

# Agile Software Development

Maurizio Morisio

Marco Torchiano

Version 1.1



**SoftEng**  
<http://softeng.polito.it>

© Maurizio Morisio, Marco Torchiano, 2005



## Outline

- Agile methodologies
- XP (eXtreme Programming)
- Research Topics
- Test Driven Development
  
- License

**SoftEng**  
<http://softeng.polito.it>

# Agile Methodologies

---

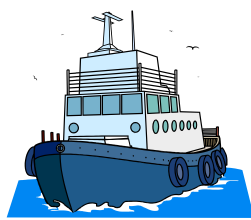
Part 1



**SoftEng**  
<http://softeng.polito.it>

# Software Development

---



Building a ship?



Growing a garden?

**SoftEng**  
<http://softeng.polito.it>

---

## Traditional development

---

- ◆ ..or heavyweight
- Emphasis on documentation, process
  - ◆ Waterfall, prototyping, iterative,
  - ◆ Iso 9000, Vision, CMM
- Still, some projects fail (do they?)

**SoftEng**  
<http://softeng.polito.it>

---

## Agile

---

- ◆ .. Or lightweight
- Teamwork and communication
- Change as a constant, give up on predictability
- Iterations and reaction to change
  - ◆ Actually, nothing really new, but mix is innovative

**SoftEng**  
<http://softeng.polito.it>

---

## Agile Manifesto

---

- In the 90's several consultants (Beck, Cockburn, Schwabern, ..) observed failed projects, especially in large organizations, developed specific lean processes (XP, Crystal, Scrum) and agreed on a Manifesto

## Agile Manifesto

---

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

# eXtreme Programming

---

Part 2



## Outline

---

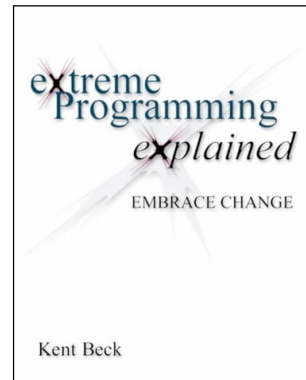
- Principles and Background
- The 12 practices
- XP practices in detail
- Issues in XP adoption



## Extreme programming

---

- Kent Beck: Extreme Programming Explained Addison-Wesley, 2000, ISBN 0-201-61641-6



**SoftEng**  
<http://softeng.polito.it>

---

## Principles and background

---

**SoftEng**  
<http://softeng.polito.it>

---

## When can XP be used?

---

- Small projects:
  - ◆ 2-10 developers, maybe 20
- Developer and customer representative are co-located
- Problems:
  - ◆ Point-and-go culture
  - ◆ Testing takes hours to execute

**SOftEng**  
<http://softeng.polito.it>

---

## Fundamentals of XP

---

- Distinguish between decisions made by business stakeholders and developers
- Simplistic – keep design as simple as possible  
“design for today not for tomorrow”
- Write automated test code before writing production code and keep all tests running
- Pair programming
- Very short iterations with fast delivery

**SOftEng**  
<http://softeng.polito.it>

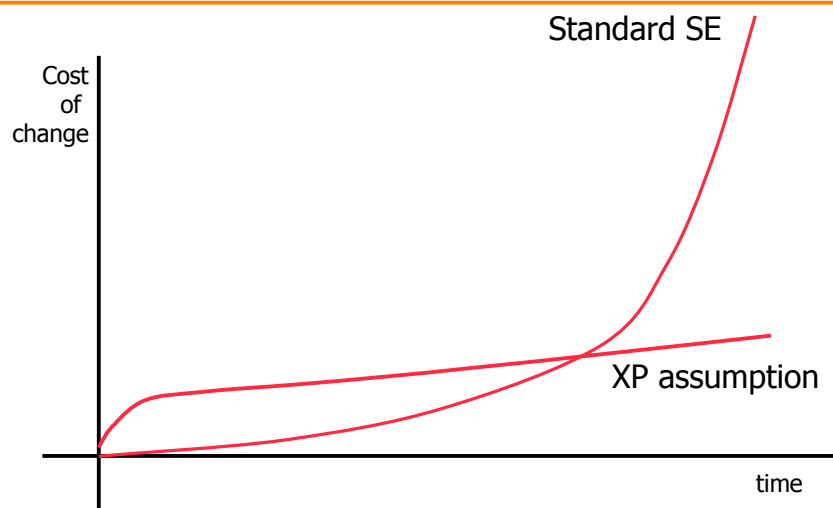
---

## Why is XP controversial?

- No specialists – every programmer participates in architecture, design, test, integration
- No up-front detailed analysis and design
- No up-front development of infrastructure
- Not much writing of design & implementation documentation beside tests and code

SoftEng  
<http://softeng.polito.it>

## Cost of change



SoftEng  
<http://softeng.polito.it>

## Some basic facts

---

- Producing code is required to deliver a system
- Dollars spent on analysis and design are wasted if the system is never used
- Business requirements have to be the drivers for software development
- Requirements change

