38

gennaio 2009

i Quaderni

sommario

i Quaderni n. 38 gennaio 2009
Supplemento al n. 2/2008
di Innovazione

Registrato al Tribunale di Roma
n. 523/2003
del 15 dicembre 2003

Direttore responsabile
Franco Tallarita

Quaderno a cura di:
Emanuela Mariotti
Area Divisionale
"Piattaforme Applicative"

Redazione
Centro Nazionale
per l'Informatica nella
Pubblica Amministrazione
Via Isonzo, 21b
00198 Roma
Tel. (39) 06 85264.1
pubblicazioni@cnipa.it

I Quaderni del Cnipa
sono pubblicati all'indirizzo:
www.cnipa.gov.it

Premessa

Questo documento contiene le linee guida allo sviluppo di software "riusabile". Si potrebbe obiettare che di linee guida su come si sviluppa software è già ricca la letteratura dell'ingegneria del software. Era necessario un ulteriore documento metodologico sull'argomento? Vi sono alcune importanti motivazioni per rispondere in modo affermativo, che cercheremo di presentare qui di seguito.

Anzitutto, i contenuti del documento: queste linee guida non definiscono una metodologia dettagliata per sviluppare software, od un insieme di specifiche tecniche indirizzate agli sviluppatori, ma delle raccomandazioni, di natura tecnica, metodologica, organizzativa, per aiutare gli "acquirenti" di software "custom" (sviluppato ad hoc su requisiti dell'utente) a costruire un capitolato tecnico per acquisire un servizio di sviluppo software che produca, alla fine, del software "facilmente riusabile". Le raccomandazioni di natura metodologica presenti nel documento, che si traducono in requisiti di processo che l'acquirente dovrebbero imporre ai propri produttori di software *custom*, sono più una meta-metodologia che una metodologia.

L'altra motivazione sta nel fatto che i contenuti di questo documento sono indirizzati alla Pubblica Amministrazione. La Pubblica Amministrazione Centrale italiana (PAC) è un grande acquirente di software *custom*, e ne detiene ingenti quantità nei propri sistemi informativi – oltre 13 milioni di punti funzione, secondo la Relazione annuale CNIPA per lo stato dell'ICT nella PAC nel 2007. Sulla base di questa costatazione, diverse disposizioni normative – per citare solo le ultime il dPCM 31 maggio 2005, n.157 e il Codice dell'Amministrazione Digitale – hanno individuato nel "riuso" di software già esistente una delle opzioni che ogni amministrazione dovrebbe considerare al momento di programmare un nuovo acquisto di software. Nonostante tali prescrizioni, e nonostante la mole del software potenzialmente disponibile, la Pubblica Amministrazione (soprattutto quella centrale) fa poco ricorso al riuso, e quando riusa spende per riusare somme non distanti da quelle che avrebbe speso per sviluppare *ex novo* il software di cui aveva esigenza (i costi del riuso si riferiscono alle personalizzazioni, adattamenti, integrazioni etc.. che vanno eseguiti sia sugli aspetti funzionali che su quelli tecnologici del software da riusare, per renderlo "utilizzabile" nel contesto – tecnico, funzionale e organizzativo – nel quale va riusato).

Perciò, obiettivo primario di questo documento è contribuire a una maggiore diffusione del riuso di software nella P.A., rendendolo più facile e possibile a costi contenuti, in modo da contribuire a razionalizzare le spese pubbliche per l'ICT e rendere disponibili risorse per finanziare una reale innovazione dei sistemi applicativi della P.A.

Altri obiettivi connessi a questo, e che le linee guida consentono di perseguire, possono essere considerati il miglioramento della qualità del software della P.A., la creazione nella Pubblica Amministrazione una nuova community orientata all'innovazione continua dei sistemi applicativi e, infine, qualificare la domanda di software *custom* del settore pubblico, in modo da indirizzare l'offerta delle imprese del settore verso l'innovazione, in un quadro competitivo internazionale in linea con i trend ICT.
Per fare ciò, il documento analizza le cause della poca efficienza del riuso nella PAC e propone una strategia per superare queste cause, offrendo alcune raccomandazioni di natura tecnica, altre di natura metodologica, altre di tipo organizzativo. Per favorire la diffusione delle raccomandazioni, tutte quelle presentate sono in linea, o direttamente tratte, dagli standard di ingegneria del software più diffusi (soprattutto standard ISO). In definitiva, si può dire che queste linee guida non definiscono una nuova metodologia per sviluppare software, né definiscono delle nuove soluzioni tecniche, ma sistematizzano – focalizzandolo sulla produzione di software "riusabile" – l'ampio materiale offerto dalla letteratura sulla ingegneria del software, presentandolo sotto forma di possibili raccomandazioni che un acquirente di software *custom* dovrebbe considerare al momento di negoziare con i propri fornitori le caratteristiche del software da sviluppare e le modalità di suo sviluppo.

