2019 级本科

软件工程

Software Engineering

张昕

zhangxin@cust.edu.cn

计算机科学技术学院软件工程系

Agile Development

Concept

What is agile software engineering

- Agile software engineering combines a philosophy and a set of development guidelines.
- The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity.
- The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.

Importance

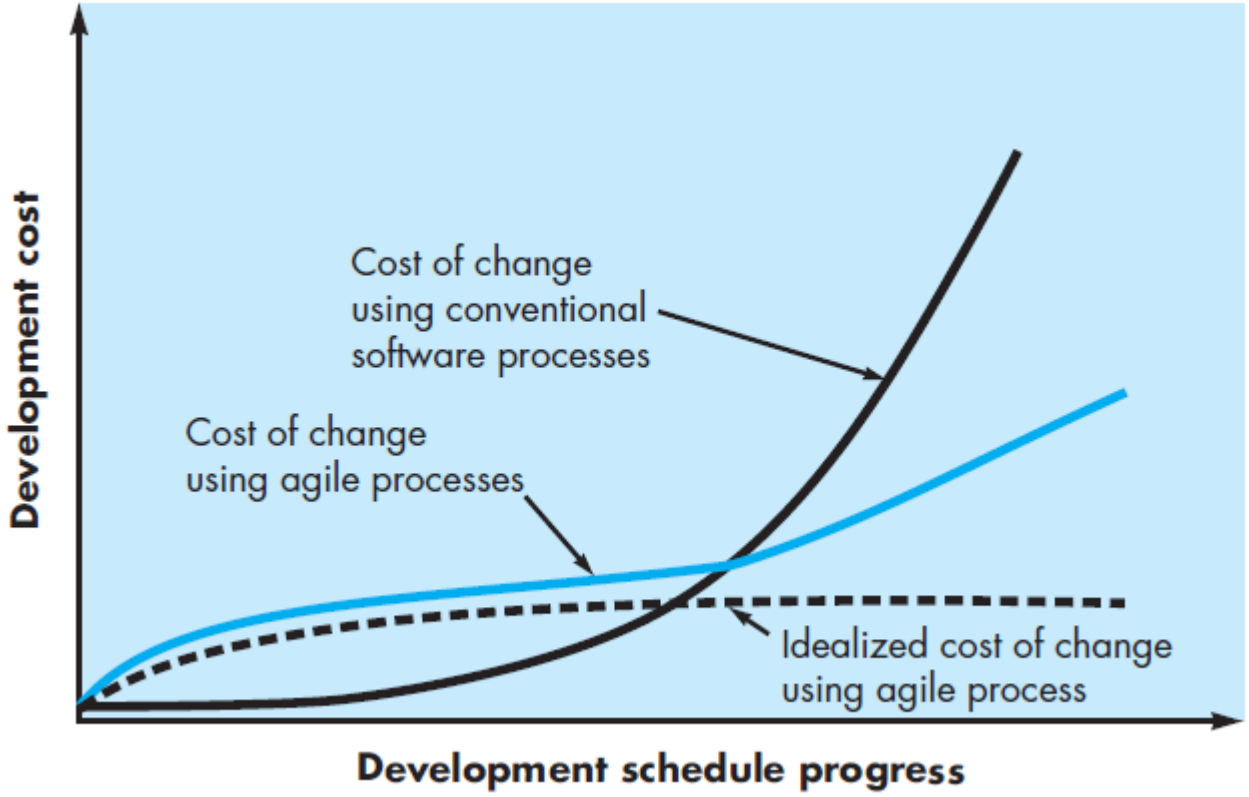
- The modern business environment that spawns computer-based systems and software products is fast-paced and ever-changing.
- Agile software engineering represents a reasonable alternative to conventional software engineering for certain classes of software and certain types of software projects.
- It has been demonstrated to deliver successful systems quickly.

