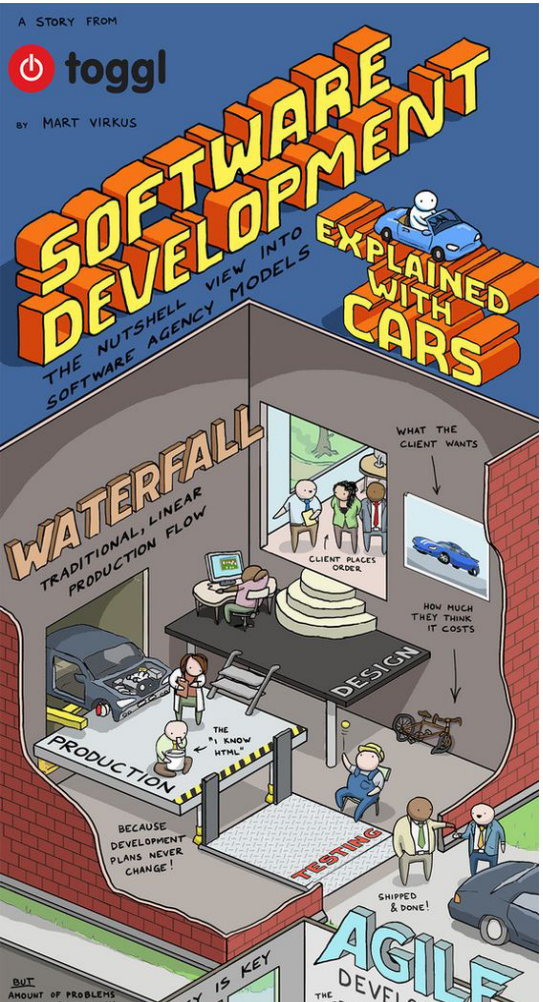
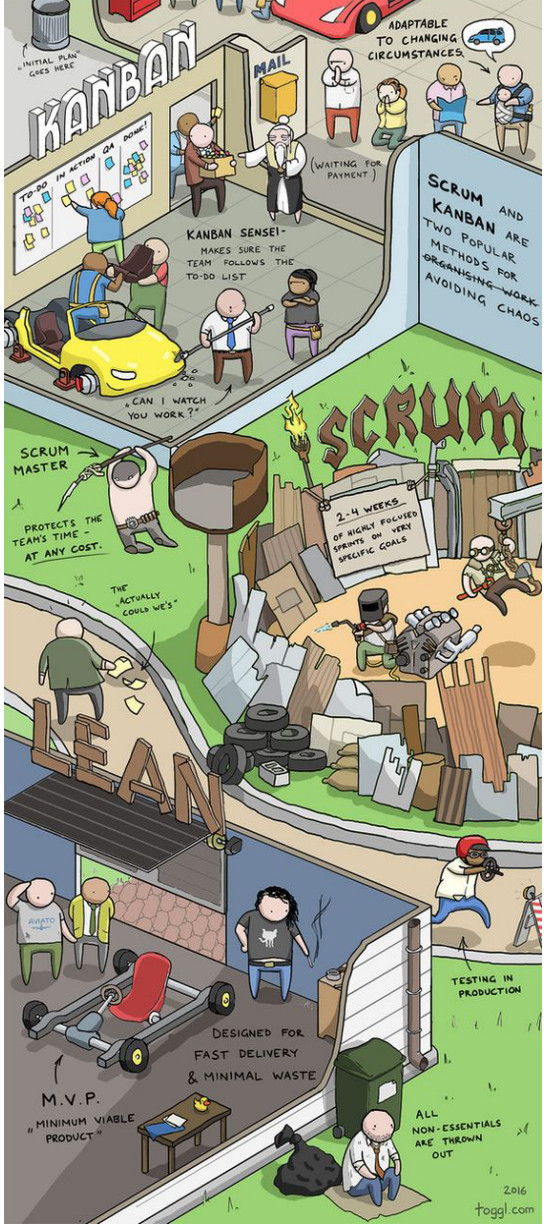
Metodi Agili e Extreme Programming - Overview -

Carlo Tasso

SW & Cars


A STORY FROM
toggl
BY MART VIRKUS
SOFTWARE DEVELOPMENT
THE NUTSHELL VIEW INTO SOFTWARE AGENCY MODELS
EXPLAINED WITH CARS
WATERFALL
TRADITIONAL LINEAR PRODUCTION FLOW
WHAT THE CLIENT WANTS
CLIENT PLACES ORDER
HOW MUCH THEY THINK IT COSTS
DESIGN
PRODUCTION
THE "I KNOW HTML"
BECAUSE DEVELOPMENT PLANS NEVER CHANGE!
TESTING
SHIPPED & DONE!
AGILE
BUT
AMOUNT OF PROBLEMS


SHIPPED & DONE!
AGILE DEVELOPMENT
TRANSPARENCY IS KEY
BUT AMOUNT OF PROBLEMS GROWS EXPONENTIALLY WITH THE SIZE OF THE CLIENT-SIDE TEAM
THE WEIRD LANGUAGES GUY
(COBOL, BASICALLY)
SMALL, INCREMENTAL CHANGES
CLIENT IS INVOLVED IN THE PROCESS
"INITIAL PLAN" GOES HERE
KANBAN
MAIL
ADAPTABLE TO CHANGING CIRCUMSTANCES
TO-DO
IN ACTION
DONE!
(WAITING FOR PAYMENT)
KANBAN SENSEI- MAKES SURE THE TEAM FOLLOWS THE TO-DO LIST
"CAN I WATCH YOU WORK?"
SCRUM AND KANBAN ARE TWO POPULAR METHODS FOR ORGANISING WORK AVOIDING CHAOS
SCRUM
SCRUM MASTER
PROTECTS THE TEAM'S TIME - AT ANY COST.
2-4 WEEKS OF HIGHLY FOCUSED SPRINTS ON VERY SPECIFIC GOALS
THE "ACTUALLY COULD WE'S"
LEAN


"INITIAL PLAN" GOES HERE
KANBAN
MAIL
ADAPTABLE TO CHANGING CIRCUMSTANCES
TO-DO
IN ACTION
DONE!
(WAITING FOR PAYMENT)
KANBAN SENSEI- MAKES SURE THE TEAM FOLLOWS THE TO-DO LIST
"CAN I WATCH YOU WORK?"
SCRUM AND KANBAN ARE TWO POPULAR METHODS FOR ORGANISING WORK AVOIDING CHAOS
SCRUM
SCRUM MASTER
PROTECTS THE TEAM'S TIME - AT ANY COST.
2-4 WEEKS OF HIGHLY FOCUSED SPRINTS ON VERY SPECIFIC GOALS
THE "ACTUALLY COULD WE'S"
LEAN
TESTING IN PRODUCTION
DESIGNED FOR FAST DELIVERY & MINIMAL WASTE
M.V.P.
"MINIMUM VIABLE PRODUCT"
ALL NON-ESSENTIALS ARE THROWN OUT
2016
toggl.com

Agile methods

Perchè l'AGILE?

