Introduction to Agile methodologies

Principles and techniques

Agile approach

Limits of the traditional models

- The waterfall model has been wide exploited
  - Until '90
  - For large projects
- Anyway, it has some limits that have **not** been solved by first iterative models
  - **Rigid** order of phases
  - Results are at the **end** of the project
  - A lot of **paper** at the beginning
  - **Not** always the result satisfies the customer
  - All requirements are needed at the **beginning**, but
    - 20-50% of the requirements **change** during the project
    - 45-65% of the requirements are **not used**

CHAOS report 2011-2015

Project size	Method	Successful	Challenged	Failed
All sizes	Waterfall	11%	60%	29%
Large	Waterfall	3%	55%	42%
Medium	Waterfall	7%	68%	25%
Small	Waterfall	44%	45%	11%

Birth of the agile model (1/2)

- In 2001, 17 experts of software engineering met to overcome the limits of the existing development models
  - At The Lodge at Snowbird ski resort in the Wasatch mountains of Utah
- To avoid projects' failures
  - Unsuccessful results of waterfall
  - Indefinite unsuccessful iterations
- The result was the Agile 'Software Development' Manifesto
- http://agilemanifesto.org/

Born of the agile model (2/2)

- The idea emerged from the meeting was that the engineering method **could not work** applied to software development, because:
  - Software development is a **creative** activity, not a productive activity
  - People involved in the development are **knowledge** workers and not bare workers
  - The "**craft**" aspect is dominant
  - **Human interaction** is very relevant

Agile manifesto (1/2)

Agile manifesto (2/2)

- Note that the agile community does not discard the "over" items, but consider the former items more important
- This leads to a **new way** to develop software
  - Possibly **better** than traditional approaches

The 12 principles (1/3)

- Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
- Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work **together** daily throughout the project.

The 12 principles (2/3)

- Build projects around **motivated** individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation.
- **Working** software is the primary measure of progress.
- Agile processes promote **sustainable** development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

The 12 principles (3/3)

- Continuous attention to **technical excellence** and **good design** enhances agility.
- **Simplicity**-the art of maximizing the amount of work not done-is essential.
- The best architectures, requirements, and designs emerge from **self-organizing** teams.
- At regular intervals, the team **reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methods and methodologies

- Based on these principles, several methods and methodologies have been proposed
- Adaptive Software Development (ASD), Agile Modeling, Agile Unified Process (AUP), Crystal Methods (Crystal Clear), Disciplined Agile Delivery, Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Lean software development, Kanban, Scrum, Scrum-ban, ….