Agile

- Agility has become today's buzzword when describing a modern software process.
  - An agile team is a nimble team able to appropriately respond to changes.
  - An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.
- Agility is more than an effective response to change.
  - It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineers and their managers) more facile.
  - It emphasizes rapid delivery of operational software and deemphasizes the importance of intermediate work products (not always a good thing); it adopts the customer as a part of the development team and works to eliminate the "us and them" attitude that continues to pervade many software projects; it recognizes that planning in an uncertain world has its limits and that a project plan must be fl exible.
- Agility can be applied to any software process.
  - It is essential that the process be designed in a way that allows the project team to adapt tasks and to streamline them, conduct planning in a way that understands the fluidity of an agile development approach, eliminate all but the most essential work products and keep them lean, and emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

Agility and the cost of change

- The conventional wisdom in software development (supported by decades of experience) is that the cost of change increases nonlinearly as a project progresses.
- It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project).
- Proponents of agility argue that a well-designed agile process "fl attens" the cost of change curve, allowing a software team to accommodate changes late in a software project without dramatic cost and time impact.
- When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.



Agile process

Characteristics

- An agile process must be adaptable, i.e., be of process adaptability (to rapidly changing project and technical conditions).
  - But continual adaptation without forward progress accomplishes little.
- An agile software process must adapt incrementally.
  - To accomplish incremental adaptation, an agile team requires customer feedback (so that the appropriate adaptations can be made).
- An effective catalyst for customer feedback is an operational prototype or a portion of an operational system.
- An incremental development strategy should be instituted.
  - Software increments (executable prototypes or portions of an operational system) must be delivered in short time periods so that adaptation keeps pace with change (unpredictability).
  - This iterative approach enables the customer to evaluate the software increment regularly, provide necessary feedback to the software team, and influence the process adaptations that are made to accommodate the feedback.

Agility principles

- The Agile Alliance defines 12 agility principles for those who want to achieve agility

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agility principles

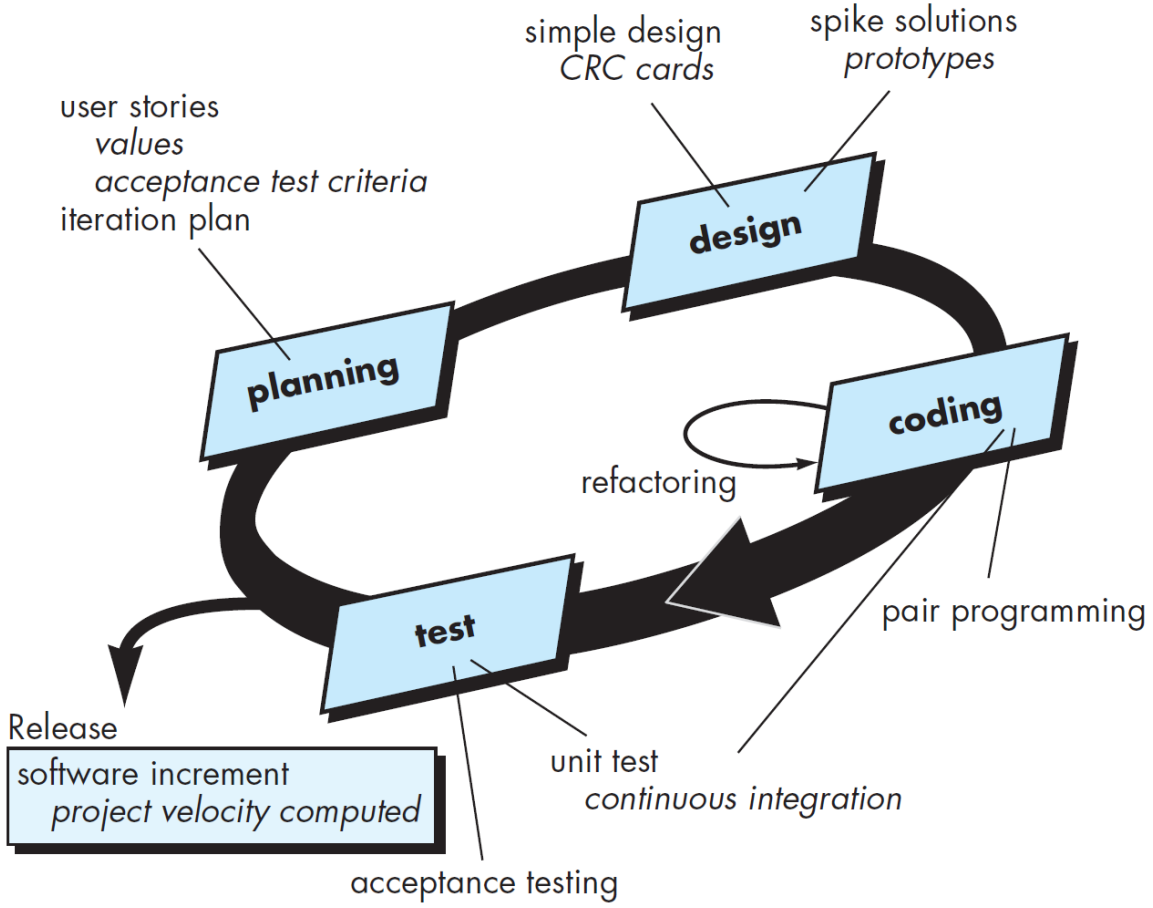
- The Agile Alliance defines 12 agility principles for those who want to achieve agility