Chiarite le finalità di queste linee guida, entriamo ora più in dettaglio nei razionali che ne sono alla base, in modo da cominciare a delinearne in modo più preciso i contenuti (citeremo diversi dati provenienti dalla relazione annuale CNIPA sullo stato dell'ICT nella PAC nel 2007).
La PAC è il più grande acquirente di tecnologia informatica in Italia. Nel 2007, ha speso 1,6 miliardi di euro in beni e servizi IT (circa l'8% del valore del settore IT in Italia), eppure stenta a trovare una politica di acquisti comune o perlomeno coordinata. La disponibilità di piattaforme pubbliche di *e-procurement* ha permesso di controllare i costi unitari delle tecnologie acquisite, ma non ne ha impedito la proliferazione scoordinata. La PAC non riesce a generare economie di scala e a razionalizzare l'uso delle risorse IT di cui dispone: le tecnologie usate restano eterogenee da una installazione all'altra, l'interoperabilità e la cooperazione applicativa tra i sistemi informativi delle varie PAC è un obiettivo ancora lontano, continuano a vedersi CED e sale server di pubbliche amministrazioni sorgere a poca distanza l'una dall'altra, i server della sola PAC installati sono oltre 31.000 e gli addetti IT interni sono oltre 15.400!
Il caso del software applicativo di proprietà della PAC è, in tal senso, eclatante: la PAC ha speso nel 2007 262 milioni di euro in acquisto di software sviluppato ad hoc e circa 60 nella sua manutenzione (più circa 50 in acquisto di pacchetti applicativi più 25 per la loro manutenzione e 13 per la loro locazione) ed è proprietaria di oltre 13,2 milioni di PF di software "custom". Si tratta di cifre rilevanti, che non sono cambiate significativamente negli ultimi anni. Ciò, nonostante le sempre maggiori ristrettezze di bilancio che imporrebbero una razionalizzazione delle spese. In particolare, diverse norme (da ultimo IL CAD) hanno imposto agli amministratori pubblici di valutare, al momento di programmare un acquisto

di software, l'opzione del riuso di software già esistente e già di proprietà della PAC. In effetti, il riuso di software tra PAC negli ultimi anni è stato molto basso e ha inciso in maniera marginale nei conti pubblici.
Vi sono diverse motivazioni per la finora scarsa efficienza del riuso nella PAC, in parte tecniche, e in parte organizzative e culturali.
Le motivazioni tecniche possono essere ricondotte sostanzialmente a due: la "specificità" dei domini applicativi coperti dal software di proprietà delle singole PAC (questi software realizzano funzioni proprie di ogni PAC che sono difficilmente esportabili in altre) e il fatto che questi software non sono stati finora progettati e realizzati con caratteristiche tali da poter essere facilmente riusati. Si tratta di problemi apparentemente diversi ma che in realtà sono tra loro strettamente connessi.
In effetti, è vero che una quota rilevante del software applicativo in uso presso la PAC serve funzioni specifiche (secondo alcune stime questa quota di software è pari a circa il 70% di quello installato). Tuttavia, il riuso non è frequente neppure nella restante quota di software che realizza funzioni più "trasversali" alle esigenze delle PAC (ad esempio quello che realizza funzioni di *back office*, a supporto del funzionamento degli uffici, che, in teoria, nella PAC dovrebbero essere molto simili da una amministrazione all'altra). In realtà, la motivazione della specificità dei software in uso presso le varie PAC, ad una analisi tecnica, risulta una giustificazione poco convincente per il loro scarso riuso. Vediamo perché.
I moderni software sono ormai realizzati dai produttori in larga parte riutilizzando e assemblando / adattando componenti e semilavorati già pronti. I produttori dispongono di librerie di componenti pronti o semi pronti, ognuno dei quali dedicato a fornire funzioni specifiche nell'architettura complessiva di un sistema applicativo. È un processo di produzione "industriale", che viene da sempre utilizzato in ingegneria e che da qualche anno è stato mutuato anche nella produzione del software. Le più recenti tecnologie IT sono state tutte incentrate sul fornire strumenti agli sviluppatori per favorire questa pratica, dai "java beans", alle architetture SOA, ai web services etc. Anche sul web esistono librerie – e cataloghi – di componenti già pronti o semilavorati cui gli sviluppatori possono attingere, così come esistono librerie di soluzioni progettuali già sviluppate (patterns, frameworks etc) che vengono utilizzate dai progettisti. La comunicazione tra ii vari componenti in queste architetture, anche quando si tratta di componenti provenienti da diversi costruttori od *open source*, non è più tecnicamente un problema e sono disponibili diversi strumenti di "integrazione" dell'apporto dei vari componenti di una architettura nell'ottica di processi di servizio.
Se questo è lo stato dell'arte della tecnologia, però, la situazione del software della PAC è molto differente: il software che la PAC possiede nei propri cataloghi o è datato o non è stato reso disponibile dal produttore come "somma di componenti", per cui appare in larga misura come costituito da "monoliti" dedicati a servire macro processi, poco "portabili" in altri contesti e non facilmente sfiocchettabili in componenti dedicati a svolgere funzioni più elementari. Perciò, quando si riusano questi monoliti, il costo del loro riuso è alto.
Dal punto di vista organizzativo le motivazioni alla poca efficienza del riuso sono anch'esse essenzialmente riconducibili a due: la scarsa conoscenza che ogni PAC ha del software di

