敏捷软体开发（英语：Agile software development），又称敏捷开发，是一种应对快速变化需求的软体开发模式，描述了一套软体开发的价值和原则。此模式中，自组织的跨功能团队在紧密的协作中发掘使用者或顾客的需求以及改良解决方案，[1]此模式也强调适度的计划、进化开发、提前交付与持续改进，并且鼓励快速与灵活的面对开发与变更。[2][3]这些原则支援许多软件开发方法的定义和持续进化。

「敏捷」（Agile 或 agile[4]）一词由「敏捷软件开发宣言」（Manifesto for agile software development）[5]中开始普及，「敏捷软件开发宣言」定义了相关的价值和原则。敏捷软体开发的框架不断的发展，两个最广泛被使用的是 Scrum 与 Kanban。[6]

敏捷一词来源于2001年初美国犹他州雪鸟滑雪圣地的一次敏捷方法发起者和实践者（他们发起组成了敏捷联盟）的聚会。

迭代和增量式软件开发方法可以追溯到1957年。进化式专案管理和适应性软体开发出现在1970年代初期。在1990年代，因针对重量级的软体开发方法的批评，而发展了许多轻量化的软体开发方法、计划与细微化开发管理。包含了，从1991年开始的迅速应用程式开发、从1994年开始的统一处理程序与动态系统开发法（DSDM）、从1995年的Scrum、从1996年的水晶清透法与极限编程法、1997年的功能驱动开发。虽然那些开发法都起源于敏捷开发宣言之前，但都统称为敏捷软体开发法。在此同时，制造业与航空业也遭遇相同变化。

在2001年，十七名软体开发人员在犹他州的雪鸟度假村会面，讨论这些轻量级的开发方法，并由Jeff Sutherland，Ken Schwaber和Alistair Cockburn发起，一同发布了「敏捷软体开发宣言」。

在2005年，由Alistair Cockburn和Jim Highsmith领导的小组编写了一份项目管理原则的附录，即“相互依存声明”，以便根据敏捷软体开发方法来指导软体专案管理。

在2009年，罗伯特·C·马丁（英语：Robert C. Martin）（Robert C Martin）编写了软体开发原则的扩展，即软体工艺宣言（Software Craftsmanship Manifesto），根据职业行为和掌握程度来指导敏捷软体开发。

在2011年，敏捷联盟创建了敏捷实践指南（2016年更名为“敏捷词汇”）、敏捷实践、术语和元素工作定义的演化开放式汇编，以及来自敏捷从业者社群的经验指南。

雪鸟会议共同起草了敏捷软件开发宣言。其中最重要的部分就是对一些与会者一致同意的软件开发价值观的表述。[7][8]其中包括了以下方针：

虽然他们也很重视右边的内容，但是更重视左边的内容。相关术语的意思是:

一些软体开发者组织了敏捷联盟，为非营利组织，根据宣言的价值观和原则促进软体开发。吉姆·海史密斯（英语：Jim Highsmith）（Jim Highsmith）代表敏捷联盟（Agile Alliance）介绍宣言：

大多数敏捷开发方法将产品开发工作细分微小化，因此大大的减少了前期规划和设计的数量。迭代或冲刺都是短时间的框架（时间），通常持续一到四周。每个迭代都具有跨功能、职能的团队，包含了规划、分析、设计、程式编码、单元测试和验收测试。在迭代结束时，将工作产品向利益相关者展示。透过上述方式让整体风险降至最低，并使产品能够快速适应变化。迭代的方式，可能不会一次增加足够的功能来保证可立即发布使用，但是目标是在每次迭代结束时，有一个可用的发行版（最小化程式缺点）。因此完整产品的发布或新功能可能需要多次迭代。

无论采用哪种开发方式，每个团队都应该包含一个客户代表（Scrum中的产品负责人）。这个人是由利益相关者同意代表他们行事，并作出个人承诺，回应开发人员在开发迭代过程中的问题。在每次迭代结束时，利益相关方和客户代表将审查进度并重新评估优先级，以优化投资回报（ROI）并确保与客户需求和公司目标保持一致。在敏捷软体开发中，会在开发团队附近设置一个讯息发布器（通常很大）实体显示器，甚至路人也可以看到它。它提供了最新的产品开发状态摘要。并透过建置状态指示灯以通知团队他们的产品开发的当前状态。

敏捷软件开发中的一个共同特点就是每日站会（也在scrum中被称为日常scrum）。 在一个简短的会议中，团队成员相互报告他们前一天对于团队的迭代目标、今天打算做的目标以及他们可以看到的任何障碍或阻碍。

经常使用诸如持续整合、自动化单元测试、配对程式开发、测试驱动开发、设计模式、领域驱动设计，程式码重构和其他技术的特定工具和技术来提高品质和提高产品开发敏捷性

与传统软体工程相比，敏捷软体开发主要针对具有动态、非确定性和非线性特征的复杂系统和产品进行开发。准确的估计、稳定的计划和预测往往很难在早期达到，因此对它们的信心可能很低，在获得价值的证据之前，敏捷开发从业人员需要相当的的信心。 需求和设计被认为是紧急情况下，过大的前期规格可能会造成很多浪费，在经济上也不划算。 由行业从多年经验的成功和失败中学到的这些基本论点。

