often the hard way) which may be useful to future projects.

### Daily log

A diary about the project written by the project manager, like a [ship's log](http://en.wikipedia.org/wiki/Ship%27s_log).

## **Core principles**

Much of the methods above are derived from seven core principles:

### Continued business justification

The Business Case is the most important document, and is updated at every Stage of the project to ensure that the project is still viable. Early termination can occur if this ceases to be the case. At lower levels, use of time and resources should be justified, such as the need to have expensive meetings. (e.g. it is good practice to begin each meeting with a sentence about what its goal is—and if such a statement is hard to find then the meeting may not be necessary.)

### Manage by exception

Regular meetings, especially the dreaded "weekly team meetings" are considered inefficient and unnecessary. Instead, work packages are assigned by Task Managers to Team Members including deliverables with time and quality tolerances. If work progresses smoothly then the workers have no need to interfere with the Task Manager's time. Only if something deviated from the plan is communication and management required from them. Some Task Managers may request short status updates such as weekly emails or informal chats, to monitor for any exceptions, but no formal management is required unless an exception occurs.

### Learn from experience

Each project maintains a Lessons Log and projects should continually refer to their own and to previous and concurrent projects' Lessons Logs to avoid reinventing wheels.

### Defined roles and responsibilities

Roles are separated from individuals, who may take on multiple roles. By naming and defining roles in the PRINCE2 standard it becomes clear exactly who has what responsibility and decision making powers, avoiding arguments. Roles in PRINCE2 are structured in four levels (Corporate or programme management, Project Board, Project Manager level and Team level). The project manager level includes the project manager but can also include some optional roles like the quality assurance authority.

### Manage by stages

The project is planned and controlled on a stage by stage basis. This includes updating the Business Case, overall plan, and detailed next-stage plan after each stage in the light of new evidence.

### Focus on products

Each work package is defined by one or more deliverable products, preferably with tolerances to time and quality quantified in advance. Even management activities such as stage planning are defined by their final output, such as a physical report containing the new stage plan. This allows all parties to clearly specify what is required, and to allocate responsibility for delivering and controlling it.

### Tailoring

PRINCE2 should not be applied blindly in a dogmatic, bureaucratic form. (This would lead to wasted time on paperwork and create false senses of security). Rather it is defined to be a method in need to tailoring to specific projects. Typical adjustments include the replacement of deliverable reports and project documents by informal (verbal or email) equivalents, alterations to the structures of the project board and management team to reflect the goals of the project (e.g. replace majority board voting by sole Executive decisions in cases where the project's goal is to serve only the Executive's interest rather than users or suppliers); assignment of multiple roles to individuals on smaller projects; and increased emphasis on stage replanning for research-intensive projects which may need to change directions as new findings are delivered. A typical criticism of PRINCE2 is that the deliverable structure can lead to focus on producing deliverables for their own sake, to "tick the boxes" rather than do more useful work. If this is occurring, it demonstrates a failure of management to apply PRINCE2 and tailoring correctly.

## **Integration with other techniques**

PRINCE2 describes only high-level aspects of project management and leaves the choice of management tools and frameworks within its tasks to the task managers. It specifically mentions the following:

* [Product based planning](http://en.wikipedia.org/wiki/Product_based_planning)
* [Change control](http://en.wikipedia.org/wiki/Change_control)
* Quality review technique

and in passing mentions as possible tools,

* [Gantt charts](http://en.wikipedia.org/wiki/Gantt_chart)
* [PERT charts](http://en.wikipedia.org/wiki/PERT_chart)
* [critical path](http://en.wikipedia.org/wiki/Critical_path) analysis

A version of PRINCE2 is being developed to manage projects that use [Agile software development](http://en.wikipedia.org/wiki/Agile_software_development) techniques. [10](http://en.wikipedia.org/wiki/PRINCE2#cite_note-10)

### Quality review technique

*See also:*[*Quality assurance*](http://en.wikipedia.org/wiki/Quality_assurance)

The quality review technique ensures a project's products are of the required standard (i.e. meet defined quality criteria). This takes place in a quality review meeting, which identifies errors in the product. The quality review meeting will not attempt to solve the problems it identifies. The meeting brings together people who have an interest in the project's outputs (or products) and people on the project team able to address issues identified.