Agile techniques

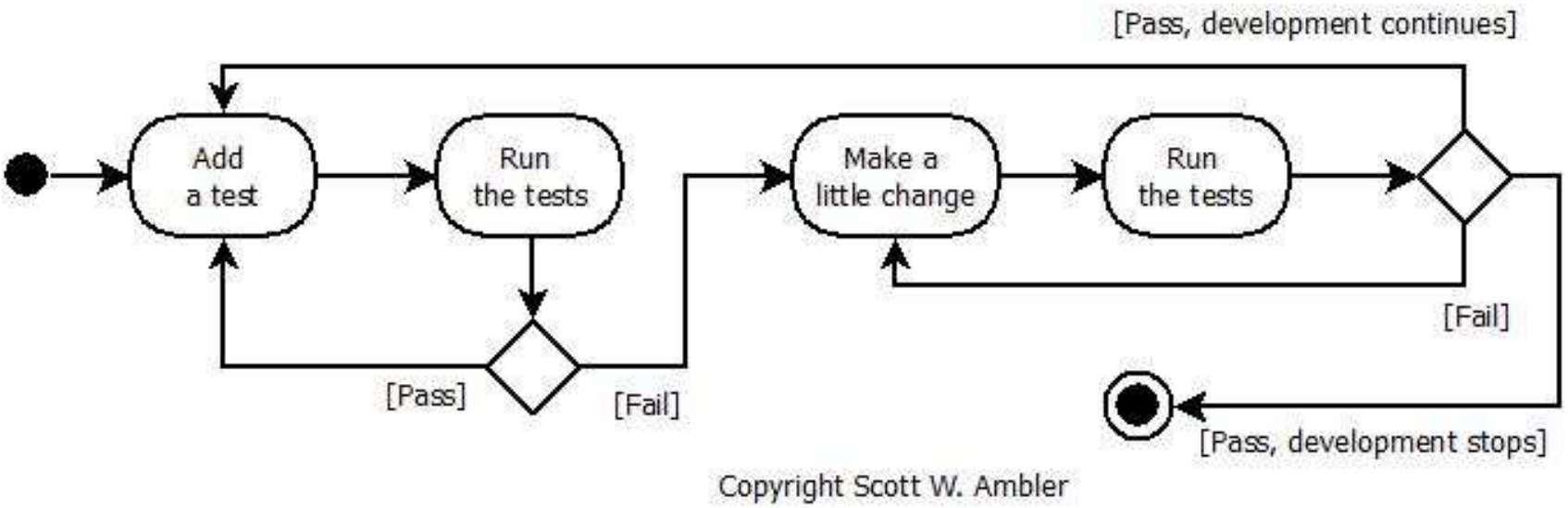
The agile techniques

- Independently of the specific method, all the agile methodologies apply these **techniques**:
  - Test Driven Development
  - Pair Programming
  - Refactoring
  - Cross functional team
  - Timeboxing
  - User stories

Test Driven Development (1/4)

- The test is written **before** the code
  - The first time the test **fails**, of course
    - There is no code, the test works correctly
- This approach is intended to **stimulate the programmer** to think about conditions in which her code could fail
- It is important to have **tools** for automatic tests (e.g., Junit) and/or execution simulators
- Also called Test-first development/programming (TFD, TFP)

Test Driven Development (2/4)

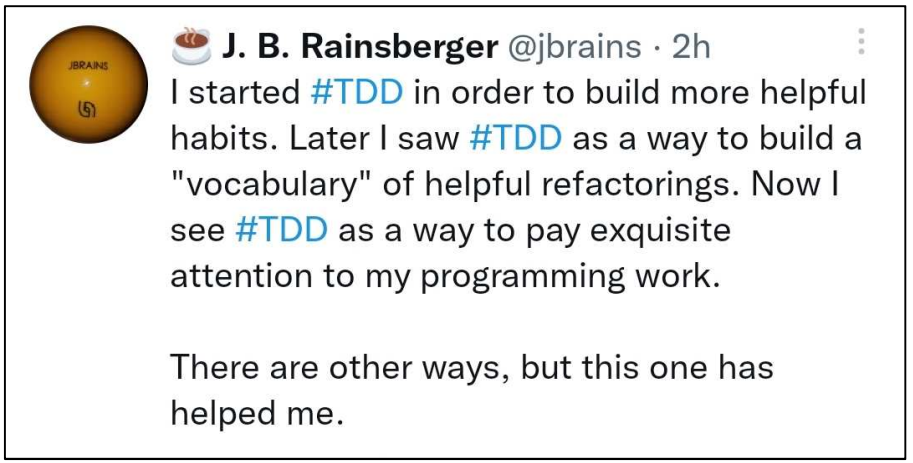


Test Driven Development (3/4)

- Hints
- Write the minimum code needed to pass the test
  - Without exaggerations
    - No every single cases
  - Don't think too far
- Useful tests are those that fail!
  - Don't write tests you already know the current code passes

Test Driven Development (4/4)

- Pros
  - The code is
    - More correct
    - More modular
    - Ready for the regression test
  - The bugs are found **quickly**
- Cons
  - Requires a lot of **more code** (up to 400% more)
  - Maintenance is **still** needed



Pair Programming (1/2)

- The work is carried out in **couples**:
  - One writes the code (**driver**)
  - One checks and suggests (**navigator**)
- Every 30 minutes the roles are **exchanged**
- The **quality** of code is higher
- The **costs** decrease (counterintuitively)
- Some studies point out that the speed decrease of 15% but the bugs decrease of 50%

Pair Programming (2/2)

- Pros
  - **Less** management risks
  - Automatic **tutoring**
  - **Higher** satisfaction
- Cons
  - **Difficult to apply** to programmers that prefer to work alone
  - Junior programmers can be **over-influenced** by senior programmers
  - Expert programmers are **bored** by novices
  - **Personality conflict** between expert programmers

Refactoring

- It is the modification of the **internal** structure of the code **without** modifying the behavior
- It can improve:
  - Readability
  - Reusability
  - Extensibility
  - Performance

- *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.* M. Fowler

Refactoring definitions

- Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behavior.

