Colmare il gap cognitivo tra committenza e software developers

Cesare Bianchi
cesare@cesarebianchi.com
UCD School of Computer Science and Informatics, Dublin 4, Ireland

1. Introduzione

L'informatizzazione è ormai un fenomeno onnipresente, e la diffusione del web ha ulteriormente accelerato la spinta, per qualsiasi azienda, a dotarsi dei nuovi strumenti indispensabili. Tuttavia, sia tra gli addetti ai lavori sia tra la committenza non informatizzata, non c'è la consapevolezza della totale differenza "ontologica" del "prodotto informatico" rispetto agli altri prodotti normalmente commerciati. Benché convinti di "parlare la stessa lingua", committenza ed informatici ragionano in modi completamente differenti: si potrebbe dire che vivono in universi paralleli. Il risultato è un cospicuo spreco di tempo e risorse.

Sono stati proposti vari metodi per migliorare l'interazione tra committenza e sviluppatori, tuttavia essi risolvono solo parte dei problemi, in quanto ignorano il fulcro della questione: il gap cognitivo tra il cliente e gli informatici incaricati di realizzare il software.

2. Un caso tipico

Un'azienda di noleggio pullman è in lenta ma costante espansione. La gestione è sempre stata fatta a mano, i computer sono stati usati solo marginalmente e poco più che come macchine da scrivere evolute. Il proprietario decide di mettersi al passo con i tempi e dotarsi di un sito internet. Non conoscendo nulla di informatica, pensa che il software sia una merce acquistabile come tutte le altre. La sua esperienza è con fornitori che hanno merci da acquistare ben chiare e definite, su cui è immediato fare un preventivo di tempi e prezzi, che in genere viene rispettato.

Si rivolge quindi ad una ditta di informatica, che prontamente invia il suo "commerciale". Costui ha solo una vaga infarinatura di informatica, ed il suo unico obiettivo è convincere il cliente ad acquistare il più possibile. Prospetta perciò al cliente la possibilità di aumentare la produttività dell'azienda tramite la realizzazione di un

software gestionale integrato al sito internet, con pubblicità inclusa sul web, il tutto da realizzare in meno di tre mesi.

Il proprietario dell'azienda sottoscrive un contratto in cui sono spiegati molto genericamente i termini del lavoro da farsi, non rendendosi conto della poca chiarezza delle diciture "sito internet" e "software gestionale".

Viene quindi inviato il "capo progetto" per stilare l'analisi del software. Costui cerca di farsi spiegare il processo di gestione dal proprietario dell'azienda, e compila un'analisi molto sommaria dei requisiti del software da implementare, che viene poi passata ad uno sparuto gruppo di sviluppatori con poca esperienza. Questi ultimi, pur non conoscendo nulla della gestione di una azienda di pullman, e sebbene in possesso solo delle sommarie informazioni contenute nell'analisi, cercano di realizzare il software commissionato. Poiché la ditta di informatica non ha fiducia nei suoi dipendenti, non permette loro di avere contatti diretti con i clienti: ogni richiesta di informazioni deve passare attraverso il capo progetto, che la gira al cliente e poi riferisce la risposta agli sviluppatori.

Dopo un anno e mezzo di comunicazioni frammentate e distorte, di frustrazioni da parte di tutti, di crescita delle funzionalità incluse nel software e conseguente crescita del costo, viene finalmente rilasciato il "prodotto", completo di manuale d'uso, totalmente incomprensibile per il cliente. Costui, per nulla soddisfatto, sostiene che il prodotto vendutogli non corrisponde a quanto richiesto, e per risolvere il contenzioso la ditta di informatica produce (con ulteriori costi interni) delle "schede di test". Dopo ulteriori discussioni e giornate dedicate ad istruire i dipendenti della ditta di pullman ad utilizzare il nuovo software, la ditta di informatica riceve finalmente il pagamento, inferiore ai costi sostenuti. L'azienda di pullman, dopo alcuni mesi, smette di usare il software, poiché si rende conto che, rispetto alla gestione a mano, si impiega più tempo e si ha meno flessibilità.

Un'opportunità di progresso è stata sprecata, ed il gap che separa l'informatica da chi non la conosce è stato ulteriormente scavato, invece di essere colmato. Ma cosa, realmente, è andato storto?

3. Un'analisi superficiale

Il caso appena esposto, *mutatis mutandis*, è la situazione che si ripete quasi sistematicamente quando una azienda richiede lo sviluppo di software. Ad una prima analisi si possono cogliere facilmente alcuni errori grossolani: ad esempio la malafede della ditta di informatica, più interessata al guadagno che al creare prodotti di qualità. Un approccio più realistico ed onesto, sapendo di non poter garantire al cliente budget e tempi predeterminati, sarebbe stato il proporgli del software modulare, ad iniziare da un piccolo programma autoconsistente e funzionante.