**SoftEng**  
<http://softeng.polito.it>

---

## Back to the basics

---

- Coding
- Testing
- Listening
- Designing

**SoftEng**  
<http://softeng.polito.it>

---

## Four values

---

- **Communication**
  - ♦ “problems with projects can invariably be traced to somebody not talking to somebody else about something important” p 29
- **Simplicity**
  - ♦ “what is the simplest thing that could possibly work?”
- **Feedback**
  - ♦ Put system in production ASAP
  - ♦ “Have you written a test case for that yet?”
- **Courage**
  - ♦ Hill climbing (simple, complex, simpler,...)
  - ♦ Big jumps take courage

**SoftEng**  
<http://softeng.polito.it>

---

## Facilities

---


- **Should improve communication**
  - ♦ Make sure that pairs can work effectively
  - ♦ Put developers close to each other
  - ♦ Open spaces

**SoftEng**  
<http://softeng.polito.it>

---

## Four project variables

---

- Cost
    - ♦ “Adding people to a late project just makes it later”
  - Scope
  - Time
  - Quality
-  Time-boxed

## The key practices

---

## 12 practices



### Customer satisfaction

- ◆ On-site customer
- ◆ Small releases



### Software quality

- ◆ Metaphor
- ◆ Simple design
- ◆ Refactoring
- ◆ Pair programming
- ◆ Testing



### Project management

- ◆ Planning game
- ◆ Sustainable development
- ◆ Collective code ownership
- ◆ Continuous integration
- ◆ Coding standards

**SoftEng**  
<http://softeng.polito.it>

## On-site customer



- Many software projects fail because they do not deliver software that meets business needs
- Real customer has to be part of the team
  - ◆ Defines business needs
  - ◆ Answers questions and resolves issues
  - ◆ Prioritizes features

**SoftEng**  
<http://softeng.polito.it>

## Small releases



- Put system into production ASAP
  - ◆ Fast feedback
- Deliver valuable features first
- Short cycle time
  - ◆ Planning 1-2 months is easier than planning 6-12 months

**SoftEng**  
<http://softeng.polito.it>

## Metaphor/Architecture



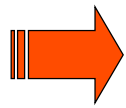
- How does the whole system work?
- What is the overall idea of the system?
- Initially: Architectural spike

**SoftEng**  
<http://softeng.polito.it>

## Simple design



- The “right” design
  - ♦ Runs all tests
  - ♦ No code duplication
  - ♦ Fewest possible classes and methods
  - ♦ Fulfills all *current* business requirements



Design for today not the future

**SOftEng**  
<http://softeng.polito.it>

## Refactoring



- Restructure system without changing the functionality
- Goal: Keep design simple
  - ♦ Change bad design when you find it
  - ♦ Remove dead code

**SOftEng**  
<http://softeng.polito.it>

## Pair programming



- “All production code is written with two people looking at one machine”
  - ♦ Person 1: Implements the method
  - ♦ Person 2: Thinks strategically about potential improvements, test cases, issues
- Pairs change all the time
- Advantages
  - ♦ No single expert on any part of the system
  - ♦ Training on the job
  - ♦ Permanent inspections
- Problems:
  - ♦ Wasted development time?
  - ♦ Pairs need to function

**SoftEng**  
<http://softeng.polito.it>

## TDD



- *Automatic* test drivers
- Write tests before production code
  - ♦ Unit tests → developer
  - ♦ Feature/acceptance tests → customer
- Strong emphasis on regression testing
  - ♦ Unit tests need to execute all the time
  - ♦ Tests for completed features need to execute all the time
- Unit tests pass 100%
- Acceptance tests show progress on user stories

**SoftEng**  
<http://softeng.polito.it>

## The planning game



- Business decisions
  - ♦ Scope: which “stories” should be developed
  - ♦ Priority of stories
  - ♦ Composition of releases
  - ♦ Release dates
- Technical decisions
  - ♦ Time estimates for features/stories
  - ♦ Elaborate consequences of business decisions
  - ♦ Team organization and process
  - ♦ Scheduling

**SoftEng**  
<http://softeng.polito.it>

## Sustainable development



- Developing full speed only works with fresh people
- Working overtime for two weeks in a row indicates problem

**SoftEng**  
<http://softeng.polito.it>

## Collective ownership



- All code can be changed by anybody on the team
- Everybody is required to improve any portion of bad code s/he sees
- Individual code ownership tends to create experts

**SOftEng**  
<http://softeng.polito.it>

## Continuous integration



- Integration happens after a few hours of development
  - ◆ Code is released into current baseline on integration machine
  - ◆ All tests are run
  - ◆ In case of errors:
    - Reverse to old version
    - Fix problems
    - Goto (1)