There are defined roles including a Producer and Scribe.

## **Certifications**

PRINCE2 certification requires passing the requisite examinations or assessment. [11](http://en.wikipedia.org/wiki/PRINCE2#cite_note-11)

### PRINCE2 Foundation

The lower level Foundation exam is a one-hour, multiple choice exam which tests candidate's knowledge of the method. The exam consists of 75 questions, 5 of which are trial questions which do not carry a mark. Of the remaining 70 questions which do carry a mark, the candidate needs to score 50% or more (i.e. 35 or more) to pass.

### PRINCE2 Practitioner

The higher level Practitioner exam lasts for 2.5 hours, and is a more complex multiple choice exam which tests candidate's ability to apply the method to a simple project scenario. The paper consists of 8 topics, with 10 questions per topic making a total of 80 marks. The pass mark is 55%, which is 44 marks or more. Passing the Foundation exam is a pre-requisite for sitting the Practitioner exam.

Candidates who have passed the PRINCE2 Practitioner exam may call themselves a Registered PRINCE2 Practitioner for 5 years after which they must pass a Re-registration examination every 5 years to maintain their Registered Practitioner status. The Re-registration exam is a one hour exam with 3 topics each containing 10 questions. The pass mark is 55%, which means candidates must score 17 marks or more to pass.

### PRINCE2 Professional

In 2012 the APM Group, accreditation body at the time, introduced a higher level qualification known as the PRINCE2 Professional qualification which is awarded following successful completion of a 2.5 day residential assessment involving group exercises and activities. The assessment criteria involve more general capabilities such as team working, which is not a specific PRINCE2 capability. Passing the Practitioner exam is a pre-requisite for sitting the Professional assessment.

### Examinations and training

Examinations can be sat by candidates who attend an accredited training course, or by those who purchase an accredited elearning course. Candidates who self-study may also purchase an exam via the APM Group's web site and can then sit the exam at a public exam centre, or at a [British Council](http://en.wikipedia.org/wiki/British_Council) office. [12](http://en.wikipedia.org/wiki/PRINCE2#cite_note-12)  The PRINCE2 Foundation exam may also be sat at home, with a live proctor observing the candidate virtually via webcam.

### Register

The APM Group publishes a successful candidate register which can be searched on the web. [13](http://en.wikipedia.org/wiki/PRINCE2#cite_note-13)  The register records the details of candidates who have sat PRINCE2 examinations.

### Trainers

Trainers must be re-accredited every years and undergo a surveillance check (either in the form of a visit by an assessor to a training course or a telephone interview which assesses their professional knowledge and training capability) every 12 months. [14](http://en.wikipedia.org/wiki/PRINCE2#cite_note-14)

### Exemptions

Qualified PRINCE2 Practitioners who go on to study for the APMP qualification of the [Association for Project Management](http://en.wikipedia.org/wiki/Association_for_Project_Management) are exempt from certain topics of the syllabus that are covered in the PRINCE2 Practitioner qualification. [15](http://en.wikipedia.org/wiki/PRINCE2#cite_note-15)

## **Scalability**

Project management is a complex discipline and it would be wrong to assume that blind application of PRINCE2 will result in a successful project. By the same token, it would be wrong to assume that every aspect of PRINCE2 will be applicable to every project. For this reason every process has a note on [scalability](http://en.wikipedia.org/wiki/Scalability). This provides guidance to the project manager (and others involved in the project) as to *how much* of the process to apply. The positive aspect of this is that PRINCE2 can be tailored to the needs of a particular project. The negative aspect is that many of the essential elements of PRINCE2 can be omitted sometimes resulting in a PINO project – Prince in Name Only. In order to counter this, APM Group have defined the concept of a PRINCE2 Maturity Model. [16](http://en.wikipedia.org/wiki/PRINCE2#cite_note-16)

## **Adoption**

PRINCE2, as a method and a certification, is adopted in many countries worldwide, including the UK, western European countries, and Australia. [17](http://en.wikipedia.org/wiki/PRINCE2#cite_note-17)

The [PMI](http://en.wikipedia.org/wiki/Project_Management_Institute) and its certification, the [PMP](http://en.wikipedia.org/wiki/Project_Management_Professional), is popular in the UK, USA and the rest of the world. [18](http://en.wikipedia.org/wiki/PRINCE2#cite_note-18)  Since the PMI's PMBOK (Project Management Body of Knowledge) is a collection of recognized good practices [19](http://en.wikipedia.org/wiki/PRINCE2#cite_note-19)  and PRINCE2 is a methodology, these two certifications are suggested to complement each other, as pointed out by the PRINCE2 official website.