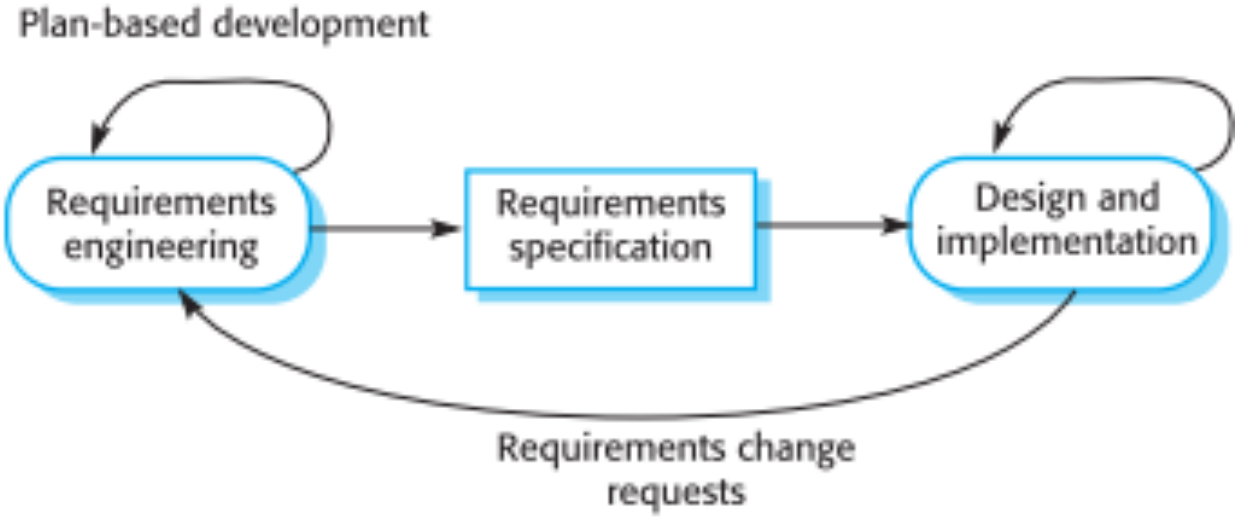
Rapid software development

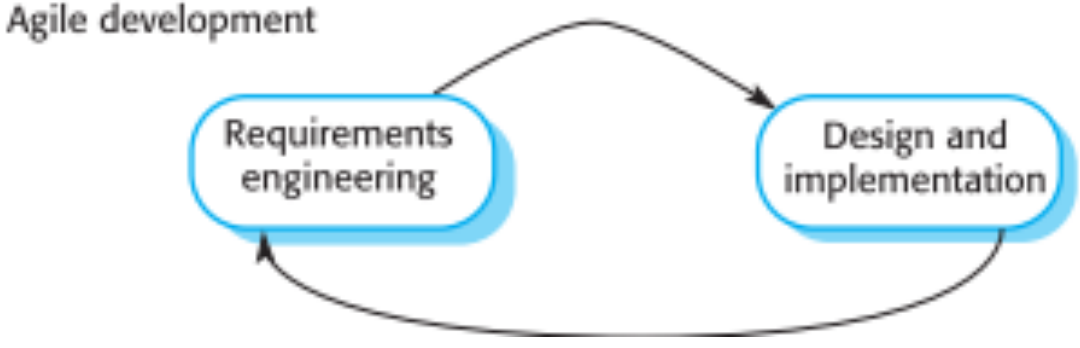
- **Rapid development and delivery** is now often the most important requirement for software systems
  - Businesses operate in a **fast – changing requirement** and it is practically impossible to produce a set of stable software requirements
  - Software has to **evolve quickly** to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the **late 1990s** whose aim was to radically **reduce the delivery time** for working software systems

↳ le tempistiche degli altri metodi erano troppo lunghe

Trovare dei metodi per consegnare un pezzo alla volta

Plan-driven vs. agile development





Plan-driven and agile development

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Idee generali dell'AGILE?

Agile methods

- Especially when dealing with small/medium size projects/tasks, the dissatisfaction with the **overheads**(*) involved in plan-driven design methods led to the creation of agile methods. Agile methods:
  - Focus on the **code** rather than the design: specification, design and implementation are inter-leaved;
  - Are based on an **iterative/incremental** approach to software development: **frequent** new versions are delivered for evaluation;
  - Are intended to **deliver** working software quickly/frequently and **evolve** this quickly to meet **changing** requirements.
  - Are supported by CASE **tools** (e.g. automated testing tools)

Per realtà piccole posso intanto consegnare senza creare da subito documentazioni complesse, ecc.

(*) Overheads: Planning, analysis & design methods, report & documentation writing, formalized quality assurance, discipline,

justified for large long-lived complex projects, sparse/large teams, coordination problems, …

.... heavyweight vs. lightweight software engineering

↳ PLAN-DRIVEN per progetti grandi

↳ AGILE, per piccoli progetti

Agile Manifesto

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software“

[Manifesto for Agile - 2001]

THE AGILE MANIFESTO

AGILE	TRADITIONAL
CUSTOMER COLLABORATION	CONTRACT NEGOTIATION
RESPONDING TO CHANGE	FOLLOWING A PLAN
WORKING SOLUTIONS	DOCUMENTATION
HIGHEST BUSINESS VALUE IN THE SHORTEST TIME	DEFERRED BUSINESS VALUE VIA LONGER DELIVERY TIME

AGILE METHODOLOGY – DEFINITION

Agile methodology promotes an environment of **adaptation**, **teamwork**, **self-organization** and **rapid delivery** that allows for a high level of customer involvement early in project planning.

