Una didattica Agile per la programmazione

Marcello Missiroli, Daniel Russo[1], Paolo Ciancarini[2]
DIEF, Università di Modena e Reggio Emilia
Via Vivarelli 10, 41125, Modena
marcello.missiroli@unimore.it
[1]DISI, Università di Bologna
Mura Anteo Zamboni 7, 40127. Bologna
daniel.russo@unibo.it
[2]DISI, Università di Bologna
Mura Anteo Zamboni 7, 40127. Bologna
paolo.ciancarini@unibo.it

Un esperimento in classe dimostra come l'utilizzo di metodologie agili apportino un significativo miglioramento sulla didattica dell'informatica nel triennio delle scuole superiori.

1. Introduzione

Negli ultimi anni imparare a programmare è diventato molto più facile e interessante. La comparsa di ambienti di sviluppo divertenti e coinvolgenti (primo fra tutti il noto **Scratch**), iniziative quali la **Hour of Code** e possibilità di autoapprendimento online (come **Codeacademy**) stanno lentamente cambiando il modo di imparare e di praticare l'arte della programmazione, anche nel grande pubblico.

Tuttavia, quando si passa all'insegnamento degli aspetti più formali della programmazione, specie nelle scuole superiori e nelle università, è facile notare che durante gli ultimi decenni ben poco è cambiato nelle modalità di insegnamento, tanto in Italia quanto nel resto del mondo. Nonostante i progressi della didattica, il pattern di insegnamento tuttora più utilizzato è quello tradizionale: lezione frontale/esercizi/verifica individuale. In particolare, la programmazione è vista come un'attività prettamente individuale, e il lavoro di gruppo è o forzato dalle circostanze (per esempio, dovuto al numero di workstation a disposizione) o è limitato all'esecuzione di un progetto conclusivo che si svolge con le fasi classiche di analisi-design-codifica-testing.

Sappiamo che negli ultimi anni in certi ambienti del mondo del lavoro si sta diffondendo il modello di sviluppo Agile. Diversi studi (il più noto è il CHAOS report[1]) mostrano come in certi domini il nuovo modello di sviluppo offra un sensibile miglioramento dei risultati in termini di produttività e qualità dei prodotti software.

Tuttavia, tali metodi sono raramente insegnati nelle aule universitarie (ed ancor meno nelle scuole superiori) – in alcuni casi l'Agile è perfino "proibito". Tutto questo è ancor più strano se si tiene conto di una notevole condivisione di

DIDAMATICA 2016, ISBN: 9788898091447

valori tra elementi della progettazione agile e quelli della moderna didattica: priorità al lavoro di gruppo, responsabilità condivisa, attenta gestione delle risorse, ecc.

Ci siamo dunque chiesti se l'implementazione di alcune attività tipiche dei modelli agili potesse portare a qualche beneficio nella didattica dell'informatica. Almeno due sono gli aspetti da approfondire. Il primo è quello relativo alle cosiddette "Pratiche Agili" [2], una serie di tecniche che rendono più efficace il lavoro quotidiano di codifica e sviluppo. Tra esse, una delle più note è il **Pair Programming**, ma possiamo citarne diverse altre, come il **Timeboxing** e il **Refactoring**.

Il secondo aspetto riguarda la metodologia generale dello sviluppo del software. Tipicamente, ogni studente o gruppo di studenti realizza almeno un software di media complessità, che spesso è il prodotto conclusivo dell'anno di corso. Per realizzarlo i ragazzi usano una metodologia più o meno ispirata dal modello a cascata tradizionale (Waterfall): analisi, progettazione, codifica, test. In realtà, ben pochi seguono *davvero* questo modello; molto spesso le allieve e gli allievi tendono a soluzioni più dirette e improvvisate, note come "*cowboy programming*": prima scrivere il codice, poi sistemare le cose più 'noiose', come ad esempio la documentazione, e sperare che tutto vada bene.