## **Advantages and pitfalls**

PRINCE2 is a structured approach to project management. It provides a method for managing projects within a clearly defined framework. PRINCE2 describes procedures to coordinate people and activities in a project, how to design and supervise the project, and what to do if the project has to be adjusted if it doesn’t develop as planned. In the method each process is specified with its key inputs and outputs and with specific goals and activities to be carried out, which gives an automatic control of any deviations from the plan.

Divided into manageable stages, the method enables an efficient control of resources. On the basis of close monitoring the project can be carried out in a controlled and organised way. The various management roles and responsibilities involved in a project are fully described and are adaptable to suit the complexity of the project and skills of the organisation.

PRINCE2 is sometimes considered inappropriate for small projects or where requirements are expected to change, due to the work required in creating and maintaining documents, logs and lists. However, AXELOS claims that the methodology is scalable and can be tailored to suit the specific requirements and constraints of the project and the environment. [21](http://en.wikipedia.org/wiki/PRINCE2#cite_note-21)

# Rational Unified Process

The **Rational Unified Process** (**RUP**) is an [iterative](http://en.wikipedia.org/wiki/Iterative_and_incremental_development) [software development process](http://en.wikipedia.org/wiki/Software_development_process) framework created by the [Rational Software](http://en.wikipedia.org/wiki/Rational_Software) Corporation, a division of [IBM](http://en.wikipedia.org/wiki/IBM) since 2003. RUP is not a single concrete prescriptive process, but rather an adaptable process [framework](http://en.wikipedia.org/wiki/Software_framework), intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the [unified process](http://en.wikipedia.org/wiki/Unified_process).

## **History**

Rational Software originally developed the rational unified process as a software process product. The product includes a hyperlinked knowledge-base with sample [artifacts](http://en.wikipedia.org/wiki/Artifact_(software_development)) and detailed descriptions for many different types of activities. RUP is included in the [IBM Rational Method Composer](http://en.wikipedia.org/wiki/IBM_Rational_Method_Composer) (RMC) product which allows customization of the process.

[Philippe Kruchten](http://en.wikipedia.org/wiki/Philippe_Kruchten), an experienced Rational technical representative was tasked with heading up the original RUP team. This journey began with the creation of the Rational Objectory Process (ROP) in 1996, when Rational acquired the Objectory Process that had been written by [Ivar Jacobson](http://en.wikipedia.org/wiki/Ivar_Jacobson" \o "Ivar Jacobson) and company. This was renamed Rational Unified Process (RUP) in subsequent releases, in part to align the name with that of the Unified Modeling Language.

These initial versions combined the Rational Software organisations extensive field experience building object-oriented systems (referred to by Rational field staff as the Rational Approach) with Objectory's guidance on practices such as use cases, and incorporated extensive content from Jim Rumbaugh's [Object Modeling Technology](http://en.wikipedia.org/wiki/Object-modeling_technique) (OMT) approach to modeling, Grady Booch's[Booch method](http://en.wikipedia.org/wiki/Booch_method), and the newly released [UML](http://en.wikipedia.org/wiki/Unified_Modeling_Language) 0.8.   [3](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-rup-introduction-history-3)

To help make this growing knowledge base more accessible, [Philippe Kruchten](http://en.wikipedia.org/wiki/Philippe_Kruchten) was tasked with the assembly of an explicit process framework for modern software engineering. This effort employed the [HTML](http://en.wikipedia.org/wiki/HTML)-based process delivery mechanism developed by Objectory. The resulting "Rational Unified Process" (RUP) completed a strategic tripod for Rational:

* a *tailorable process* that guided development
* *tools* that automated the application of that process
* *services* that accelerated adoption of both the process and the tools.