The Agile Spirit:

un cambiamento di paradigma con uno shift di priorità

- Incremental
  - *It is better to incrementally deliver Working software* over comprehensive documentation
- Cooperation
  - *Customer collaboration* over contract negotiation
- Straightforward
  - *Individuals and interactions* over processes and tools
- Adaptive
  - *Responding to change* over following a plan

Principles of agile methods

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Agile method applicability

- Product development where a software company is developing a **small or medium-sized product** for sale.
  - Virtually all software products and apps are now developed using an agile approach
- **Customized** system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.
- In general, Agile methods are probably best suited to small/medium-sized **business** systems, **PC/Mobile** products

Problems with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Si lavora su piccoli interventi, per cui prioritising changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts (/management) may be a problem as with other approaches to iterative development.

Vari approcci dell'AGILE

Agile Methodologies

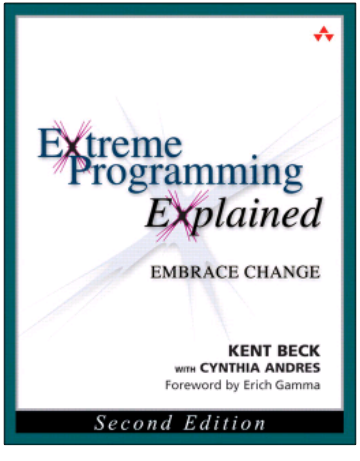
- **eXtreme Programming (XP)**
- **Scrum** (corso IS P&L della LM Informatica) → più diffuso
- Crystal family of methodologies
- Feature-Driven Development (FDD)
- Adaptive Software Development (ASD)
- Dynamic System Development Model (DSDM)
- Agile Unified Process (AUP)
- IBM's Agile Scaling Model (ASM)
- … DevOps …
- …

Extreme programming

The XP Guru: Kent Beck

- eXtreme Programming
  - The most prominent agile development methodology

Kent Beck



1st ed. Oct 1999

2nd ed. Nov 2004

Extreme programming

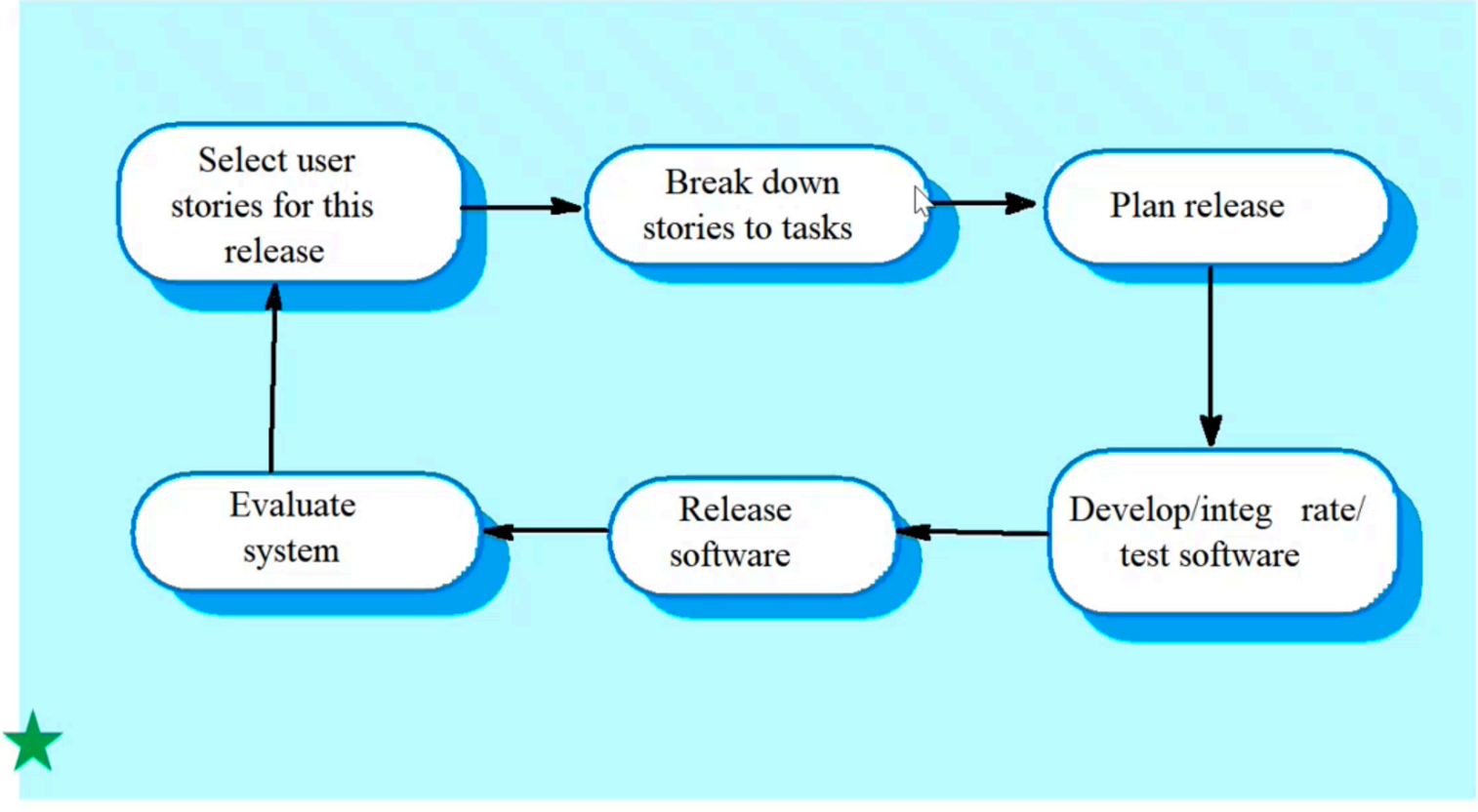
↳ Lavoro per incrementi su un sistema piccolo

- Extreme Programming (XP) takes an 'extreme' approach to iterative* development.
  - **New versions** may be built several times per day;
  - **Increments** are delivered to customers every 2 weeks or less ;
  - All **tests** must be run for every build and the build is only accepted if tests run successfully.