**SOftEng**  
<http://softeng.polito.it>

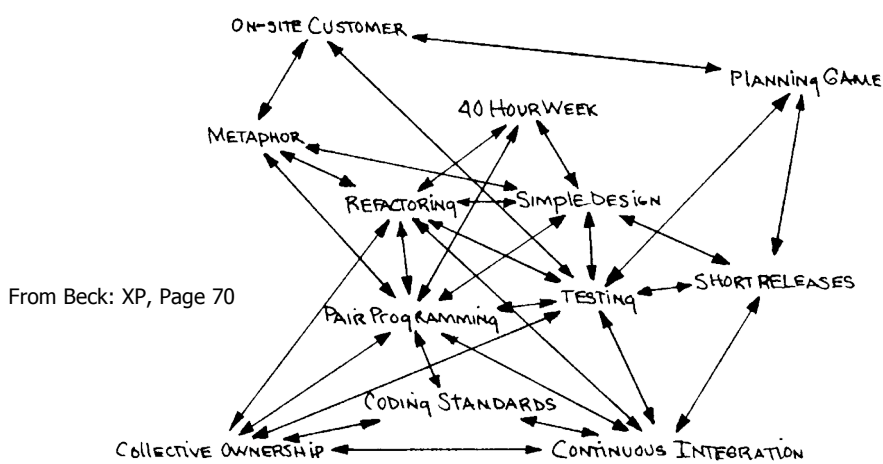
## Coding standards



- Team has to adopt a coding standard
  - ♦ Makes it easier to understand other people's code
  - ♦ Avoids code changes because of syntactic preferences

SoftEng  
<http://softeng.polito.it>

## How everything fits together



SoftEng  
<http://softeng.polito.it>

## XP Practices in detail

---

## Requirements definition

---

- User stories
  - ♦ On index cards
  - ♦ Short descriptions (about 3 sentences) of a feature
  - ♦ In customer language, no techno babble
  - ♦ Provide value to customer
  - ♦ Independent of each other
  - ♦ Testable
  - ♦ Estimated by developers
  - ♦ Small → decompose large stories
- Collect story cards and *prioritize* them

## Architectural spike

---

- Throw-away prototype
- Answers technical issue
- Reduce technical risk or improve reliability
- Usually: Pair for 1-2 weeks

## Planning

---

- Release planning
  - ♦ Chooses a few months worth of user stories
  - ♦ Date and scope
  - ♦ Can be changed
- Iteration planning
  - ♦ Few weeks
  - ♦ Set of stories prioritized by customer
  - ♦ Define set of tasks for each story
- Never slip a date → change scope

## Estimation

---

- Based on similar stories from the past (“yesterday’s weather”)
- Team effort: optimism wins
- Estimates are not commitments
- Ideal Engineering Time (IET) → no interruption

## Steering phase

---

- Iteration (1-3 weeks): task cards
- Recovery after overestimating velocity
- New stories

## Management strategy

---

- Team members accept responsibility
- Committed to do quality work
- Not much management overhead
- Basic measurement
  - ◆ Ratio between estimated development time and calendar time
  - ◆ Percentage of feature complete
- Coaching

**SoftEng**  
<http://softeng.polito.it>

---

## Tracking progress

---

- Two questions
  - ◆ How many ideal hours/days have you worked?
  - ◆ How many more does it take
- Project metrics
  - ◆ Velocity = IET/Calendar Time
  - ◆ Bugs
  - ◆ Time worked on a task
  - ◆ Stories completed
  - ◆ #Acceptance tests defined and passing
  - ◆ #unit tests
  - ◆ ...

**SoftEng**  
<http://softeng.polito.it>

---

## Testing

---

- Write tests before production code
- Use JUnit
- Acceptance tests for user stories
- Unit tests for methods
- Keep as much functionality out of UI code as meaningful

## When is a user story done?

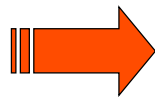
---

- All unit tests pass
- All acceptance test pass
- The customer accepts it
- All refactorings are done

## What is refactoring?

---

- Changing the code without changing its functionality
- Goal: make code easier to maintain



investment into the future

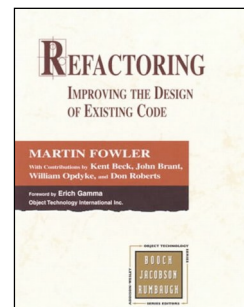
**SOftEng**  
http://softeng.polito.it

---

## Refactoring

---

- Refactoring book by Fowler
- <http://www.refactoring.com>



**SOftEng**  
http://softeng.polito.it

---

## Why refactor?

---

- To improve the design of software
- To make code easier to understand
- To help find bugs
- As a result
  - ◆ Coding becomes faster
  - ◆ Outside source code documentation less required

## When do you refactor?

---

- When the code “smells bad”
  - ◆ Repeating code
  - ◆ Code difficult to understand
  - ◆ Long methods
  - ◆ Long message chains
  - ◆ Switch statements instead of polymorphism
  - ◆ ...

## How do you refactor?

---

- If code smells bad
  1. make a change
  2. test the system.
  3. if it works goto (1) else goto (2)
  
- Refactoring is heavily dependent upon
  - ◆ automated test drivers and
  - ◆ working code.