The rule of three

- The Rule of Three (by Don Roberts)
  - The first time you do something, you just do it
  - The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway
  - The third time you do something similar, you refactor
- Tip: Three strikes and you refactor

When to refactor?

- Refactor when you **add functions**
- Refactor when you need to **fix a bug**
- Refactor as you do **code review**

When to NOT refactor?

- There are times when the existing code is such a **mess** that although you could refactor it, it would be easier to **start from the beginning**
- The other time you should avoid refactoring is when you are **close to a deadline**. At that point the productivity gain from refactoring would appear after the deadline and thus be too late

Example of refactoring – data structure (1/3)

```
public class Library {
   private Book[] books =  new Book[10000];
   ...
   public Book getBook(int index) {
      return books[index]; }
}
```

- Problems:
  - Fixed size
    - Too much allocated memory at the beginning
    - No more than 10000 books
  - Difficult to manage adding and removing books

Example of refactoring – data structure (2/3)

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Actions
  - Exploit a dynamic data structure
    - Dynamic memory management
    - Dynamic addition and removal of items

Example of refactoring – data structure (3/3)

```
public class Library {
  private ArrayList<Book> books =  new
Arraylist<Book>();
  ...
  public Book getBook(int index) {
    return books.elementAt(index); }
}
```

- After refactoring:
  - Dynamic size
    - Fair amount of memory
  - Books can be added and removed easier

Example of refactoring – hierarchy (1/4)

```
public void printTree(TreeNode t) {
  if (t.isLeaf())
    System.out.println(t.getInfo());
  else
    for (TreeNode st : t.getChildren())
      printTree(st);
}
public class TreeNode {
  private Object info;
  private boolean leaf;
  ...
}
```

- Problems:
  - External iteration (may not be optimized)

Example of refactoring – hierarchy (2/4)

- Actions
  - Define classes that implement the operation(s)
  - Create a hierarchy

Example of refactoring – hierarchy (3/4)

```
public void printTree(TreeNode t) {
   t.print();
}
public class TreeNode {
   private Object info;
   private boolean leaf;
   ...
}
```

Example of refactoring – hierarchy (4/4)

```
public class TreeNodeDir extends TreeNode {
  public void print() {
  for (TreeNode st : getChildren())
      st.print();
  }
}
public class TreeNodeLeaf extends TreeNode {
  public void print() {
    System.out.println(getInfo());
  }
}
```

Example of refactoring – code extraction (1/3)

```
if (args.length == 0) {
  int i; String host = "localhost";
  Socket s = new Socket(host, 80);
  InputStream is = s.getInputStream();
  while ((i = is.read()) != -1)
    System.out.print((char) i);
}
else {
  int i; String host = args[0];
  Socket s = new Socket(host, 80);
  InputStream is = s.getInputStream();
  while ((i = is.read()) != -1)
    System.out.print((char) i);
}
```

- Problems:
  - Duplication of code
  - Difficult to read

Example of refactoring – code extraction (2/3)

- Actions
  - Extract the common code
  - Put it in a function
  - Call the function with appropriate arguments

Example of refactoring – code extraction (3/3)

```
private void printFromHost(String host){
   int i; Socket s = new Socket(host, 80);
   InputStream is = s.getInputStream();
   while ((i = is.read()) != -1)
     System.out.print((char) i);
}

if (args.length == 0)
   printFromHost("localhost");
else
   printFromHost(args[0]);
```

Example of refactoring – polymorphism (1/3)

```
if (extension.equals("txt")) {
   TxtFile t = new TxtFile(path);
   t.getInfo();
}
if (extension.equals("doc")) {
   DocFile d = new DocFile(path);
   d.getInfo();
}
```

- Problems:
  - Duplication of code

Example of refactoring – polymorphism (2/3)