(*) More appropriate to refer to '**incremental/iterative** development'. In altri termini una delle sue caratteristiche fondamentali è un' *estremizzazione dell'incremental development, con incrementi molto piccoli*, rilasciati molto frequentemente e con una pianificazione collaborativa e *negoziata (con il cliente)* che procede incrementalmente lungo il ciclo di sviluppo

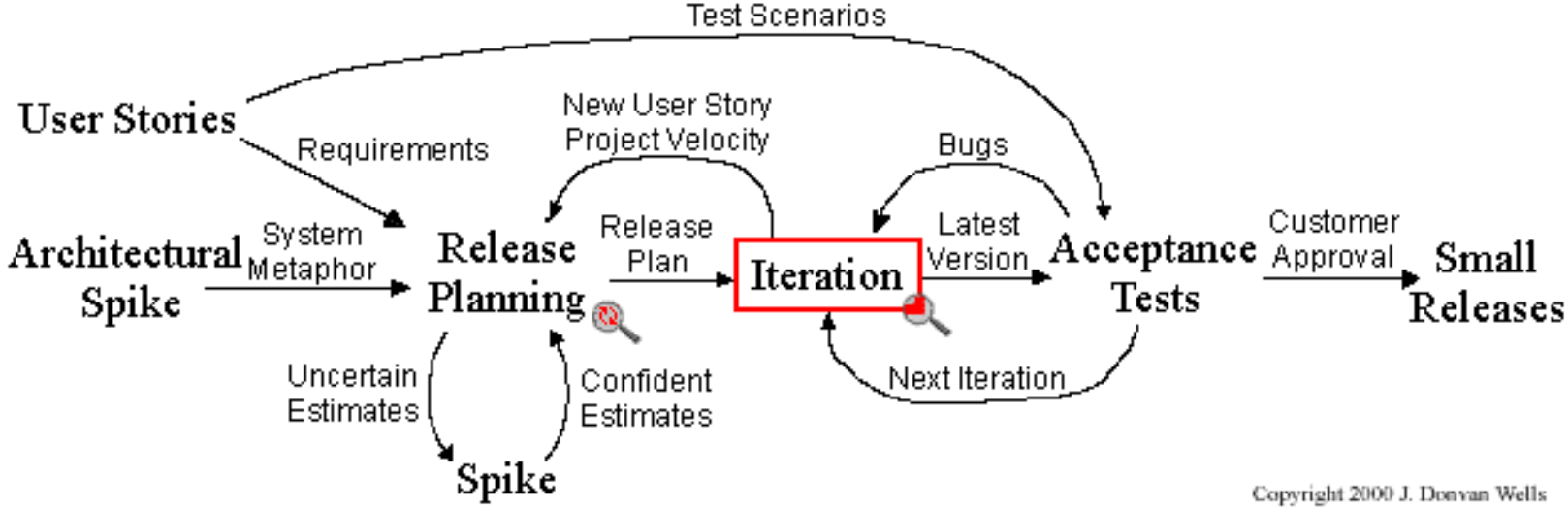
The XP release cycle:

User stories → tasks →develop → release → evaluate → …

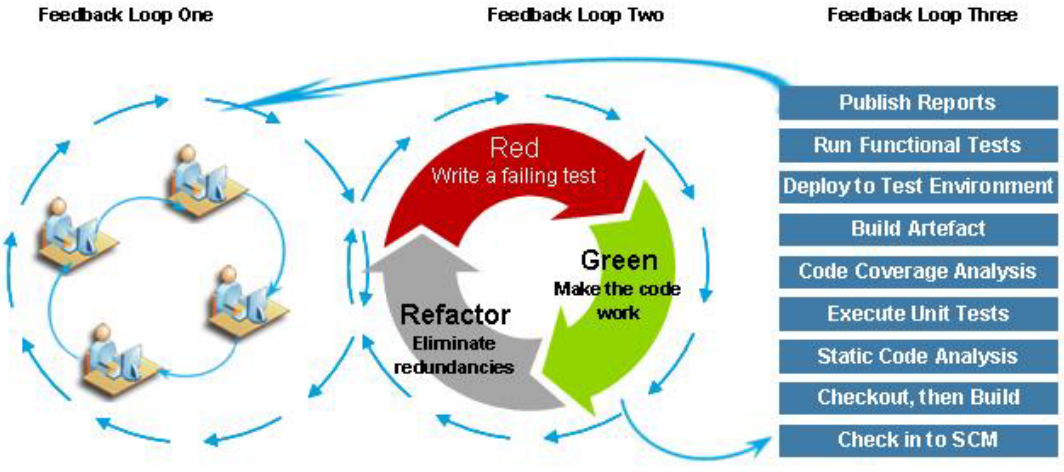


More detailed workflows

Extreme Programming Project



General Xtreme Programming Framework



Pair Programming

- Collective Code Ownership
- Knowledge transfer across the whole team
- Increased discipline and better time management
- Less interruptions, better code

Test Driven Development

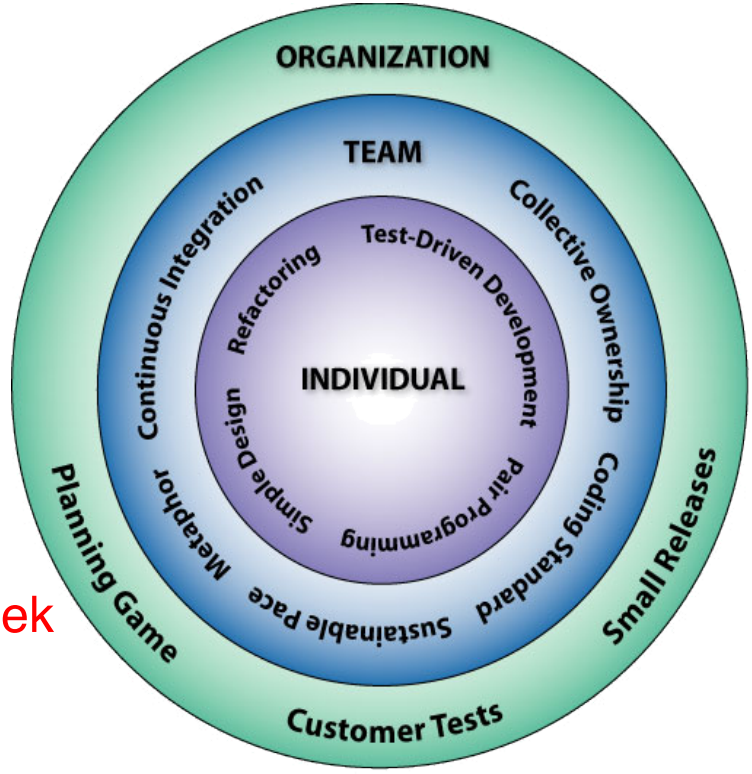
- Increase code Quality, reduce risk
- Encourage innovative solutions, through test validation
- Only code necessary to satisfy a test is written
- Better Design decisions are made

Continuous Integration

- Continuous Feedback on the project status and build quality
- Reduced integration risk and incremental regression testing
- Reduced time to market with business critical features

The 12 Key Practices

- Incremental Planning
- Small Releases
- Metaphor (stories)
- Simple Design
- Test-Driven Development
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- Sustainability: 40-Hour Workweek
- On-site Customer
- Precise coding Standards



Extreme programming practices 1

Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development ÔTasksÕ
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices 2

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code. (**NB.** Semplicità delle soluzioni tecniche! NO banalizzazione dei requisiti!!)