L'utopia di riuscire ad estrapolare tutte le informazioni in un'unica analisi iniziale e stilare una pianificazione certa, è invece purtroppo ampiamente diffusa, e senza dubbio l'impedire una comunicazione diretta tra committente e sviluppatori non consente a questi ultimi di accedere in modo chiaro ad informazioni necessarie.

Per porre riparo a questi problemi da anni si propongono diversi metodi, come ad esempio "Scrum" (Schwaber , 1997), che prevedono brevi cicli di analisi-sviluppo-testing, in cui il cliente viene coinvolto il più possibile, al punto tale che la situazione ideale prevedrebbe che gli sviluppatori lavorassero direttamente nella sede dell'azienda cliente. Opponendosi al modello "waterfall" (Royce, 1970), in cui il processo di

implementazione procede appunto "a cascata", con attività rigide e predeterminate, è stata spesso prospettata la necessità di modelli iterativi ed incrementali di sviluppo software, come stigmatizzato infine dall'"Agile Manifesto" (Beck, Kent, 2001). Rifacendosi all'esperienza del modello Toyota, Poppendieck e Poppendieck (2003) hanno adattato i principi del metodo "Lean" al mondo dello sviluppo software, sottolineando in questo modo la necessità di ridurre gli sprechi, *in primis* la burocrazia e la poca chiarezza dei requisiti.

Ovviamente, prerequisito per adottare tali metodi è la responsabilizzazione degli sviluppatori e di conseguenza la fiducia nella loro correttezza e capacità professionale (Boehm e Turner - 2004 - sottolineano ad esempio che solo sviluppatori senior hanno l'esperienza necessaria per usare metodi "Agile").

Nonostante ciò, pur focalizzandosi sulla comunicazione tra cliente e sviluppatori, un aspetto fondamentale del problema continua ad essere ignorato: la totale diversità della loro *forma mentis*.

4. Il software come modello della realtà

Nella storia delle tecniche per lo sviluppo di software, è costante il riferimento implicito alla necessità di ricreare, nel computer, un modello astratto e semplificato del mondo reale (limitato ovviamente al dominio di cui il programma si andrà ad occupare). A partire dal celeberrimo paradigma "entità-relazioni" usato per modellare le basi dati (Pin-shan Chen, 1976), passando per la programmazione orientata ad oggetti (in cui vengono definiti oggetti astratti, che rappresentano schematicamente entità reali, i quali sono capaci di svolgere azioni ed interagire con altri oggetti astratti), per giungere allo Unified Modeling Language (Object Management Group, 1997), usato per creare diversi modelli necessari all'analisi ed implementazione del software (ad es. del processo produttivo, degli oggetti astratti, della base dati, etc.), si è assistito ad un proliferare di tecniche empiriche per sintetizzare ed astrarre gli aspetti essenziali del dominio applicativo.

Benché nell'ambito della ricerca siano chiari da tempo gli aspetti filosofici implicati nella creazione di tali "ontologie" (Klein e Hirschheim, 1987), solo recentemente è stato proposto di strutturare, come passo fondante per la progettazione del software, delle ontologie formali del dominio applicativo (Sugumaran e Storey, 2006). Nella pratica quotidiana, tuttavia, sia per mancanza di formazione sia per inconsapevolezza della delicatezza delle operazioni connesse alla creazione di modelli astratti della realtà, si seguono metodi empirici ed intuitivi, certamente più rapidi ma per loro natura maggiormente proni ad errori. E' utile però anche ricordare la considerazione fatta poc'anzi, riguardo alla impossibilità di analizzare a priori l'intero processo produttivo e dominio applicativo, e la necessità perciò di adottare metodi iterativi ed incrementali.

5. Universi paralleli

Risulta perciò chiaro che gli sviluppatori di software ragionano secondo schemi ben diversi da quelli comuni: il loro compito è creare un micro-mondo astratto usando linguaggi di programmazione basati su semplici costrutti logici. D'altro canto, tranne rari casi, sono totalmente digiuni della "logica di business" (ovverosia il dominio applicativo e i processi produttivi del committente), ed essendo abituati a ragionare secondo schemi

semplici e razionali, hanno difficoltà a comprendere (e modellare) le complesse sfaccettature del dominio reale.

Di contro, il committente conosce a fondo (sebbene spesso solo in modo empirico ed intuitivo) i processi produttivi, ma avendo un background culturale che spesso non include non solo l'informatica ma anche strumenti più semplici di astrazione e di logica, si trova nell'impossibilità di comprendere i modelli mentali usati dagli informatici.

Poiché questo divario spesso non viene affatto riconosciuto, o se avvertito, la soluzione viene lasciata alla buona volontà dei singoli, si incorre di norma in incomprensioni evitabili e sprechi inutili di tempo e risorse. Se invece tale divario venisse riconosciuto, analizzato e risolto, se ne potrebbe trarre profitto sia in termini di qualità del lavoro che di apprendimento e crescita, e, non meno importante, di bilancio consuntivo delle risorse impiegate.