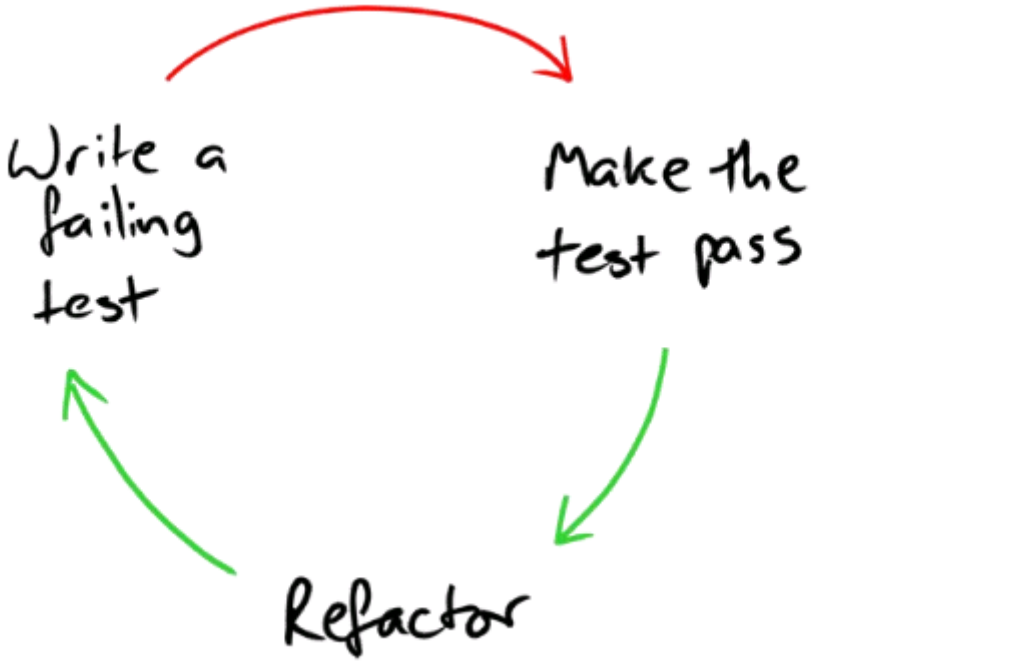
- Actions
  - Define a hierarchy with polymorphic method(s)
  - Exploit the polymorphism to call the polymorphic method once

Example of refactoring – polymorphism (3/3)

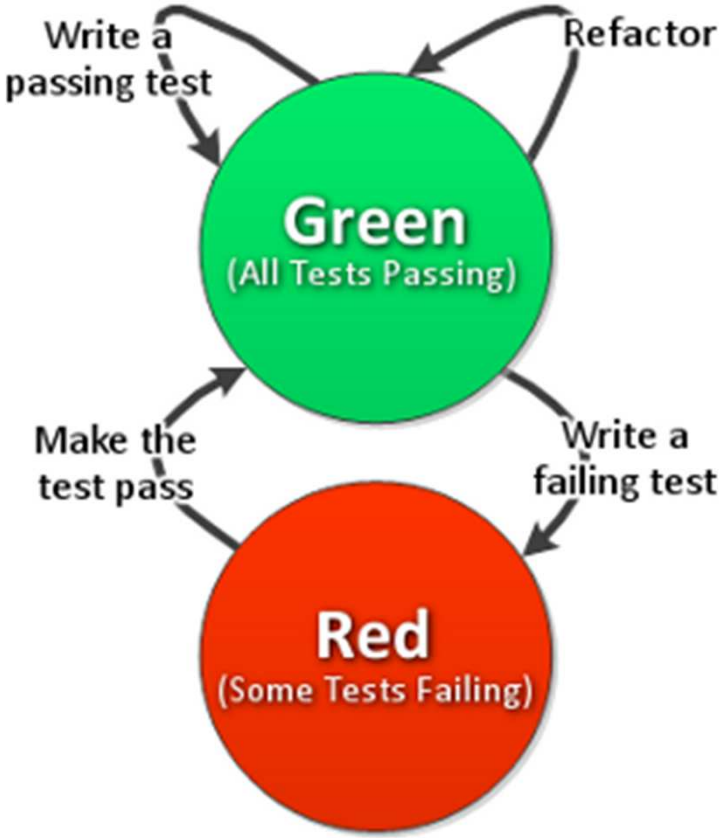
```
GenericFile f;
if (extension.equals("txt")) {
   f = new TxtFile(path);
}
if (extension.equals("doc")) {
   f = new DocFile(path);
}
f.getInfo();
```