## Stages in Refactoring

---

- Simple Stuff
- Cleanup and Rearrangement
- Advanced Stuff

## Simple Stuff

---

- Rename variables and procedures
- Replace temp variables with procedure calls
- Replace “Magic Numbers” with constants

## Cleanup

---

- Split up large blocks of code into individual methods
- Make tiny method bodies inline
- Create temp variables to simplify complicated expressions

## Advanced Stuff

---

- Replace Conditional Logic with Polymorphism
- Shorten Parameter Lists
- Move Methods and Fields

**SOftEng**  
http://softeng.polito.it

---

## Example

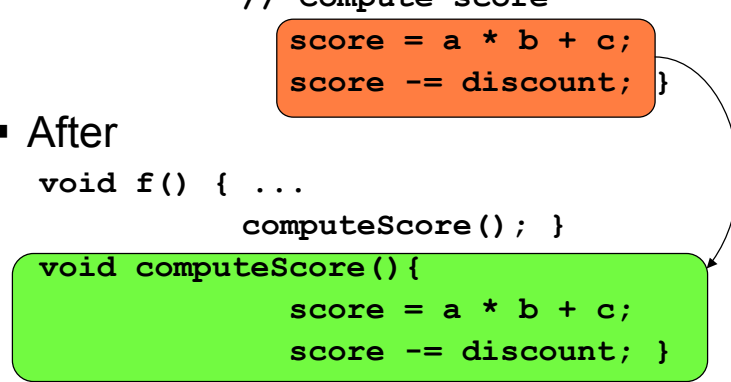
---

- Before

```
void f() { ...  
    // Compute score  
    score = a * b + c;  
    score -= discount;  
}
```

- After

```
void f() { ...  
    computeScore();  
}  
  
void computeScore() {  
    score = a * b + c;  
    score -= discount;  
}
```



**SOftEng**  
http://softeng.polito.it

---

## Example

---

- Before

```
String a,b,c;  
return a+b+c;
```

- After

```
StringBuffer sb = new StringBuffer(a);  
sb.append(b);  
sb.append(c);  
return sb.toString();
```

**SoftEng**  
<http://softeng.polito.it>

---

## Issues in XP adoption

---

**SoftEng**  
<http://softeng.polito.it>

---

## All techniques?

---

- Proposers state that combination of all techniques provide highest benefit
- Stepwise adoption
  - ◆ Pick your worst problem and apply corresponding XP technique

## Business contracts

---

- Fixed scope/fixed price contracts problematic – why?
- Fixed cost and fixed programmer hours

## Colocation and project size

---

- Co-location of team members required
- Scalability of the process:  
Small teams → small projects

**SoftEng**  
<http://softeng.polito.it>

---

## Research Topics

---

Part 3

 **SoftEng**  
<http://softeng.polito.it>

## Research Topics

---

- Pair programming effectiveness
- TDD effectiveness
- Compatibility between XP and “heavy” practices (e.g. CMM)
- Economics of Agility

## Pair programming - effects

---

- More quality and
- Less productivity?

## Williams

---

- Williams, Laurie, Kessler, Robert R., Cunningham, Ward, and Jeffries, Ron, [Strengthening the Case for Pair-Programming](#), *IEEE Software*, July/Aug 2000 .
  - ♦ University study with 41 students
  - ♦ Higher quality code
    - Test cases passed individuals: 73.4%-78.1%
    - Test cases passed pairs: 86.4%-94.4%
  - ♦ Pairs completed assignments 40-50% faster (average 15% higher costs)
  - ♦ Pair programming preferred by students (85%)

**SoftEng**  
<http://softeng.polito.it>

---

## Long (on 5 studies)

---

- Quality
  - ♦ Better quality PP than solo programmers
  - ♦ Meaningful effect
  - ♦ Both with students and professionals
  - ♦ Improves quality of program without impacting quality of programmer
    - Average programmers benefit from it
- Not difficult to learn
  - May be more difficult for more skilled programmers

**SoftEng**  
<http://softeng.polito.it>

---

## Long (on 5 studies)

---

- Productivity
  - ♦ PP lower productivity than solo programmer
  - ♦ Meaningful effect
  
- ♦ One study suggests that PP may have same productivity in context of difficult algorithms – changing requirements

## TDD – effects

---

- More quality And
- Less productivity?

## Erdogmus, Morisio, Torchiano

---

- Experiment with 25 students
- Test first vs. Test last
- No statistical difference in quality and productivity
- However, TF write more tests
- More tests correlate with more productivity!
- More tests imply a minimum quality level

**SOftEng**  
<http://softeng.polito.it>

---

## Long (on 5 studies)

---

- Quality
  - ◆ Quality is higher for TDD
  - ◆ Meaningful effect
- Productivity
  - ◆ Not clear
- Difficult to learn

**SOftEng**  
<http://softeng.polito.it>

---

## CMM and agile

---

- Agile proponents usually consider them not reconcilable
  - ♦ Because of planning, measuring, documentation
- CMM proponents are more flexible
  - ♦ XP project could be rated level 2 (Paulk)
  - ♦ XP project could raise to level 3 (Glazer)
  - ♦ XP has no practices at level 4-5 though
  - ♦ XP is a methodology (how), CMM is the management environment (what), they complete each other (Paulish, Siemens)
- Some are harsh
  - ♦ Agile is a step back to hacking (Rakitin)