Requirements scenarios - **User Stories**

- In XP, user requirements are expressed as scenarios or user stories. Stories are expressed through the terms of the domain (… metaphor).
- A story is the description of a user-system interaction aimed at some specific goal
- These are written on **cards** and the development team breaks them down into implementation tasks. These tasks are the basis of schedule and cost estimates. (vedi anche CRC card nell'OOP)
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

User Stories

Customer Story and Task Card B/W Development \ COLA

DATE: 3/19/98 TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST ___

STORY NUMBER: [redacted] 1275 PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs in the middle of the B/W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:
For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExcessCOLA BIN

Date	Status	To Do	Comments

Story card for ‘document downloading’

Downloading and printing an article

First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer

You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.

If the article is a print-only article, you can keep the PDF version so it is automatically deleted from your computer

Stories v/s Use Cases

descrizioni a parole

- A story is not a use case, a use case is not a story
- A use case defines functional requirements in total
- Set of use cases define breadth and depth of full system behaviour, augmented with non-functional requirements
- A **story points to a small portion** of breadth and depth
- Stories incrementally build up full scope
- Usually **2-3 days** of work
- Further broken down into tasks

The old way…

Some tasks for ‘document downloading’

Task: singoli passaggi

Downloading and printing an article

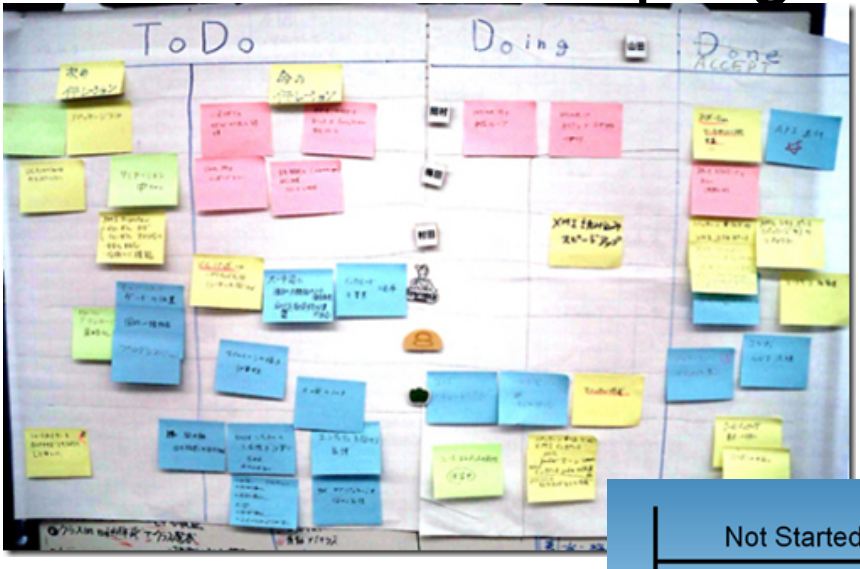
First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

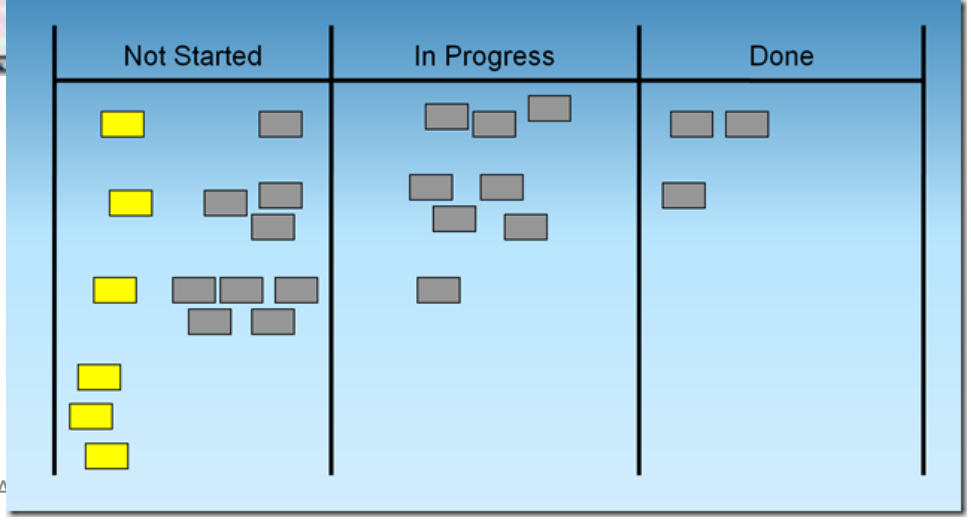
After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer

You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.

If the article is a print-only article, you can keep the PDF version so it is automatically deleted from your computer

Board with user stories, tasks, features, progress



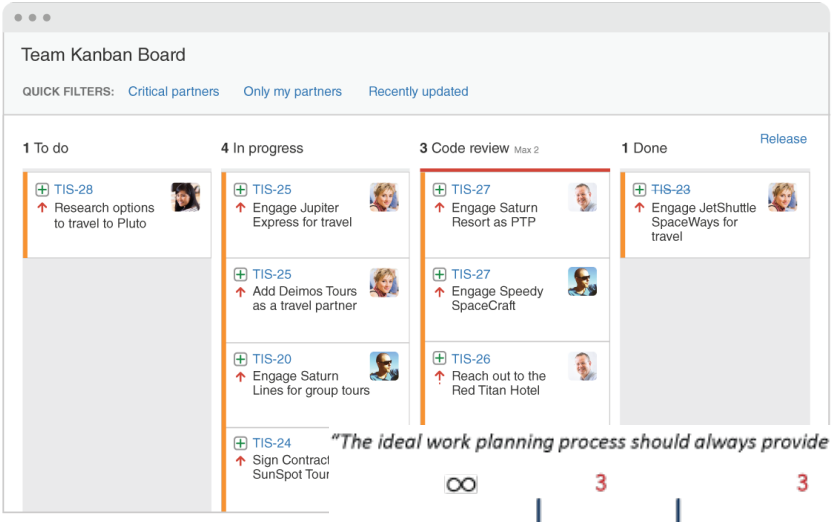


Kanban (development)