Example of hardware refactoring

TDD + Refactoring (1/2)



TDD + Refactoring (2/2)



Cross functional team

- Every team must be composed of persons that, as a whole, can carry out the task
- Each person in the team can do more than one thing
  - No rigid specialists
  - People able to self-organize, without a defined leader
  - People able to work in parallel
  - More available information
  - More points of view
- Not easy from the social point of view…

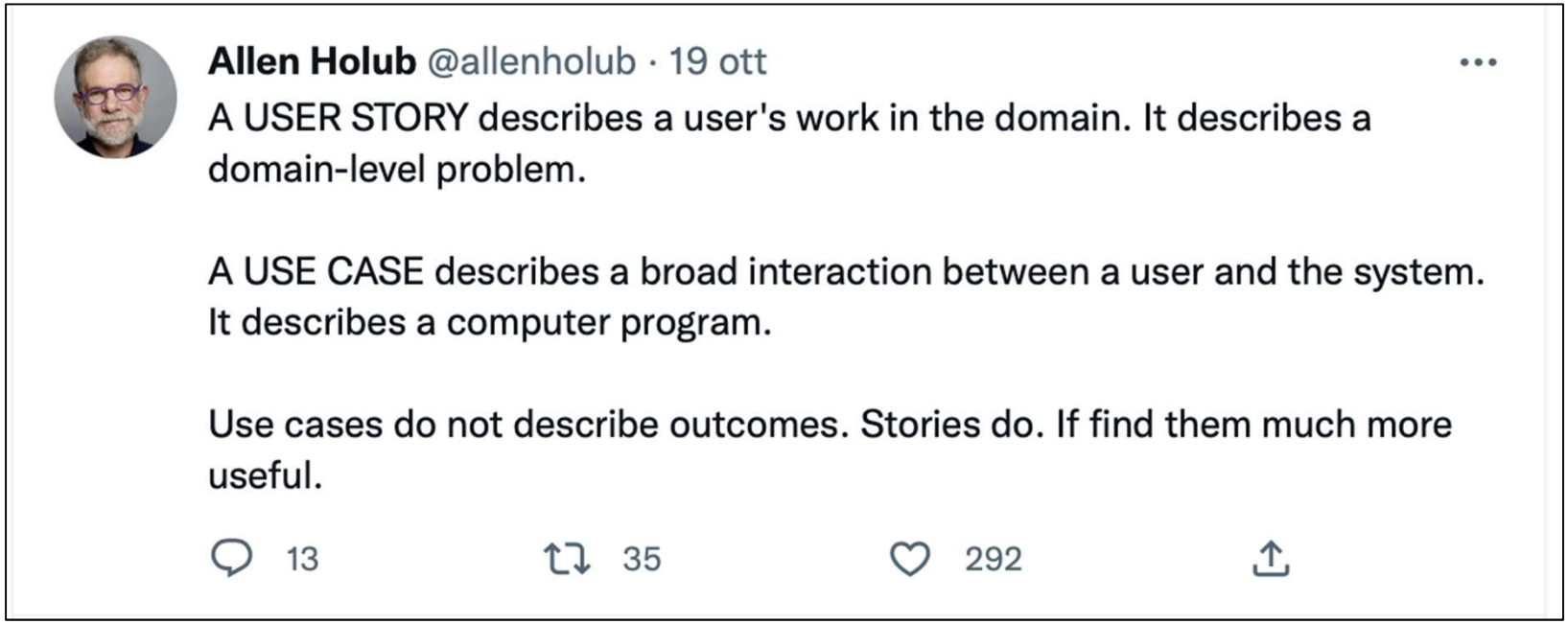
Timeboxing

- This term was introduced in 1981 and was used in RAD models of '90
- The idea is to force the work to be completed in the **predefined time**
- Pros
  - Faster development process
  - Faster delivery of the software
- Cons
  - The management of the project is more difficult
  - Not suitable for projects that cannot be divided into simple iterations