从适应到预测的软体开发法存在于连续体中，敏捷软体开发法倚靠连续体的适应性面。适应性开发法的一个关键是透过「滚动波」法进行计划。其中确定了里程碑，但留下了灵活性，以达到他们的路径，也允许里程碑本身的改变。

适应性方法的重点是快速适应不断变化的现实。当专案需求发生变化时，适应性团队也会发生变化。 自适应团队很难描述未来会发生什么。 离专案目标日期越远，适应性方法越是模糊，越无法确知那天会发生什么变化。一个自适应的团队无法准确地报告他们下周将要完成的任务，而只是他们下个月计划的功能。当被问及六个月后的发布时，一个自适应团队可能只能报告发布的使命声明，或预期价值与成本的声明。

相较之下，预测法著重于详细分析和规划未来，并迎合已知的风险。在极端情况下，预测团队可以准确报告在整个开发过程中计划的功能和任务。预测法依靠有效的早期阶段分析，如果这样做很不妥，专案可能难以改变方向。预测团队通常设立一个变更控制委员会，以确保他们只考虑最有价值的变化。

风险分析可以于自适应（敏捷或价值驱动）和预测（计划驱动）法之间进行选择。巴里·伯姆（英语：Barry Boehm）（Barry Boehm）和理查德·特纳（Richard Turner）认为，连续统一体的每一面都有自己的主场，如下表所示:

敏捷软体开发法和瀑布法之间的区别，其中之一就是质量和测试方法。在瀑布模型中，构建阶段之后总是有单独的测试阶段; 但是，在敏捷软件开发测试中，与编程相同的叠代完成。

由于测试是在每一次叠代中完成的-开发一小部分软件，用户可以经常使用这些新的软体并验证其价值。用户知道更新后的软体的真实价值后，可以对软体的未来作出更好的决策。在每次叠代中进行一次价值回顾和软体重新计划会话 - Scrum通常只有两个星期的叠代循环 - 帮助团队不断调整自己的计划，以最大限度地提高其价值。 遵循与PDCA循环类似的模式，因为工作已经计划、完成、检查（在审查和回顾中），并且任何商定的变更都会被执行。

这种叠方法支援产品更甚于专案思维。这在整个开发过程中提供更大的灵活性; 而在专案中，需求是从一开始就定义和锁定的，以后很难改变它们。叠代开发允许软体产品根据业务环境或市场需求的变化而发展。由于敏捷软件开发的叠代方式较短，因此与精益创业概念有著密切的联系。

在给IEEE计算机的一封信中，Steven Rakitin表达了对敏捷软件开发的愤世嫉俗，称敏捷开发为 "一个破坏软件工程规范的尝试" ，并将 "将软件工作在综合性文档之上" 翻译为 "我们要花时间编码。请记住，真正的程序员不写文档 。"

敏捷软体开发的支持者认为，开发人员应该编写文档，如果这是实现相关目标的最佳途径，但是与编写静态文档相比，往往有更好的方法来实现这些目标。斯科特·安布勒（Scott Ambler）指出，文档应该做到「够好」即可，过多或全面的文档通常会造成浪费，开发人员很少相信详细的文档，因为它通常与程式码不同步，而文档太少 也可能导致维护、沟通、学习和知识共享的问题。Alistair Cockburn写了透明清晰法：

宣言中还包括以下原则：[9]  
[10]

敏捷软体开发法支援广泛的软体开发生命周期。有的专注于实践（例如，极限编程、务实编程，敏捷建模），而有的专注于管理工作流程（例如Scrum，看板）。有的支援需求规范和开发（例如FDD）的活动，而另一些则试图涵盖整个开发生命周期（例如DSDM，RUP）。

流行的敏捷软件开发框架包括（以下列举常见的方法）：

敏捷软体开发得到了许多具体实践的支持，涵盖了需求、设计、建模、编码、测试、计划、风险管理、流程、品质等方面。一些值得注意的敏捷软体开发实践包括：

敏捷联盟提供了一个全面的线上指南来应用敏捷与相关做法。

在文献中，有不同的术语指适应法的概念，包括“剪裁法”、“片段适应法”和“情景方法工程”。 方法剪裁被定义为：一个程序或能力，在这个程序或能力中，人类代理程式通过在情境、意图和方法片段之间的响应变化和动态的相互作用来确定特定项目情境的系统开发方法。

几乎所有的敏捷方法都适用于剪裁法。即使是DSDM方法也被用于此目的，并且已经成功地在CMM环境中进行了定制。敏捷法和传统的软体开发法之间的情境适应性，可以被认为是一个明显的特征，后者因规范而相对更加僵化。敏捷法实际的含义是可以使产品开发团队根据个别产品的需求来调整工作实践。实践是作为方法框架一部分的具体活动和产品。在更为极端的层面上，可以调整由多种原则组成的方法背后的哲学（Aydin，2004）。