This guidance was augmented in subsequent versions with knowledge based on the experience of companies that Rational had acquired.

In 1997, a requirements and test discipline were added to the approach, much of the additional material sourced from the Requirements College method developed by Dean Leffingwell et.al. at Requisite, Inc., and the SQA Process method developed at SQA Inc., both companies having been acquired by Rational Software.

In 1998 Rational Software added two new disciplines:

1. business modeling, much of this content had already been in the Objectory Process
2. a Configuration and Change Management discipline, sourced through the acquisition of Pure Atria Corporation.

These additions lead to an overarching set of principles that were defined by Rational and articulated within RUP as the six *best practices* for modern software engineering:

1. Develop iteratively, with risk as the primary iteration driver [4](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-rup-in-brief-4)
2. Manage requirements
3. Employ a component-based architecture
4. Model software visually
5. Continuously verify quality
6. Control changes

These best practices were tightly aligned with Rational's product line, and both drove the ongoing development of Rational's products, as well as being used by Rational's field teams to help customers improve the quality and predictability of their software development efforts.

Additional techniques including performance testing, UI Design, data engineering were included, and an update to reflect changes in UML 1.1.

In 1999, a project management discipline was introduced, as well as techniques to support real-time software development and updates to reflect UML 1.3

Between 2000 and 2003, a number of changes introduced guidance from ongoing Rational field experience with iterative development, in additional to tool support for enacting RUP instances and for customization of the RUP framework. These changes included:

1. the introduction of concepts and techniques from approaches such as eXtreme Programming (XP), that would later come to be known collectively as agile methods. This included techniques such as pair programming, test-first design, and papers that explained how RUP enabled XP to scale for use on larger projects.
2. a complete overhaul of the testing discipline to better reflect how testing work was conducted in different iterative development contexts.
3. the introduction of supporting guidance - known as "tool mentors" - for enacting the RUP practices in various tools. These essentially provided step-by-step method support to Rational tool users.
4. automating the customization of RUP in a way that would allow customers to select parts from the RUP process framework, customize their selection with their own additions, and still incorporate improvements in subsequent releases from Rational.

IBM acquired Rational Software in February 2003.

In 2006, IBM created a subset of RUP tailored for the delivery of [Agile](http://en.wikipedia.org/wiki/Agile_software_development) projects - released as an OpenSource method called [OpenUP](http://en.wikipedia.org/wiki/OpenUP" \o "OpenUP) through the [Eclipse](http://en.wikipedia.org/wiki/Eclipse_(software)) web-site. [5](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-5)

## **Rational unified process topics**

### RUP building blocks

RUP is based on a set of building blocks and content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved. The main building blocks, or content elements, are the following:

* Roles (who) – A role defines a set of related skills, competencies and responsibilities.
* Work products (what) – A work product represents something resulting from a task, including all the documents and models produced while working through the process.
* Tasks (how) – A task describes a unit of work assigned to a Role that provides a meaningful result.

Within each iteration, the tasks are categorized into nine disciplines:

* Six "engineering disciplines"
  + Business modelling
  + Requirements
  + Analysis and design
  + Implementation
  + Test
  + Deployment
* Three supporting disciplines
  + Configuration and change management
  + Project management
  + [Environment](http://en.wikipedia.org/wiki/Environment_discipline)

### Four project life-cycle phases

The RUP has determined a project life-cycle consisting of four phases. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although in essence the key to the process lies in the iterations of development that lie within all of the phases. Also, each phase has one key objective and milestone at the end that denotes the objective being accomplished. The visualization of RUP phases and disciplines over time is referred to as the [RUP hump](http://en.wikipedia.org/wiki/RUP_hump) chart.