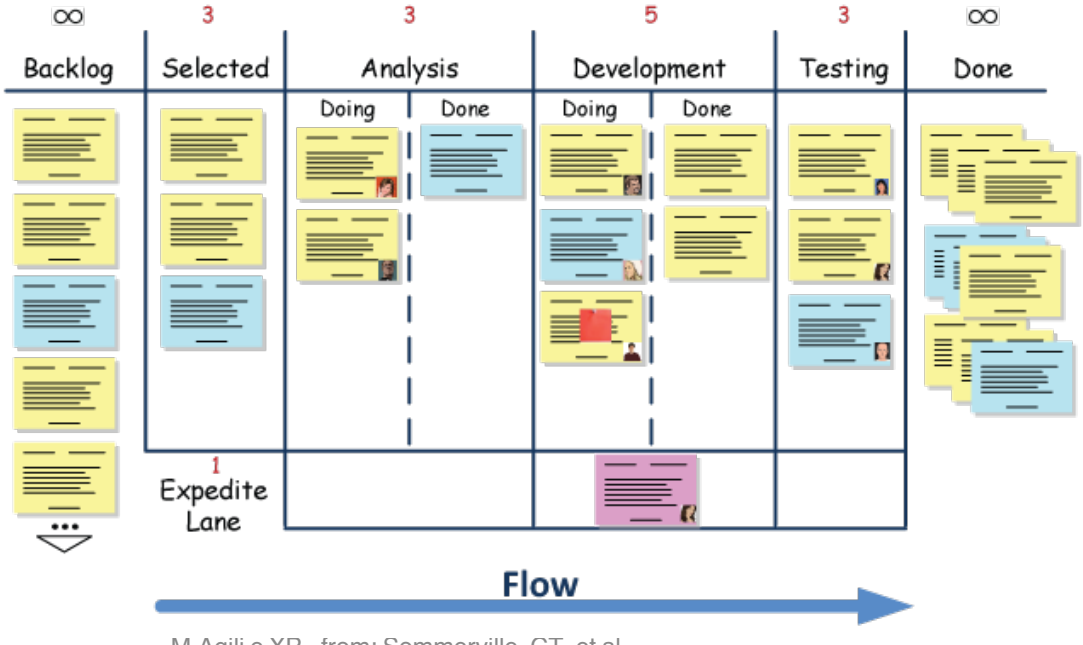
(corso IS P&L LM Informatica)

Kanban is a method for managing knowledge work which balances the demand for work to be done with the available capacity to start new work. Intangible work items are visualized to present all participants with a view of the progress of individual items, and the process from task definition to customer delivery. Team members "pull" work as they have capacity, rather than work being "pushed" into the process when requested.

Kanban in the context of software development provides a visual process-management system that aids decision-making concerning what to produce, when to produce it, and how much to produce. Although the method originated in software development and IT projects, the method is more general in that it can be applied to any professional service, where the outcome of the work is intangible rather than physical.
[da Wikipedia]



"The ideal work planning process should always provide the development team with best thing to work on next, no more and no less."



XP and change

Ogni volta che hai una specifica cerca di perderci il minor tempo per svilupparla in modo da dare l'output subito all'utente

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort **anticipating** changes as this reduces costs later in the life cycle.
- XP, however, claims that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (**refactoring**) to make changes easier when they have to be implemented.

Refactoring

Prima faccio una versione più "scarna e disordinata" per far prima e consegnare un output che FUNZIONA poi faccio refactoring e lo sistemo internamente

- Il **refactoring** indica il processo di modifica della struttura interna di un programma eseguito senza modificarne il comportamento funzionale esterno o le funzionalità esistenti. Tipicamente, il refactoring viene applicato al fine di migliorare le proprietà non funzionali del software, quali la leggibilità e la struttura del codice, la sua aderenza al paradigma di programmazione, il suo grado di manutenibilità, la sua estensibilità, le prestazioni, e così via.

Refactoring

- Improve the design of existing code without changing its functionality
  - Relies on unit testing to ensure the code is not broken
- **Bad smells** in code (**segnali** che suggeriscono la necessità di far refactoring!!):
  - Long method / class
  - Duplicate code
  - Methods does several different things (bad cohesion)
  - Too much dependencies (bad coupling)
  - Complex / unreadable code

How Refactoring Works ?

- Delivering working software faster is important!
  - You can write the code to run somehow
    - With **simple design**
    - With less effort
  - Later you can refactor the code if necessary
- **Refactoring is not a reason to intentionally write bad code!**
  - **Good coding style is always important!**
- Richiede bravi programmatori

When refactoring and its benefits

- Programming teams look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are then easier to make because the code is better-structured and clearer.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Simple Design

- No Big Design
  - Reduces the overhead
  - Ship working functionality faster and get feedback early
- "Do The Simplest Thing That Could Possibly Work"
  - Later use refactoring to change it
- Not too much formal documentation

How It Works ?