User stories

- User stories are descriptions of what users do with the software system or need from the software system
  - Everyday or business language is exploited
- Examples
  - "As a <*role*>, I want <*goal/desire*> so that <*benefit*>"
  - "As <*who*> <*when*> <*where*>, I <*what*> because <*why*>"
  - "As <*persona*>, I want <*what?*> so that <*why?*>"
  - And similar variants
- They are useful to define requirements in an informal and quick way
- Usually, stories derive from **epics**, more general descriptions

User stories vs. Use cases



User stories examples (Mike Cohn)

- Epic
  - As a user, I want to backup my entire hard drive.
- Stories
  - As a power user, I want to specify files or folders to backup based on file size, date created and date modified.
  - As a user, I want to indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.

User stories examples (Scrum Alliance Website)

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Profile
  - As a Practitioner, I want my profile page to include additional details about me (i.e., some of the answers to my Practitioner application) so that I can showcase my experience.
  - As a trainer, I want my profile to list my upcoming classes and include a link to a detailed page about each so that prospective attendees can find my courses.
  - As a site member, I can view the profiles of other members so that I can find others I might want to connect with.

User stories examples (Scrum Alliance Website)

- News
  - As a site visitor, I can read current news on the home page so that I stay current on agile news.
  - As a site visitor, I can access old news that is no longer on the home page, so I can access things I remember from the past or that others mention to me.
  - As a site editor, I can set the following dates on a news item: Start Publishing Date, Old News Date, Stop Publishing Date so articles are published on and through appropriate dates. These dates refer to the date an item becomes visible on the site (perhaps next Monday), the date it stops appearing on the home page, and the date it is removed from the site (which may be never).

User stories examples (Scrum Alliance Website)

- Home Page
  - As a site editor, I want to have a prominent area on the home page where I can put special announcements, not necessarily news or articles, so that I can give them additional exposure.
  - As a site editor, I'd like to have some flexibility as to where things appear so as to accommodate different types of content.
  - As a trainer, the upcoming courses are what I want visitors to notice so that they register and there's a benefit to my membership.
  - As a site visitor, I want to see new content when I come to the site, so I come back more often.

User stories (4/4)

- Pros
  - Fast requirements collection
  - Better management of requirements changes
  - Solicit user-developer interactions
- Cons
  - Stories are likely to be vague and incomplete
  - Difficult to scale up
  - Non-functional requirements are difficult to capture

Is agile useful?

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000

The success of agile models

- The agile models are widely accepted
  - Especially in **small** companies
  - But also in **large** companies
  - And also in companies of **other fields** (e.g., Toyota)
- They must be **correctly** applied to provide advantages over the traditional methods
- But it is **not easy**
- They require:
  - Self-control
  - Harmony
  - Planning
  - Experience (perhaps the most important)

Agile: not for all

- The agile models are **not** a panacea
- They work worse:
  - With **large** projects
  - With **distributed** development teams
  - In companies with a "**Command & Control**" approach
  - When there are many **inexpert** developers
  - When agile is **not** understood
    - Agile **façade** but traditional work

Bermuda triangle of Agile

Caveat

Gergely Orosz
@GergelyOrosz

What many sw engineers don't realize:

The majority of jobs do not hire you to write the highest quality code or produce the cleanest architecture.

They hire you to solve their business problems very efficiently. Sometimes this means high-quality code. Sometimes not at all.

Traduci il Tweet

16:31 · 10 Apr 22 · Twitter Web App

1.285 Retweet **202** Tweet di citazione

8.736 Mi piace

Gergely Orosz @GergelyOrosz · 2g
In risposta a @GergelyOrosz
And to add, sometimes this means deleting code.

Agile methodologies

- We will see some agile methodologies
  - Agile Unified Process
  - Scrum
  - Feature Driven Development (FDD)
  - Dynamic System Development Method (DSDM)
  - eXtreme Programming (XP)
- We will discuss about Agile Modeling

Agile methods vs. heavy methods

	Agile Methods	Heavy Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Project size	Small	Large
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable/Exploratory	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in Project	End of Project
Team	Small/Creative	Large

From: "A Comparison between Agile and Traditional Software Development Methodologies", by M.A. Awad)

User stories by Dilbert