#### Inception phase

The primary objective is to scope the system adequately as a basis for validating initial costing and budgets. In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc.), and financial forecast is established. To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated. After these are completed, the project is checked against the following criteria:

* [Stakeholder](http://en.wikipedia.org/wiki/Stakeholder_(corporate)#In_management) concurrence on scope definition and cost/schedule estimates.
* Requirements understanding as evidenced by the fidelity of the primary use cases.
* Credibility of the cost/schedule estimates, priorities, risks, and development process.
* Depth and breadth of any architectural prototype that was developed.
* Establishing a baseline by which to compare actual expenditures versus planned expenditures.

If the project does not pass this milestone, called the lifecycle objective milestone, it either can be cancelled or repeated after being redesigned to better meet the criteria.

#### Elaboration phase

The primary objective is to mitigate the key risk items identified by analysis up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

The outcome of the elaboration phase is:

* A use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.
* A description of the software architecture in a software system development process.
* An [executable architecture](http://en.wikipedia.org/wiki/Executable_Architecture) that realizes architecturally significant use cases.
* Business case and risk list which are revised.
* A development plan for the overall project.
* Prototypes that demonstrably mitigate each identified technical risk.
* A preliminary user manual (optional)

This phase must pass the lifecycle architecture milestone criteria answering the following questions:

* Is the vision of the product stable?
* Is the architecture stable?
* Does the executable demonstration indicate that major risk elements are addressed and resolved?
* Is the construction phase plan sufficiently detailed and accurate?
* Do all stakeholders agree that the current vision can be achieved using current plan in the context of the current architecture?
* Is the actual vs. planned resource expenditure acceptable?

If the project cannot pass this milestone, there is still time for it to be cancelled or redesigned. However, after leaving this phase, the project transitions into a high-risk operation where changes are much more difficult and detrimental when made.

The key domain analysis for the elaboration is the system architecture.

#### Construction phase

The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.

This phase produces the first external release of the software. Its conclusion is marked by the initial operational capability milestone.

#### Transition phase

The primary objective is to 'transit' the system from development into production, making it available to and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase.

If all objectives are met, the product release milestone is reached and the development cycle is finished.

### The IBM Rational Method Composer product

The IBM Rational Method Composer product is a tool for authoring, configuring, viewing, and publishing processes. See [IBM Rational Method Composer](http://en.wikipedia.org/wiki/IBM_Rational_Method_Composer) and an open source version [eclipse process framework](http://en.wikipedia.org/w/index.php?title=Eclipse_process_framework&action=edit&redlink=1) (EPF) project for more details.

### Certification

In January 2007 the new RUP certification examination for *IBM Certified Solution Designer - Rational Unified Process 7.0* was released which replaces the previous version of the course called*IBM Rational Certified Specialist - Rational Unified Process*. [6](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-rupcert-6)  The new examination will not only test knowledge related to the RUP content but also to the process structure elements. [7](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-rup7-7)

To pass the new RUP certification examination, a person must take IBM's *Test 839: Rational Unified Process v7.0*. You are given 75 minutes to take the 52 question exam. The passing score is 62%. [8](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-test389-8)

### Six best practices

Six best practices as described in the rational unified process is a paradigm in [software engineering](http://en.wikipedia.org/wiki/Software_engineering) that lists six ideas to follow when designing any software project to minimize faults and increase productivity. These practices are: [9](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-9)  [10](http://en.wikipedia.org/wiki/Rational_Unified_Process#cite_note-10)

**Develop iteratively**

It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solution on how to minimize cost in terms of development phases.

**Manage requirements**

Always keep in mind the requirements set by users.

**Use components**

Breaking down an advanced project is not only suggested but in fact unavoidable. This promotes ability to test individual components before they are integrated into a larger system. Also, code reuse is a big plus and can be accomplished more easily through the use of [object-oriented programming](http://en.wikipedia.org/wiki/Object-oriented_programming).

**Model visually**

Use diagrams to represent all major components, users, and their interaction. "UML", short for [Unified Modeling Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language), is one tool that can be used to make this task more feasible.

**Verify quality**

Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

**Control changes**

Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result it is essential to make sure that changes made to a system are synchronized and verified constantly. (See [Continuous integration](http://en.wikipedia.org/wiki/Continuous_integration)).

Nguồn Tham khảo: <http://www.wn.wikipedia.org/>

[www.en.wikipedia.org](http://www.en.wikipedia.org/)