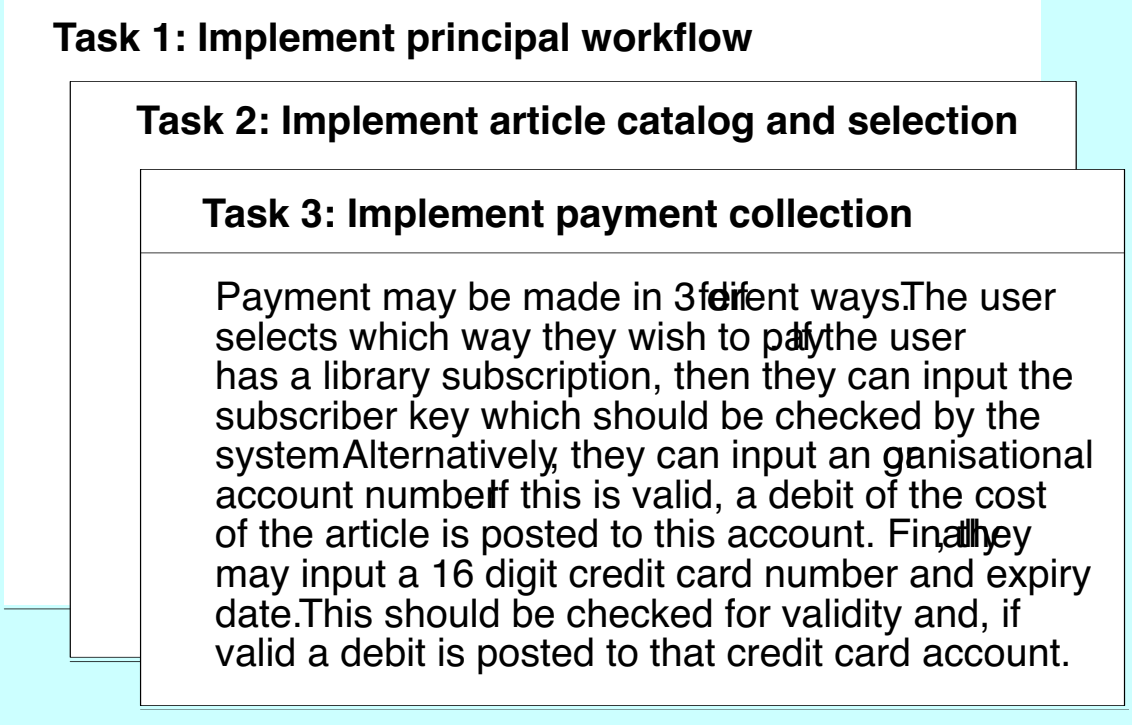
- **Simple design does not mean "no design"**
  - It is about establishing priorities
  - It's a set of tradeoffs you make
  - If something is important for this release and for the whole system, it should be designed well
  - Don't lose time to design something you will not use soon (has a low priority)!

Testing in XP

- Test-first development (Test-driven development)

- Incremental test development from user stories
- User involvement in test development and validation.
- Automated test packages (e.g. Junit) are used to run all component tests each time that a new release is built. (***regression*** *testing*).

Task cards for document downloading



Test case description

Test 4: Test credit card validity

Input:
A string representing the credit card number and two integers rep
the month and year when the card expires
Tests:
Check that all bytes in the string are digits
Check that the month lies between 1 and 12 and the
year is greater than or equal to the current year
Using the first 4 digits of the credit card number
check that the card issuer is valid by looking up the
card issuer table. Check credit card validity by submitting the car
number and expiry date information to the card
issuer
Output:
OK or error message indicating that the card is invalid

Test-first development

- Designing tests **before** code clarifies the requirements to be implemented. Tests are your design spec and fine grained requirements all in one.
- (Test driven development:) Tests are written as programs rather than data so that they can be executed automatically, immediately after the code has been written, in order to discover possible problems. The test includes a check that it has executed correctly.
- Si introduce codice specifco per fare gli specifici test relativi al codice x che si sta sviluppando. Si mette in esecuzione tale codice x e viene eseguito automaticamente anche il relativo test.
- All previous (REGERESSION test) and new tests are automatically run when a new functionality is added. Thus checking that the new functionality has not introduced errors.
- **Must** run at 100% before proceeding with integration and release

Test-first development / 2

- Acceptance Tests (Collaudo)
  - Written with the customer
  - Acts as “contract”
  - Measure of progress

Test automation

- Test automation means that tests are written as executable components before the task is implemented
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. **Junit**) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be (re)run and problems that the new code has introduced can be caught immediately. (regression testing)

Customer involvement

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team (egoless programming).
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming **is NOT worse**, as it appears similar to that of two people working independently.

HOW Pair Programming

- Two software engineers work on one task at one computer
  - *The driver* has control of the keyboard and mouse and creates the implementation
  - *The navigator* watches the driver's implementation
    - Identifies defects and participates in on-demand brainstorming
  - The roles of driver and observer are periodically rotated
  - Pairs are frequently dynamically redefined

Pair Programming

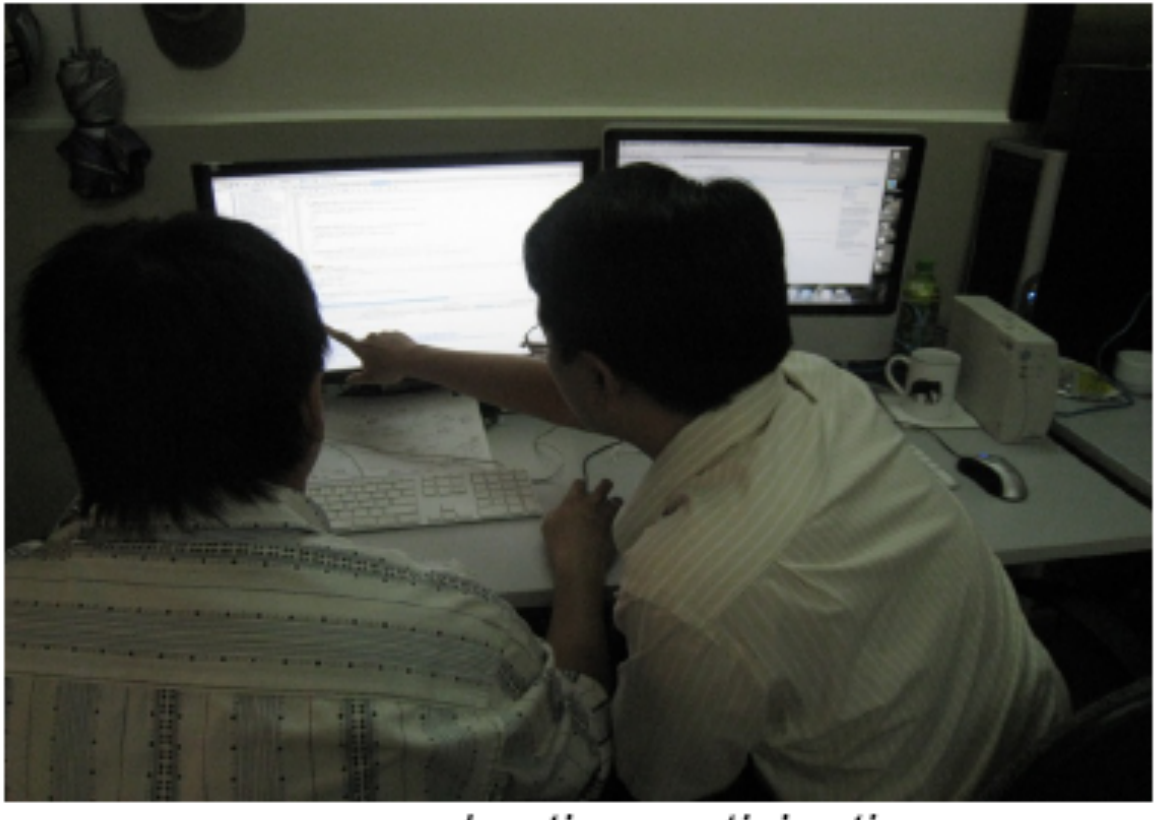
- Pairs produce higher quality code
- Pairs complete their tasks faster
- Pairs enjoy their work more
- Pairs feel more confident in their work