**SoftEng**  
<http://softeng.polito.it>

---

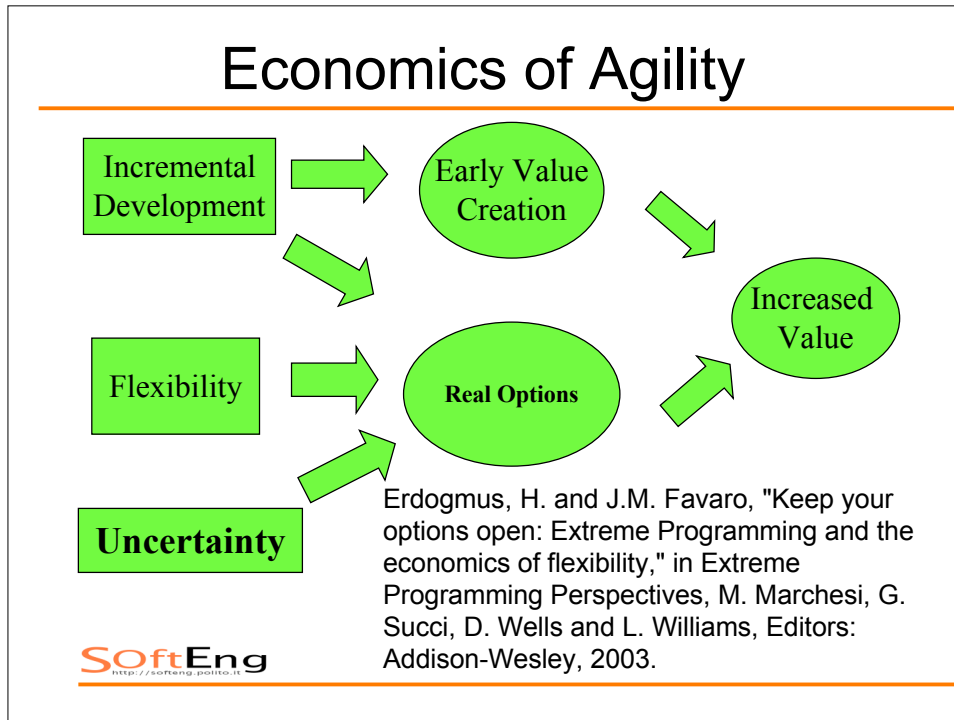
## References

---

- M. C. Paulk. Extreme programming from a CMM perspective. In IEEE Software, Nov-Dic 2001. pp. 19-26
- S. Rakitin, *Manifesto Elicits Cynicism*. IEEE Computer, 2001. 34(12). p. 4
- T. DeMarco, B. Boehm, *The Agile Methods Fray*. In IEEE Computer 35(6), June 2002, pp. 90-92.
- B. Boehm, *Get Ready for Agile Methods, with Care*. IEEE Computer 35(1), January 2002. pp. 64-69.

**SoftEng**  
<http://softeng.polito.it>

---

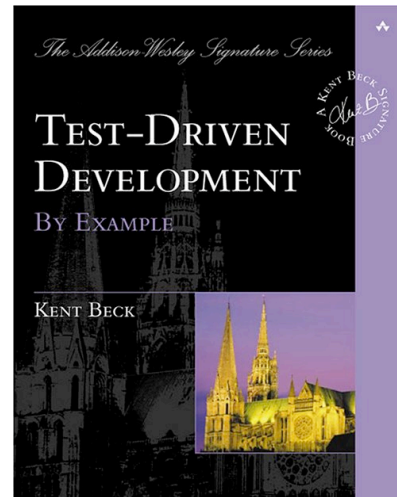


## Test Driven Development

Part 4

## Test-Driven Development

K. Beck, *Test-Driven Development: by Example*: Addison Wesley, 2003.



**SOftEng**  
http://softeng.polito.it

## Test cases come codice

```
testRaddoppia2(){  
    assertEquals(4, raddoppia(2));  
} // PASS
```

```
testRaddoppia3(){  
    assertEquals(6, raddoppia(3));  
} // FAIL
```

**SOftEng**  
http://softeng.polito.it

## Junit

---

- Test framework
  - ♦ plug-in di Eclipse
  - ♦ per fare test cases come codice Java
  - ♦ framework: set di classi e convenzioni per usarle

```
testRaddoppia2(){  
  
}
```

```
int raddoppia(){  
  
}
```

**SoftEng**  
http://softeng.polito.it

## Junit

---

- É possibile usare JUnit all'interno di Eclipse per eseguire i programmi invece di scrivere delle classi con il metodo `main()`.
- É sufficiente:
  - ♦ scrivere una sotto-classe della classe `TestCase`
  - ♦ aggiungere ad essa dei metodi di test

**SoftEng**  
http://softeng.polito.it

## Junit - set up

---

- in Eclipse
- aprire project's property window
- java build path
- libraries
- external jar
  - ◆ add org.junit

**SoftEng**  
<http://softeng.polito.it>

---

## Elementi del framework

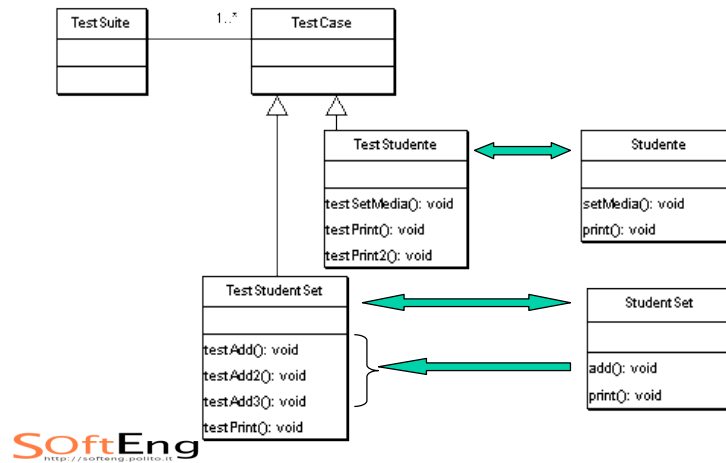
---

- `assert*()`
  - ◆ funzioni di confronto
- `TestCase`
  - ◆ classe contenente serie di test
  - ◆ una per ogni classe di production code
  - ◆ vari test per ogni metodo di classe production
- `TestSuite`
  - ◆ classe contenente serie di `TestCase`