Le metodologie agili, come ad esempio **Scrum**, utilizzano diverse strategie volte a minimizzare i passaggi burocratici e favorire l'interazione interna ed esterna, alternando frequentemente le fasi di progettazione e codifica; diversi studi affermano che in questo modo lo sviluppo risulta più efficiente, proattivo e, perché no, piacevole.

Tutto questo si riferisce al mondo del lavoro, specialmente alle realtà imprenditoriali medio-piccole. Ma non potrebbero forse alcuni di questi aspetti avere riflessi positivi anche nel mondo della scuola? Si può insegnare a programmare in un modo diverso, più sociale, più responsabile, e in fin dei conti più creativo? In fondo, già esistono diversi casi in cui le metodologie agili sono applicate con successo nella scuola in generale, seppur poco note e diffuse (vedere il sistema **Eduscrum** [3] e **TTMS** [4]) riconducibili sotto il cappello del **Manifesto della Scuola Agile** [5].

Paradossalmente, di esempi concreti e applicativi dei metodi agili all'insegnamento dell'*informatica* non vi è traccia.

In più, gli unici studi presenti sono di natura fenomenologica: descrivendo l'esperienza. Per questo motivo abbiamo deciso di utilizzare un approccio sperimentale, andando a misurare l'efficienza e l'efficacia dell'implementazione di metodologie agili rispetto a quelle tradizionali. Pertanto abbiamo utilizzato strumenti tipici dell'**Empirical Software Engineering**, per valutare la significatività statistica di questo nuovo approccio [6].

Questo articolo è una versione ridotta e di stampo divulgativo di una nostra pubblicazione a ICSE 2016 [10]. Non potendo entrare nel dettaglio delle scelte metodologiche adottate, rimandiamo ogni approfondimento all'articolo integrale in lingua inglese.

2. Metodologia

Come è facile intuire, imbastire un esperimento scientifico in una scuola pubblica impone diverse limitazioni. Docenti e studenti non troppo motivati, rigidezze dell'orario scolastico, l'importanza riposta nella sacralità del "programma da svolgere" sono solo alcune delle difficoltà riscontrate.

La nostra scelta è stata quella di realizzare una esperienza di Apprendimento Esperienziale [7] in modo da aumentarne l'appeal sia per i docenti sia per gli studenti coinvolti; per gli stessi motivi, la durata dell'esperimento sulle classi è stato limitato a poche ore (da tre a sei), e in ogni sessione dell'esperimento si sono testate più tecniche. Il target dell'esperimento è costituito dagli ultimi due anni delle scuole in cui si insegna informatica ad un elevato livello: ITIS ad indirizzo informatico, ITC di Sistemi Informativi Aziendali, LS delle scienze applicate. Per il primo esperimento l'ambito geografico è stato limitato alle province di Modena, Reggio Emilia e Parma.

3. Il primo esperimento: pratiche agili

Al primo esperimento hanno preso parte solo tre istituti, l'IIS Corni di Modena, l'IIS Pascal di Reggio Emilia e l'ITC Bodoni di Parma; spicca negativamente, a questo proposito, l'assenza dei licei – secondo i colleghi contattati, con l'arrivo a completamento della riforma Gelmini gli studenti non hanno più le competenze necessarie per realizzare un sito web dinamico, neppure se molto semplice. Complessivamente, all'iniziativa hanno partecipato un'ottantina di ragazzi, ripartiti su quattro classi, come risulta dalla Tabella 1.