Il riconoscere e l'affrontare tali differenze culturali e di *forma mentis* potrebbe infatti innescare una dialettica da cui entrambe le parti potrebbero uscire arricchite. Il committente, soprattutto, oltre ad ottenere un prodotto più rispondente alle proprie necessità, potrebbe fruire degli strumenti logici degli informatici per fermarsi a riflettere ed analizzare più a fondo i processi produttivi che dà per scontati, così da avere degli *insight* utili a razionalizzare ed ottimizzare la propria azienda.

6. Costruire un ponte

Tuttavia, spesso né gli informatici né il committente hanno gli strumenti cognitivi necessari ad analizzare il gap tra loro esistente e mettere in atto un processo comunicativo che permetta l'instaurarsi di tale dialettica virtuosa. Al fine di iniziare una comunicazione non equivoca e non "patologica", può occorrere un soggetto terzo che, oltre a conoscere approfonditamente la logica informatica, sia esperto di comunicazione ed interazione, così da fungere da "ponte" iniziale tra le due parti.

L'identificazione di tale soggetto "ponte" può essere difficoltosa, in quanto i manager delle aziende informatiche sono in genere ex-programmatori con spirito imprenditoriale, cui manca sia la capacità di analizzare tale gap cognitivo, sia anche la consapevolezza di dover trattare il software come un servizio evoluto e non come una merce.

Da un punto di vista di marketing e di budget, il coinvolgimento di una ulteriore figura professionale potrebbe apparire uno spreco di denaro. Considerando però la miriade di progetti, come l'esempio illustrato all'inizio, il cui bilancio finale è di norma di un ordine di grandezza più grande di quello inizialmente stimato, risulta chiaro come tale spreco potrebbe essere invece facilmente evitato, investendo nel ridurre le incomprensioni.

La figura professionale che sarebbe perciò necessaria per ottimizzare la comunicazione tra le parti e la produzione del software, per ridurre gli sprechi, aumentare la qualità e massimizzare la soddisfazione (professionale ed economica) di tutte le parti, dovrà perciò essere esperta sia di psicologia che di informatica, nonché avere una spiccata capacità di analisi, per comprendere di volta in volta il dominio applicativo del cliente corrente.

Il suo compito non sarà di analisi dei requisiti del software, bensì quello, inizialmente, di evidenziare le differenze culturali e cognitive delle parti coinvolte, e poi

di supervisionare la comunicazione tra le parti, al fine di evitare fraintendimenti e l'instaurarsi di comunicazioni patologiche.

In questo senso, non dovrà fare da tramite (con il rischio di sostituire semplicemente la figura - ridondante ed inutile - del "capo progetto" citato nel caso tipico) tra sviluppatori e committente, cosa che non risolverebbe il problema ed aumenterebbe di nuovo la complessità del sistema e l'entropia prodotta, bensì appunto fungere da supervisore e counselor.

7. Conclusioni

In conclusione, ciò che occorre, in prima istanza, è un salto culturale che riconosca nel software non più una merce da vendere (con tutte le logiche di mercato che tale concezione si porta dietro), bensì un servizio di ottimizzazione e razionalizzazione dei processi produttivi, utile ad una crescita professionale di tutte le persone coinvolte. Considerando poi il gap culturale e cognitivo esistente tra cliente e sviluppatori, al fine di ridurre le incomprensioni e con esse gli sprechi, risulta necessaria una nuova figura professionale, che supervisioni le interazioni tra le parti, aiutandole a comprendersi, ad apprendere e a crescere, professionalmente quanto umanamente.

Ringraziamenti

Desidero ringraziare la Dr. Proia per il suo sostegno ed i suoi suggerimenti, il Dr. Vicari per avermi introdotto ai metodi di gestione dello sviluppo del software, nonché le varie aziende per cui ho lavorato, per avermi ispirato la necessità di analizzare questo problema.

Bibliografia

Beck, K. et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. http://agilemanifesto.org/

Boehm, B.; R. Turner (2004). "Balancing Agility and Discipline: A Guide for the Perplexed", Boston, MA: Addison-Wesley. pp. 55–57

Klein, H. K., Hirschheim, R. A. (1987), "A Comparative Framework of Data Modelling Paradigms and Approaches", The Computer Journal 30 (1): 8-15

Object Management Group (1997), "Unified Modeling Language", http://www.omg.org/spec/UML/

Pin-shan Chen, P. (1976), "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems (1), pp. 9-36

Poppendieck, M., Poppendieck, T. (2003), "Lean Software Development: An Agile Toolkit", Addison-Wesley Professional

Royce, W. (1970), "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON 26 (August): 1–9

Schwaber, K. (1997) "Scrum Development Process" in OOPSLA Business Object Design and Implementation Workshop, Sutherland, Patel, Casanave, Miller, Hollowell, Eds. London

Sugumaran, V., Storey V.C. (2006). "The role of domain ontologies in database design: An ontology management and conceptual modeling environment", ACM Transactions on Database Systems 31 (3), pp. 1064–1094