Pair Programming

Pair Programming

Sit Together / Whole Team

Pair Programming



... and active participation

How It Works ?

- Pair programming is great for complex and critical logic
  - When developers need good concentration
  - Where quality is really important
  - Especially during design
  - Reduces time wasting, e.g. ICQ chatting
- Trivial tasks can be done alone

Collective Code Ownership

- Code belongs to the project, not to an individual engineer!
- Any engineer can modify any code
- Better quality of the code
- Engineers are not required to work around deficiencies in code they do not own
  - Faster progress
  - No need to wait for someone else to fix something
- Respect for other members and for the Project

How it Works ?

- Collective code ownership is absolutely indispensable
  - You need to fight the people who don't agree with this!
  - Fire people writing unreadable and unmaintainable code
  - Don't allow somebody to own some module and be irreplaceable

★Continuous Integration

- Pair writes up unit test cases and code for a task (part of a user story)
- Pair unit tests code to 100%
- Pair integrates
- Pair runs ALL unit test cases to 100%
- Pair moves on to next task
- Should happen **once or twice a day**

On-Site Customer

- Customer available on site
  - Clarify user stories
  - Make critical business decisions, prioritize
- Developers don't need to guess/make assumptions
- Developers don't have to wait for decisions
- Face to face communication minimizes the chances of misunderstanding

The old way


I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING !

On-Site Customer

Communicate via phone ...

★Small Releases

- Timeboxed
- As small as possible, but still delivering business (ECONOMIC) value
  - No releases to ‘implement the database’
- Get customer **feedback** early and often
- Do the planning after each iteration
  - Do they want something different?
  - Have their priorities changed?

How It Works ?

- Small releases are really valuable
  - Manage the risk of delivering something wrong
  - Helps the customer to define better requirements
- **Release every few weeks or less**

Forty-Hour Work Week

- Small and frequent releases
- Kent Beck says, " . . . fresh and eager every morning, and tired and satisfied every night"
- Burning the midnight oil kills performance
- Tired developers make more mistakes
  - Slows you down more in the long run
- If you mess with people's personal lives (by taking it over), in the long run the project will pay the consequences

Coding Standards

- Use coding conventions
  - Rules for naming, formatting, etc.
  - Write readable and maintainable code
- Method commenting
  - Self-documenting code
  - Don't comment bad code, rewrite it! **Avere il coraggio di buttare via il codice mal fatto!**
- Refactor to improve the design
- Use code audit tools (FxCop, CheckStyle, TFS)

How It Works ?

- Coding standards are important
  - Enforce good practices to whole the team – tools, code reviews, etc.
- Standards should be simple
  - Complex standards are not followed
  - Standards should be more strict for larger teams
  - Developers don't like utter rules like "*comment any class member*"

The 13th Practice?
The Stand Up Meeting

- Start the day with 15/10-minute meeting
  - Everyone stands up (so the meeting stays short) in circle
  - Going around the room everyone says specifically:
    - What they did the day before
    - What they plan to do today
    - Any obstacles they are experiencing
  - Can be the way pairs are formed
- Far girare le informazioni, comunicare, non lavorare in isolamento, condividere i problemi e le soluzioni

People Communicate Most Effectively Face-to-Face

Stand-up meeting

How XP Solve Some SE Problems

Problem	Solution
Slipped schedule	Short development cycles
Cancelled project	Intensive customer presence
Cost of changes	Extensive, ongoing testing, system always running
Defect rates	Unit tests, customer tests
Misunderstand the business	Customer part of the team
Business changes	Changes are welcome
Staff turnover	Intensive teamwork

★ Agile development critical issues

si fa fatica a capire a che punto si è. I tempi sono variabili. È adatto per piccole realtà

- Agile **project management** requires an approach, which is adapted to incremental development and the practices used in agile methods. The standard approach is not very adequate in the agile scenario
- New **contractual** approaches have to be introduced, since a standard contract that pays for developer time rather than functionality.
- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team. They do not **scale up** easily to complex projects with multiple stakholders and teams. They need to be extended to large system development.
- Lack of product documentation, keeping customers involved in the development process, maintaining the continuity of the development team are all issues against **maintaining** the sw system by means of the agile approach! And for long-lifetime systems, this is a real problem as the original developers will not always work on the system.

Set 15 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- Origine e motivazione dei metodi Agili. Plan driven vs. agile. Agile Manifesto. Principi e priorità generali. Applicabilità. Criticità.
- Extreme Programming (XP). Idea base. Organizzazione ciclica generale del processo di sviluppo. Aspetti chiave dell'XP. User stories: caratteristiche, diffrenze da use case, scomposizione in task, board per esporre storie e task e stato di avanzamento, cenno al metodo Kanban. Requirement change e Refactoring (R.): a cosa serve il R., quando è opportuno eseguirlo e come sfruttarlo, con che vantaggi. Semplicità nelle progettazione e Regression. Testing: test driven development, test-first development, importanza del regression testing (al 100%), collaudo, automazione del testing, criticità dell'approccio, coinvolgimento dei client nel testing, Pair Programming (PP), caratteristiche, modalità, quando ha più senso. Proprietà 'collettiva' del codice, cosa significa, come funziona. Integrazione continua e PP. Presenza del cliente. Piccole release (incrementi) frequenti. 40h alla settimana. Standard di codifica, Meeting giornaliero, in piedi. Criticità dell'approccio Agile: management, contrattualistica, manutenzione.

Fine delle LEZIONI
della Parte del prof. C. TASSO

Grazie dell'attenzione!