Tabella 1 – Soggetti del primo esperimento

Soggetti	**Complessivi**			**Pair Programmers**						**Solo**		
	Totale	PP	SP	BB	BM	BS	MM	MS	SS	B	M	S
Classe IV	61	50	11	8	10	6	8	10	8	4	3	3
Classe V	23	18	5	4	4	4	2	0	4	2	1	2
Test1	44	36	8	6	8	4	6	8	6	3	2	2
Test2	40	32	8	6	6	6	4	2	6	3	2	3

Dato che una delle tecniche che volevamo testare era il Pair Programming [2] , volevamo verificarne l'impatto su tutte le tipologie di studenti. Ogni classe è stata così ripartita in tre gruppi di merito secondo la loro corrente valutazione scolastica (Buoni, Medi, Scarsi); abbiamo quindi costruito in modo casuale coppie in tutte le sei possibili combinazioni di gruppi di merito (Buoni-Buoni → B-B, Buoni-Medi → B-M, ecc.) In aggiunta, alcuni ragazzi hanno programmato da soli ("Solo Programming"), in modo da fungere da gruppo di controllo, sempre mantenendo distinta la tipologia di merito.

Una prima versione dell'esperimento (Test1, versione "breve") è stata studiata per i ragazzi di 4°, incentrata sul Test-Driven Development [2]. Si tratta un'adattamento del famoso Tennis Game Kata[8], che chiede di sviluppare una classe Java in grado di convertire il punteggio del tennis in un formato leggibile da esseri umani. In una prima fase formativa ai ragazzi è stato mostrato il funzionamento dei test automatici (Junit) e la "filosofia" del TFD -TDD e del Refactoring. È chiaro che non ci si può aspettare che i ragazzi inizino a scrivere batterie di test come per magia, per cui i test dell'esperienza sono stati scritti da noi.

Durante la fase operativa (rigidamente limitata a due ore) ad ogni gruppo o individuo è stato proposto un test. Solo dopo che il test era diventato "verde", per cui il codice corrispondente era stato scritto correttamente, si consegnava al gruppo il test successivo. Di tanto in tanto, i ragazzi erano costretti a pause forzate per rifattorizzare il codice. Ogni dieci minuti, si obbligava il cambio di ruolo tra le coppie, come stabilito dal Pair Programming.

La seconda versione (Test2, "lunga") richiedeva competenze superiori, ed era quindi concepita per le quinte. Si chiedeva di realizzare un sito di microblogging, con tanto di login, amministrazione e realizzazione su un vero servizio di hosting esterno. Agli studenti è stato fornito un gruppo di User Story (anche qui, scritte da noi) ed è stato loro detto di sceglierne una e realizzarla. Al termine, il gruppo doveva dimostrare che la funzionalità era stata realizzata sul sito di produzione, e solo in quel caso potevano procedere alla storia successiva. Come nel caso precedente, si è proceduto ad alternare i ruoli tra le coppie e a richiedere un feedback conclusivo.

Dato che volevamo controllare se la nuova modalità di sviluppo modificava il modo con cui i ragazzi scrivevano il codice, abbiamo chiesto ed ottenuto dai docenti una serie di programmi simili per dimensioni, tempistiche ed argomenti a quello realizzato. Abbiamo quindi eseguito una serie di analisi incrociate sul codice, cercando differenze sostanziali in alcuni indicatori fondamentali, come la complessità ciclomatica, quantità di codice e commenti prodotti e così via.

4. Analisi dei dati

L'esperienza ha prodotto risultati piuttosto interessanti. Per prima cosa ci siamo concentrati sulla performance degli allievi, quello che in un esperienza didattica si tradurrebbe in un voto – cioè quello che, in fondo, importa di più a tutti gli attori coinvolti. I risultati sono riassunti nella Tabella 2.

Tabella 2 – Tabella delle prestazioni del primo esperimento

Gruppo	M-M		M-S		B-M		B-B		B-S		S-S		Solo	
	VO	VA	VO	VA	VO	VA	VO	VA	VO	VA	VO	VA	VO	VA
Media	4	6	5	5,5	8,14	6,5	7,5	7	7,4	6	5	5	5,7	6,2
Varianza	3,5	0	5,5	0	2,48	0	4,3	0	0,8	0	4,5	0	6,8	1,2

VO = Voto ottenuto; VA = Voto atteso; M-M = Medio-Medio, M-S = Medio-Scarso, ecc..