**SoftEng**  
<http://softeng.polito.it>

---

## Framework e convenzioni



## Assert\*()

- ◆ Metodi definiti su TestCase
  - per condizione
    - ◆ assertTrue("message when test fails", condition);
  - per valori di ritorno object, int, longs, byte
    - ◆ assertEquals(expected\_value, expression);
  - per valori di ritorno float, double:
    - ◆ assertEquals(expected\_value, expression, error);
  - se la condizione testata
    - ◆ e' vera, esegue istruzione seguente
    - ◆ e' falsa, break a fine metodo

## Studente.get/setMediaVoti()

### •Test code

```
public class TestStudiante extends
TestCase {
public void testGetMediaVoti() {
    Studente s = new Studente();
    s.setMediaVoti(25.5);
    assertEquals(25.5,
        s.getMediaVoti(), 0.005);
}
}
```

**SoftEng**  
http://softeng.polito.it

### •Production code

```
public class Studente {
    int matricola;
    double mediaVoti;
    Studente next;

    void setMediaVoti(
        double voto){
        mediaVoti = voto; }
    double getMediaVoti() {
        return mediaVoti; }
}
```

## Regole

### •Test code

```
public class TestStudiante extends TestCase {
    public TestStudiante(String name){
        super(name);
    }
    public void testGetMediaVoti() {
        Studente s = new Studente();
        s.setMediaVoti(25.5);
        assertEquals(25.5,
            s.getMediaVoti(), 0.005);
    }
}
```

deve esserci

nome metodo deve  
iniziare con "test"

**SoftEng**  
http://softeng.polito.it

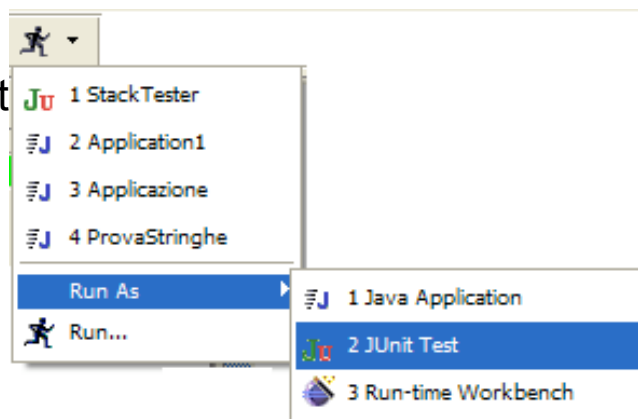
## Funzionamento

- Per un test case JUnit:
  - ♦ Esegue tutti i suoi metodi di test pubblici
    - Ovvero quelli che iniziano con “test”
  - ♦ Ignora tutto il resto
- La classe può contenere metodi di supporto (helper methods)
  - ♦ Non sono pubblici o non iniziano con “test”
  - ♦ Possono essere chiamati dai metodi di test

SOftEng  
<http://softeng.polito.it>

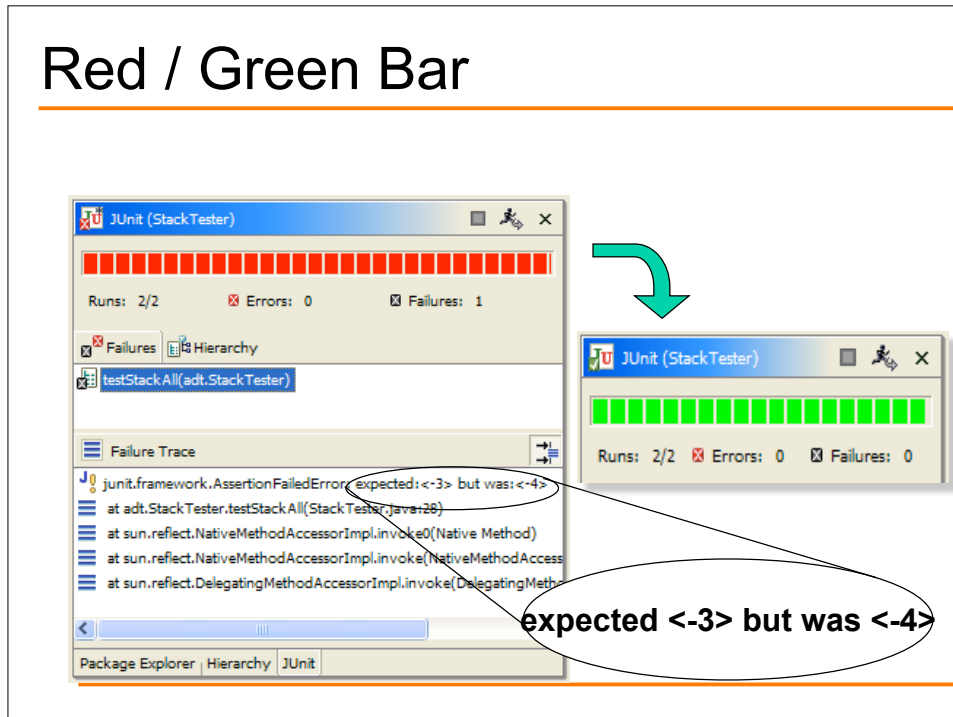
## Run as JUnit Test

- Run
- Run As..
- Junit Test



SOftEng  
<http://softeng.polito.it>

## Red / Green Bar



## Organizzazione tests

- test cases di classe
  - ♦ test case e' gia test suite
- suite per test case di tutte classi