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from selforganizing teams.
12. At regular intervals, the team refl ects on how to become more effective, then tunes and adjusts its behavior accordingly.

Not every agile process model applies these 12 principles with equal weight, and some models choose to ignore (or at least downplay) the importance of one or more of the principles.

Extreme Programming (XP)

- The most widely used approach to agile software development.
  - Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities:
    - planning,
    - design,
    - coding, and
    - testing.



Extreme Programming (XP)

Planning

- The planning activity begins with listening — a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to get a broad feel for required output and major features and functionality.
  - Listening leads to the creation of a set of "stories " (also called *user stories*) that describe required output, features, and functionality for software to be built.
  - Each *story* is written by the customer and is placed on an index card. The customer assigns a *value* (i.e., a priority) to the story based on the overall business value of the feature or function.
  - Members of the XP team then assess each story and assign a *cost* — measured in development weeks —to it. If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again. It is important to note that new stories can be written at any time.

Extreme Programming (XP)

Planning

- Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team.
  - Once a basic commitment (agreement on stories to be included, delivery date, and other project matters) is made for a release, the XP team orders the stories that will be developed in one of three ways:
    - (1) all stories will be implemented immediately (within a few weeks),
    - (2) the stories with highest value will be moved up in the schedule and implemented first, or
    - (3) the riskiest stories will be moved up in the schedule and implemented first.
  - After the first project release (also called a software increment) has been delivered, the XP team computes project velocity. Stated simply, project velocity is the number of customer stories implemented during the first release. Project velocity can then be used to
    - (1) help estimate delivery dates and schedule for subsequent releases and
    - (2) determine whether an overcommitment has been made for all stories across the entire development project. If an overcommitment occurs, the content of releases is modified or end delivery dates are changed.

Extreme Programming (XP)

- Planning
  - As development work proceeds, the customer can add stories, change the value of an existing story, split stories, or eliminate them. The XP team then reconsiders all remaining releases and modifies its plans accordingly.

Extreme Programming (XP)

Design

- XP design rigorously follows the KIS (keep it simple) principle. A simple design is always preferred over a more complex representation. In addition, the design provides implementation guidance for a story as it is written—nothing less, nothing more. The design of extra functionality (because the developer assumes it will be required later) is discouraged.
- XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context.
  - CRC (class-responsibility-collaborator) cards identify and organize the objectoriented classes that are relevant to the current software increment.
  - The CRC cards are the only design work product produced as part of the XP process.
- XP encourages refactoring —a construction technique that is also a design technique.
  - Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure. It is a disciplined way to clean up code [and modify/simplify the internal design] that minimizes the chances of introducing bugs. In essence, when you refactor you are improving the design of the code after it has been written.
  - The intent of refactoring is to control these modifications by suggesting small design changes that "can radically improve the design". Refactoring means that design occurs continuously as the system is constructed.