Se ad un primo sguardo i risultati delle coppie e dei singoli (gruppo di controllo) sono sostanzialmente identici, le cose cambiano parecchio considerando le diverse tipologie di coppie. In particolare, le coppie formate da elementi eterogenei hanno conseguito, in generale, risultati migliori del previsto, mentre per quelle omogenee vale il contrario. Riteniamo che la pratica del Pair Programming, che promuove l'interazione e la condivisione delle conoscenze, sia particolarmente efficace quando uno dei due componenti sia – anche a livello inconscio – più autorevole dell'altro, realizzando una sorta di "insegnamento tra pari". Laddove manchi questo elemento, molti vantaggi del PP vanno persi, ed alcune coppie si perdono anche in futili litigi (osservati in diversi casi). Il caso più significativo è dato alle coppe Medio-Medio, che hanno ottenuto lo scarto peggiore rispetto alle aspettative; ma anche i gruppi formati da due ragazzi potenzialmente bravi hanno, in generale, deluso.

L'analisi statica del codice ha prodotto risultati altrettanto interessanti. Le tabelle 3A e 3B ne riassumono i risultati, confrontati con codice realizzato dagli stessi ragazzi in precedenti esperienze di tipo simile. I dati mostrati sono la Complessità Computazionale (CC), linee di commento (C), linee duplicate (LD), Problemi riscontrati (PRB), linee di codice, commenti esclusi (NCLOC) e l'indice SQALE [9], un indice generale della qualità del codice.

Nel caso dell'esperienza più lunga si nota un lieve vantaggio dei metodi agili sulla qualità del codice prodotto, in particolare considerando il numero di righe duplicate e il numero totale di problematiche. Nella verifica più corta si è notato, al contrario, un peggioramento generalizzato della qualità, in particolare la complessità ciclomatica e la leggibilità complessiva.

La nostra interpretazione è che i tempi di sviluppo così ristretti e rigidi hanno spinto i ragazzi a trovare le soluzioni più sbrigative al problema, ma forse

lontane dalle quelle ottimali. Anche se il refactoring era previsto, gli studenti non avevano nessun reale incentivo a realizzare “buon” codice, dato che l'unico obiettivo dichiarato era quello di passare il test. Evidentemente, per valutare la qualità del codice occorre un esperimento ad hoc.

Tabella 3A – Analisi del codice (Test breve - Java)

Gruppo	CC	DIFF	C	DIFF	LD	DIFF	PRB	DIFF	NCLOC	DIFF	SQALE
B-B	22,5	181%	10,8	169%	0,0	-100%	16,0	140%	141,3	275%	Meglio
B-M	21,0	163%	8,2	85%	13,8	25%	7,5	1%	169,0	158%	Uguale
B-S	25,0	213%	21,5	1333%	0,0	/	12,5	150%	59,0	81%	Peggio
M-M	22,5	181%	1,0	-69%	0,0	-100%	8,7	28%	88,0	130%	Meglio
M-S	18,8	134%	1,8	-56%	0,0	/	6,0	20%	81,5	110%	Uguale
S-S	20,0	150%	3,5	250%	11,3	/	7,3	-22%	67,7	122%	Peggio
B (Solo)	15,8	96%	30,3	278%	0,0	-100%	9,7	20%	84,3	122%	Peggio
M (Solo)	9,0	13%	4,0	300%	0,0	-100%	12,5	150%	41,5	-16%	Molto Peggio
S (Solo)	16,0	100%	5,0	0%	8,0	-56%	5,7	-51%	44,0	-1%	Peggio
MEDIA	18,9	137%	9,5	254%	3,7	/	9,5	49%	86,3	109%	Peggio

Tabella 3B – Analisi del codice (Test lungo - PHP)