## Organizzazione

---

- TestStudente extends TestCase
  - ◆ tutti test case di Studente
- TestStudentSet extends TestCase
  - ◆ tutti test case di StudentSet
- AllTest extends TestSuite
  - ◆ con test case di Studente e StudentSet

## Strategie per il test

---

- Unit tests (developer tests)
- Acceptance tests (customer tests)

## Studente

---

- Nominal: voti tra 0 e 33
- Boundary: 0 e 33
  - ◆ testare 0 e 33
- Exceptional
  - ◆ testare fuori dal boundary

**SoftEng**  
http://softeng.polito.it

---

## setMediaVoti() - nominale

---

```
public class TestStudente extends TestCase {  
    public void testGetMediaVoti() {  
        Studente s = new Studente();  
        s.setMediaVoti(25.5);  
        assertEquals(25.5, s.getMediaVoti(), 0.005);  
    }  
}
```

**SoftEng**  
http://softeng.polito.it

---

## setMediaVoti() - boundary

```
public void testMediaVotiBoundary() {  
    Studente s = new Studente();  
    s.setMediaVoti(0);  
    assertEquals(0, s.getMediaVoti(), 0.005);  
    s.setMediaVoti(33);  
    assertEquals(33, s.getMediaVoti(), 0.005);  
}
```



## setMediaVoti() - exceptional

```
public void testMediaVotiNonValida() {  
    Studente s = new Studente();  
    s.setMediaVoti(-1);  
    assertEquals(0, s.getMediaVoti(), 0.005);  
    s.setMediaVoti(100);  
    assertEquals(33, s.getMediaVoti(), 0.005);  
}
```



## Regression test

---

- ◆ dato metodo o classe, e test suite TS che passa con successo
- ◆ data modifica/estensione di production code
- ◆ TS passa ancora con successo?
- Regression test = ripassare TS ogni volta che si modifica production code
- fatto automaticamente da JUnit

**SOftEng**  
<http://softeng.polito.it>

---

## Bibliografia Generale

---

- A. Cockburn, *Agile Software Development*: Addison Wesley, 2001.
- K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.
- K. Beck, *Test-Driven Development: by Example*: Addison Wesley, 2003.
- M.Fowler, *Refactoring: Improving the Desing of Existing*. Addison-Wesley, 1999.

**SOftEng**  
<http://softeng.polito.it>

---

---

## Test Driven Development

Es. Cassa del supermercato

**SOftEng**  
<http://softeng.polito.it>

---

## Cassa Supermercato

---

- L'obiettivo è realizzare il sistema software per la gestione della cassa di un supermercato
- Si tralasciano tutti i problemi relativi alla gestione dell'interfaccia utente e dell'hardware speciale (Es. lettore di codice a barre)

**SOftEng**  
<http://softeng.polito.it>

---

## Storia 1 – Sessione di cassa

---

### ▪ Lettura codice e stampa del totale

- ◆ Quando un nuovo cliente arriva alla cassa, la cassiera passa i prodotti sul lettore di codice a barre, poi termina la sessione.
- ◆ Quando la cassiera termina la sessione (con un apposito comando) la cassa stampa la somma dei prezzi dei prodotti letti.
- ◆ La lettura del codice a barre viene simulata con l'inserimento da tastiera dei codici dei prodotti.

```
> P001  
> P002  
> P001  
> CLOSE  
Totale: 7.5
```

**SoftEng**  
http://softeng.polito.it

---

## Storia 2 – Stampa descrizione

---

### ▪ Stampa della Descrizione

- ◆ Ogni volta che viene letto il codice di un prodotto la cassa accede ad un database di prodotti e recupera il prezzo e la descrizione del prodotto.
- ◆ Immediatamente dopo la lettura del codice la cassa stampa la descrizione del prodotto seguita dal prezzo.

```
> P001  
Acciughe 3.0  
> P002  
Aglione 1.5  
> P001  
Acciughe 3.0  
> CLOSE  
Totale: 7.5
```

**SoftEng**  
http://softeng.polito.it

---

## Storia 3 – Calcolo tasse (IVA)

- Quando la sessione è terminata la cassa deve stampare
  - ♦ il totale *senza* l'IVA,
  - ♦ l'IVA, e
  - ♦ il totale *con* l'IVA.
- NB:
  - ♦ occorre *scorporare* l'IVA del 20% dai prezzi

```
> P001
Acciughe 3.0
> P002
Aglione 1.5
> P001
Acciughe 3.0
> CLOSE
Senza IVA: 6.25
  IVA 20%: 1.25
Con IVA: 7.5
```