某些方法，如Scrum和极限编程，使得方法适应的需求是明确的。有了这些不太规范的框架，原则之一就是没有一个程序适合每一个产品的开发，而是应该根据产品的需求量身定做。 Mehdi Mirakhorli提出了一个剪裁实践，为适应所有实践提供了足够的路线图和指导方针。RDP 的实践是为极限编程而设计的。这种做法首先作为ICSE 2008会议APSO研讨会上的一篇长篇研究论文提出，是目前唯一提出并且适用于定制极限编程的方法。虽然它是专门针对极限编程的解决方案，但是这种做法有扩展到其他方法的能力。乍看之下，这种做法似乎属于静态方法适应的范畴，但RDP实践的经验表明，它可以像动态方法适应一样对待。静态方法适应和动态方法适应之间的区别是微妙的。

Scrum不是为剪裁法而设计的。 Schwaber指出：“Scrum不是一种需要加强的方法，这是我们首先遇到的麻烦，认为问题没有一个完美的方法，努力集中在企业所需的变化上。 Bas Vodde强调了这一说法，表明Scrum不像传统的大型方法论，需要你“挑选”元素。 这是基础知识的基础上，您添加额外的元素来本地化和使用情况。

敏捷软体开发被广泛认为非常适合于某些类型的环境，包括从事绿地项目的小型专家团队，在拥有传统基础架构的大型组织中采用敏捷软件开发方法所遇到的挑战和局限性， 记录和理解。

作为回应，一系列的策略和模式已经发展成为克服大规模开发工作（> 20个开发人员）或分布式（非集中式）开发团队等挑战; 现在有几个公认的框架，试图减轻或避免这些挑战。

对于所有这些是否有效或者确实符合敏捷开发的定义，存在许多相互矛盾的观点，而且这仍然是一个积极和正在进行的研究领域。

当敏捷软体开发应用于分布式环境（团队分散在多个业务地点）时，通常称为分布式敏捷开发。目标是利用每种方法提供的独特优势。分布式开发允许组织通过战略性地在全球不同地区建立团队来构建软件，实际上是全天候建立软件（或称为“跟随太阳模式”）。另一方面，敏捷开发在响应变化时提供了更高的透明度，持续的反馈和更大的灵活性。

敏捷方法有时候被误认为是无计划性和纪律性的方法，实际上更确切的说法是敏捷方法强调适应性而非预见性。

适应性的方法集中在快速适应现实的变化。当项目的需求起了变化，团队应该迅速适应。这个团队可能很难确切描述未来将会如何变化.

相比迭代式开发两者都强调在较短的开发周期提交软件，敏捷方法的周期可能更短，并且更加强调队伍中的高度协作。

两者没有很多的共同点，瀑布模型是最典型的预见性的方法，严格遵循预先计划的需求、分析、设计、编码、测试的步骤顺序进行。步骤成果作为衡量进度的方法，例如需求规格，设计文档，测试计划和代码审阅等等。

瀑布式的主要的问题是它的严格分级导致的自由度降低，项目早期即作出承诺导致对后期需求的变化难以调整，代价高昂。瀑布式方法在需求不明并且在项目进行过程中可能变化的情况下基本是不可行的。

相对来讲，敏捷方法则在几周或者几个月的时间内完成相对较小的功能，强调的是能将尽早将尽量小的可用的功能交付使用，并在整个项目周期中持续改善和增强。

有人可能在这样小规模的范围内的每次迭代中使用瀑布式方法，另外的人可能将选择各种工作并行进行，例如极限编程。

在敏捷方法其独特之处以外，他和其他的方法也有很多共同之处，比如迭代开发，关注互动沟通，减少中介过程的无谓资源消耗。通常可以在以下方面衡量敏捷方法的适用性：从产品角度看，敏捷方法适用于需求萌动并且快速改变的情况，如系统有比较高的关键性、可靠性、安全性方面的要求，则可能不完全适合；从组织结构的角度看，组织结构的文化、人员、沟通则决定了敏捷方法是否适用。跟这些相关联的关键成功因素有：

最重要的因素恐怕是项目的规模。规模增长，面对面的沟通就愈加困难，因此敏捷方法更适用于较小的队伍，40、30、20、10人或者更少。大规模的敏捷软件开发尚处于积极研究的领域。

另外的问题是项目初期的大量假定或者快速收集需求可能导致项目走入误区，特别是客户对其自身需要毫无概念的情况下。与之类似，人之天性很容易造成某个人成为主导并将项目目标和设计引入错误方向的境况。开发者经常能把不恰当的方案授予客户，并且直到最后发现问题前都能获得客户认同。虽然理论上快速交互的过程可以限制这些错误的发生，但前提是要有效的“负反馈”，否则错误会迅速膨胀，并在最终提交时造成极大返工。

已经有一些项目管理工具用于敏捷开发，可以用它们来帮助规划，跟踪，分析和整合工作。  
这些工具在敏捷开发中扮演的重要的角色，也是知识管理的一种方法。

通常包括：版本控制整合，进度跟踪，工作分配，集成发布和迭代规划，论坛和软件缺陷的报告和跟踪。

目前列入敏捷方法的有：