Gruppo	CC	DIFF	C	DIFF	LD	DIFF	PRB	DIFF	NCLOC	DIFF	SQALE
B-B	2,5	23%	0,1	-67%	30,3	-70%	39,0	-56%	188,0	7%	Uguale
B-M	3,5	69%	0,2	51%	90,3	-9%	47,7	-23%	160,7	-32%	Peggio
B-S	2,6	25%	1,1	342%	159,0	20%	88,3	-2%	256,3	15%	Meglio
M-M	0,9	-55%	0,0	/	14,3	-66%	3,8	-95%	8,9	-93%	Uguale
M-S	0,7	-65%	0,2	-85%	133,0	83%	35,0	-16%	30,0	-46%	Meglio
S-S	2,2	8%	0,0	/	76,5	-2%	35,5	-63%	74,5	-60%	Molto Peggio
B (Solo)	5,0	143%	0,0	/	67,5	-27%	83,5	258%	323,5	187%	Uguale
M (Solo)	0,7	-65%	0,2	0%	29,0	-43%	21,5	-5%	16,0	-75%	Molto Peggio
S (Solo)	1,7	-19%	0,0	/	143,0	240%	46,7	419%	181,0	298%	Peggio

MEDIA	2,2	7%	0,2	4%	82,6	14%	44,6	46%	137,7	22%	Peggio

I dati grezzi e i codici sono fin d'ora disponibili per la consultazione [11]. Sono altresì disponibili un paio di video che documentano una delle esperienze [12].

Per verificare l'esattezza delle nostre ipotesi, stiamo eseguendo un esperimento confermativo su classi diverse, limitato per necessità alle sole classi quarte. L'esperimento è molto simile al precedente: le uniche differenze consistono nel fare a meno del gruppo di controllo e lievi modifiche alla formulazione dei test, allo scopo di rendere la classe Java più flessibile e premiare i ragazzi in grado di compiere un refactoring efficace.

Tale esperimento è ancora in svolgimento, ma i primi risultati confermano sostanzialmente sia il miglioramento delle prestazioni nelle coppie eterogenee, sia il peggioramento della qualità del codice realizzato.

5 Le opinioni: studenti e docenti

Di grande interesse sono i commenti ricevuti. In particolare, il Pair Programming ha riscosso grande successo tra gli studenti: lo considerano una buona idea, nonché un modo per imparare qualcosa di nuovo; le coppie hanno in generale funzionato bene, cosa certamente non scontata trattandosi di gruppi semicasuali. L'opinione è confermata anche dai docenti coinvolti, che hanno indicato un generale miglioramento del clima di classe, la condivisione delle informazioni e la facilità di implementazione della tecnica stessa.

Le altre pratiche agili, per la verità, non hanno convinto più di tanto il nostro campione; in particolare, il timeboxing è risultato il metodo più osteggiato, forse per lo stress che imponeva. Solo le User Stories ottengono un giudizio appena accettabile.

Una nuova pratica didattica non può prescindere dai docenti che dovranno metterla in atto. Per questo motivo abbiamo pensato di proporre un sondaggio aperto a tutti i docenti delle province interessate relativo alla possibile introduzione di metodologie agili. Il compito si è rivelato più arduo del previsto, visto che non esiste un “elenco pubblico dei docenti di informatica”: in mancanza di questo, abbiamo utilizzato mezzi diretti (lettere ai dirigenti scolastici, mailing list, telefonate), riuscendo a coprire soltanto un 20% del target previsto. Tuttavia, i dati sono significativi in quanto le risposte dei docenti risultano molto uniformi.

Anche qui, la pratica più positiva risulta quella del Pair Programming. Citando un docente: “*Tutti erano contenti del Pair Programming. In realtà,*

abbiamo avuto qualche protesta dagli studenti selezionati come gruppo di controllo".

I docenti riconoscono il valore didattico del test n. 2 (sito web, più lungo), ma sono un po' incerti sull'utilità del test più breve. L'opinione sulle altre pratiche è ambivalente, anche se i più ritengono che il TDD abbia una cattiva influenza sulla performance dei ragazzi, come pure il timeboxing, seppur in misura minore. Le User Stories e le condizioni di accettazione sono andate meglio, pur con qualche riserva.