**SoftEng**  
http://softeng.polito.it

## Storia 4 – Calcolo sconto

- Prima che la sessione sia terminata la cassiera può inserire un codice speciale (*SCONTO*) seguito da un numero.
  - ♦ Questo indica che il totale verrà scontato della percentuale specificata.
- La cassa deve stampare il totale senza lo sconto, e l'importo dello sconto.

```
> P001
Acciughe 3.0
> P002
Aglione 1.5
> P001
Acciughe 3.0
> CLOSE
Senza sconto: 6.25
  sconto: 1.25
Senza IVA: 5.0
  IVA 20%: 1.0
Con IVA: 6.0
```

**SoftEng**  
http://softeng.polito.it

## Storia 5 – Stampa scontrino

- Quando viene terminata la sessione, la cassa deve stampare un sommario (scontrino) con i nomi ed i prezzi di tutti i prodotti acquistati.

**SoftEng**  
http://softeng.poitto.it

```

> P001
Acciughe 3.0
> P002
Aglione 1.5
> P001
Acciughe 3.0
> CLOSE
--- Scontrino ---
Acciughe 3.0
Aglione 1.5
Acciughe 3.0
Senza sconto: 6.25
          sconto: 1.25
Senza IVA: 5.0
  IVA 20%: 1.0
Con IVA: 6.0
    
```

## Storia 6 – Due x Uno

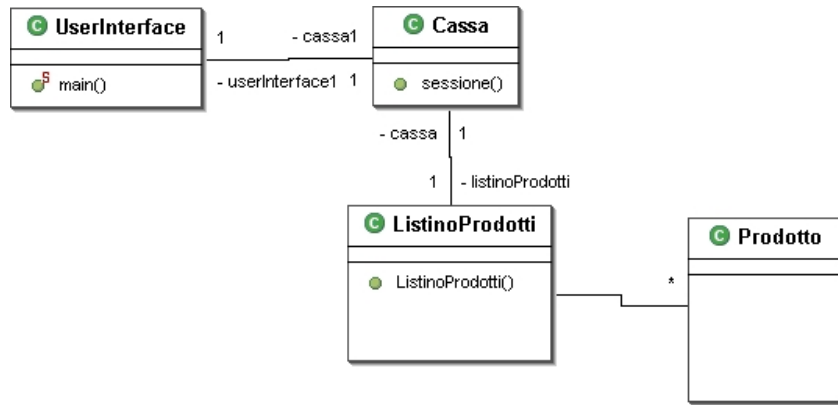
- Quando non c'è nessuna sessione in corso, si può definire una promozione per un prodotto. La cassiera inserisce un codice speciale (*PROMO*) seguito dal codice del prodotto da promuovere.
  - ♦ Per il prodotto in promozione, ogni due prodotti il secondo è gratis.

**SoftEng**  
http://softeng.poitto.it

```

> P001
Acciughe 3.0
> P002
Aglione 1.5
> P001
Acciughe *gratis*
> CLOSE
--- Scontrino ---
Acciughe 3.0
Aglione 1.5
Acciughe *gratis*
Senza IVA: 3.75
  IVA 20%: 0.75
Con IVA: 4.5
    
```

## Storia 1 - UML



**SoftEng**  
<http://softeng.polito.it>

## Storia 1 - implementazione

**SoftEng**  
<http://softeng.polito.it>

## Storia 1 - test

---

- Listino prodotti:
  - ♦ c'e' prodotto, non c'e' prodotto
- Cassa
  - ♦ sessione vuota
  - ♦ sessione un prodotto
  - ♦ sessione due prodotti
  - ♦ sessione con prodotto mancante

**SoftEng**  
http://softeng.unipi.it

---

## Licensing Note






### Attribution-NonCommercial-NoDerivs 2.5

**You are free:**

- to copy, distribute, display, and perform the work

**Under the following conditions:**

-  **Attribution.** You must attribute the work in the manner specified by the author or licensor.
-  **Noncommercial.** You may not use this work for commercial purposes.
-  **No Derivative Works.** You may not alter, transform, or build upon this work.
  - For any reuse or distribution, you must make clear to others the license terms of this work.
  - Any of these conditions can be waived if you get permission from the copyright holder.

**Your fair use and other rights are in no way affected by the above.**







This is a human-readable summary of the Legal Code (the full license) found at the end of this document

---

# License (1)



THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

## 1. Definitions

-  **"Collective Work"** means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
-  **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
-  **"Licensor"** means the individual or entity that offers the Work under the terms of this License.
-  **"Original Author"** means the individual or entity who created the Work.
-  **"Work"** means the copyrightable work of authorship offered under the terms of this License.
-  **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

**2. Fair Use Rights.** Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.








**3. License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

-  to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
-  to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

# License (2)

**4. Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

-  You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested.
-  You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
-  If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing, (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
-  For the avoidance of doubt, where the Work is a musical composition:
  -  Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.
  -  Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.
-  **Webcasting Rights and Statutory Royalties.** For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

## License (3)


---


### 5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.


**6. Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.


### 7. Termination


 This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.


 Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

### 8. Miscellaneous

 Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

 If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

 No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

 This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.