Extreme Programming (XP)

Coding

- After stories are developed and preliminary design work is done, the team does not move to code, but rather develops a series of unit tests that will exercise each of the stories that is to be included in the current release (software increment).
- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test. Once the code is complete, it can be unit-tested immediately, thereby providing instantaneous feedback to the developers.
- A key concept during the coding activity (and one of the most talked-about aspects of XP) is pair programming.
    - XP recommends that two people work together at one computer workstation to create code for a story. This provides a mechanism for real-time problem solving (two heads are often better than one) and real-time quality assurance (the code is reviewed as it is created). It also keeps the developers focused on the problem at hand.

Extreme Programming (XP)

Testing

- The unit tests that are created should be implemented using a framework that enables them to be automated (hence, they can be executed easily and repeatedly). This encourages a regression testing strategy whenever code is modified (which is often, given the XP refactoring philosophy).
- As the individual unit tests are organized into a "universal testing suite", integration and validation testing of the system can occur on a daily basis.
    - This provides the XP team with a continual indication of progress and also can raise warning fl ags early if things go awry. Wells states: "Fixing small problems every few hours takes less time than fixing huge problems just before the deadline."
- XP acceptance tests , also called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer. Acceptance tests are derived from user stories that have been implemented as part of a software release.

Dynamic Systems Development Method (DSDM)

- The Dynamic Systems Development Method (DSDM) is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".
  - The DSDM philosophy is borrowed from a modified version of the Pareto principle—80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.
  - DSDM is an iterative software process in which each iteration follows the 80 percent rule. That is, only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

Dynamic Systems Development Method (DSDM)

- The consortium has defined an agile process model, called the DSDM life cycle, that begins with a feasibility study that establishes basic business requirements and constraints and is followed by a business study that identifies functional and information requirements. DSDM then defines three different iterative cycles:
  - Functional model iteration— produces a set of incremental prototypes that demonstrate functionality for the customer. (Note: All DSDM prototypes are intended to evolve into the deliverable application.) The intent during this iterative cycle is to gather additional requirements by eliciting feedback from users as they exercise the prototype.
  - Design and build iteration— revisits prototypes built during the functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users. In some cases, the functional model iteration and the design and build iteration occur concurrently.
  - Implementation— places the latest software increment (an "operationalized" prototype) into the operational environment. It should be noted that (1) the increment may not be 100 percent complete or (2) changes may be requested as the increment is put into place. In either case, DSDM development work continues by returning to the functional model iteration activity.

Agile Unified Process

- The Agile Unified Process (AUP) adopts a "serial in the large" and "iterative in the small" philosophy for building computer-based systems.
  - By adopting the classic UP phased activities—inception, elaboration, construction, and transition—AUP provides a serial overlay (i.e., a linear sequence of software engineering activities) that enables a team to visualize the overall process flow for a software project.
    - However, within each of the activities, the team iterates to achieve agility and to deliver meaningful software increments to end users as rapidly as possible.

Agile Unified Process

- Each AUP iteration addresses the following activities
  - Modeling. UML representations of the business and problem domains are created. However, to stay agile, these models should be "just barely good enough" to allow the team to proceed.
  - Implementation. Models are translated into source code.
  - Testing. Like XP, the team designs and executes a series of tests to uncover errors and ensure that the source code meets its requirements.
  - Deployment. Like the generic process activity discussed in Chapters 3, deployment in this context focuses on the delivery of a software increment and the acquisition of feedback from end users.
  - Configuration and project management. In the context of AUP, configuration management (Chapter 29) addresses change management, risk management, and the control of any persistent work products 12 that are produced by the team. Project management tracks and controls the progress of the team and coordinates team activities.
  - Environment management. Environmental management coordinates a process infrastructure that includes standards, tools, and other support technology available to the team.