Di rilevo questa osservazione: "*Alcuni individui non erano per nulla interessati all'esperimento, e altri non gradivano che le indicazioni dei task fossero così precise e immutabili. L'impegno non è quindi stato del 100%, e i risultati lo dimostrano*"

Ciononostante, se analizziamo le opinioni sull'effetto generale dell'esperimento sulla classe, i risultati sono un po' sorprendenti. Percepiscono un generale effetto benefico sul clima e partecipazione della classe, nonché un piccolo miglioramento dei voti conclusivi, indipendentemente dal livello di abilità o l'aver preso parte al gruppo di controllo o meno.

Anche se i più ammettono candidamente di non sapere nulla o quasi nulla dello sviluppo agile, si dimostrano comunque interessati e disponibili a informarsi o seguire corsi sull'argomento.

Tutto ciò lascia aperta una porta alla possibilità di un uso delle metodologie agili nella pratica corrente, seppur con qualche riserva. Come rileva un docente:

"*Si tratta di metodi interessanti da usare di tanto in tanto. Non sempre, però: ci viene chiesto di aggiornarci continuamente, ma tutto questo richiede tempo e considerevole impegno*".

6 Il prossimo esperimento: sviluppo agile

Forti dell'esperienza acquisita, stiamo continuando ad approfondire il tema dell'Educazione Agile. In particolare, la domanda di ricerca da cui abbiamo iniziato un nuovo lavoro (in corso) è: la *metodologia di sviluppo agile* è o no più funzionale all'insegnamento della programmazione rispetto al tipico *metodo di sviluppo tradizionale* (Waterfall)? Tra le tante, una delle più significative differenze tra i due approcci consiste nel differire molte decisioni progettuali sino alla fase di codifica: l'approccio agile rigetta infatti il principio del Big Design Up Front (BDUF) [13].

Premettiamo che i progetti scolastici sono, per loro natura, soltanto simulazioni di un reale progetto produttivo. Ciò è dovuto a diversi fattori specifici e necessari all'insegnamento – l'intervento-guida del docente, i tempi elastici, la necessità di ripetizione, ecc. Ciò detto, possiamo comunque affermare che gran parte dei progetti scolastici, come ad esempio il progetto per l'esame di stato, assomiglia a una forma ibrida di Waterfall con elementi iterativi, dovuti all'intervento correttivo/consultivo del docente.

Abbiamo cercato quindi di costruire un'esperienza che potesse testare uno lo sviluppo Waterfall, più simile alla realtà da un lato, e una simulazione accettabilmente realistica di una metodologia agile dall'altro. In entrambi i casi si è deciso di eliminare la parte di progettazione iniziale, che avrebbe richiesto un tempo eccessivo e avrebbe reso molti difficile il confronto tra i dati.

Nel caso Waterfall, in linea con le pratiche usuali, ai programmatori è fornito un SRS dettagliato e completo, che simula il precedente lavoro dell'analista capo; nel caso Agile, si sono progettate User Story piuttosto generiche, lasciando quindi ai ragazzi ampi margini discrezionali anche nella strutturazione generale del programma.

Interessante anche il ruolo interpretato dal docente: nel caso Waterfall incarna l'aspetto “manageriale” del progetto, attento quindi ad aspetti formali come il rispetto dei tempi, mentre nel caso Agile interpreta il ruolo del **Product Owner**, quindi con responsabilità dirette sulle funzionalità quali, ad esempio, decidere se una storia è stata correttamente realizzata oppure no.

Non si impongono particolari metodologie di lavoro, come le pratiche agili utilizzate nel precedente esperimento, oppure l'uso di specifici strumenti software (IDE, sistemi di versioning, ecc.); ad ogni gruppo è stata lasciata piena libertà di auto-organizzazione. Parimenti, non ci sono vincoli particolari sulla costruzione dei gruppi, se non sulla loro dimensione (dai 4 ai 6) e sulla distribuzione il più possibile omogenea degli elementi particolarmente “bravi”, in modo da dare a tutti le stesse potenzialità di riuscita.

Forti dell'esperienza accumulata dallo studio precedente, abbiamo deciso di ampliare considerevolmente la base numerica e geografica del campione da analizzare. Gli accordi con le scuole sono ancora in corso, ma al momento della scrittura di questo documento sono più di dieci le scuole che hanno già accettato o addirittura svolto l'esperimento, mentre altrettante sono state contattate e devono prendere le loro decisioni. È quindi concepibile pensare che il test coinvolga più di duecento ragazzi, distribuiti su otto regioni italiane, dal Trentino alla Sicilia.

I dati raccolti finora sono troppo limitati per un'analisi approfondita; tuttavia, si nota una tendenza abbastanza evidente: i gruppi Waterfall realizzano prodotti tipicamente con codice pulito e una interfaccia grafica impeccabile, ma con poche funzionalità; i gruppi Scrum producono invece programmi più funzionali, ma con interfacce scarne e spesso di difficile utilizzo. Stabilire quale di questi due approcci sia più funzionale alla didattica è ovviamente oggetto di discussione e di “scelte di campo” tra i docenti che non approfondiremo in questa sede.

7. Conclusioni

Un esperimento come questo non può certo dare risposte definitive sull'utilità didattica delle pratiche agili: un conto è realizzare un’esperienza autoconclusiva, un altro è il loro utilizzo sistematico nella pratica quotidiana.

Tuttavia, si possono trarre alcuni suggerimenti utili per integrare “la cassetta degli attrezzi” di ogni insegnante:

- Introdurre il Pair Programmaing utilizzando coppie eterogenee.
- Utilizzare le User Stories per comunicare e progettare.
- Più in generale, tentare l'utilizzo di varie tecniche agili per via del positivo effetto sulla motivazione e il clima della classe.

Il nostro gruppo di lavoro sta proseguendo con gli esperimenti cercando di allargare il più possibile la base geografica dell'esperimento. Non è esclusa la prosecuzione degli esperimenti all'anno successivo, per cui invitiamo i colleghi che leggono il presente documento a contattarci se ritengono che l'esperienza possa essere interessante anche per i propri studenti.

Bibliografia

[1] The Standish Group. Chaos manifesto 2013, https://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf

[2] Guide to Agile Practices, http://guide.agilealliance.org

[3] Eduscrum, http://eduscrum.nl/en/

[4] Teaching that Makes Sense, http://www.ttms.org/

[5] Agile School Manifesto, http://www.infoq.com/articles/agile-schools-education.

[6] C. Wohlin, P. Runeson, M. Host, and M. Ohlsson. Experimentation in software engineering, 2000.

[7] Kolb, David A. Experiential learning: Experience as the source of learning and development. FT press, 2014.

[8] Tennis Game Kata, http://www.codingdojo.org/cgi-bin/index.pl?KataTennis

[9] J.-L. Letouzey. The SQALE method for evaluating, technical debt. In Proceedings of the Third International Workshop on Managing Technical Debt, pagine 31–36. IEEE Press, 2012.

[10] Marcello Missiroli, Daniel Russo and Paolo Ciancarini, "Learning Agile Software Development in High School:an Investigation", in corso di prubblicazione nei Proceeding of 2016, The 38th International Conference on Software Engineering, Austin, TX, May 14 - 22, 2016, DOI: http://dx.doi.org/10.1145/2889160.2889180

[11] Dati relativi al primo esperimento: https://bitbucket.org/marcello missiroli/agileschool data.

[12] Video di riferimento: parte formativa, https://youtu.be/4DvMcIiEPKE; parte operativa, https://youtu.be/UyZ1jZcRjFM

[13] Wikipedia: https://en.wikipedia.org/wiki/Big_Design_Up_Front