proprietà delle altre PAC (non esistono "cataloghi" condivisi di software applicativo di proprietà, né indici o repertori che rendano noto a tutti i potenziali "riusatori" ciò che è disponibile); la mancanza di cultura sulle tecniche e metodiche del riuso: nelle PAC sono rari gli "esperti" tecnici del riuso che possano valutare i costi e i benefici di un riuso rispetto ad altre soluzioni di acquisto (software interamente a pacchetto, personalizzazione custom di pacchetti, sviluppo ex novo custom).

Queste le premesse da cui parte questo documento. In un centinaio di pagine (forse molte, ma è stata privilegiata la completezza della rappresentazione, a scapito talvolta della sintesi), il documento, partendo da una analisi di dettaglio delle cause della scarsa efficienza del riuso nella PAC, propone delle soluzioni, presentate come una strategia in 4+4 punti (i primi essenziali e da mettere in atto rapidamente, i secondi che potrebbero essere attuati in un secondo momento).

Per i motivi che verranno evidenziati nel documento, il titolo delle linee guida è stato fissato in "linee guida allo sviluppo di software riusabile multiuso". In effetti, un software riusabile è un software che può essere "usato" più volte in diversi contesti, proprio come farebbe al caso della PAC.

1. Referenze

Queste linee guida sono state realizzate da un gruppo di lavoro istituito con delibera del Presidente del CNIPA n. 11 del 10 aprile 2006, poi integrato con successiva delibera n. 6 del 2 maggio 2007. Al gruppo hanno partecipato gli esperti designati dal CNIPA, Aitech-Assinform, Consip, Inail, Inps e Sogei, che sono qui di seguito citati e ringraziati per la loro attività.

CNIPA – Gianluigi Raiss, Stefano Fuligni,

Assinform – Giuseppe Neri, che è stato affiancato da numerosi esperti in rappresentanza delle principali Società che operano nel settore dello sviluppo del software

Consip – Claudio d'Alessandro e Paolo Luxardo

Inail – Carmela de Padova e Angelo Scarcia

Inps – Corrado Cardellicchio, Claudio Checcherini e Fabrizio Lucchetta

Sogei – Domenico Natale, Francesco Claudio Milone ed Enrico Pesce

Università – Vincenzo Ambriola (Università di Pisa), Giuseppe Santucci (Università La Sapienza di Roma)

Ha collaborato Luca Santillo per le sezioni riguardanti le metriche funzionali.

2. Contenuti, destinatari ed utilizzo delle linee guida

2.1 Contenuti delle linee guida

Le linee guida riportate in questo documento forniscono raccomandazioni (tecniche, metodologiche e organizzative) per definire i requisiti di un servizio di sviluppo di nuovo software *custom*[1] per la Pubblica Amministrazione (P.A.), affinché il software realizzato sia poi facilmente riusabile anche in altri progetti. Va subito precisato che tali raccomandazioni, come anticipato in Premessa, non definiscono una nuova metodologia di sviluppo software né introducono nuove soluzioni tecniche. Hanno viceversa lo scopo di rappresentare alcuni dei requisiti che un acquirente di software *custom* (per la precisione un acquirente pubblico) dovrebbe negoziare con il proprio fornitore, al fine di ottenere un software che sia poi più facilmente riusabile.

Tutte le raccomandazioni fornite in queste linee guida si rifanno, o richiamano direttamente, a standard di ingegneria del software, e in particolare a quelli ISO.

Le raccomandazioni contenute in questo documento completano quelle già contenute nelle linee guida CNIPA per la gestione di un progetto di riuso di software già esistente e di proprietà della P.A. (disponibili sul sito www.cnipa.gov.it) e nelle *Linee Guida alla qualità degli appalti ICT* (disponibili sul sito www.cnipa.gov.it), che trattano i criteri di scelta tra le opzioni Make or Buy or Reuse.

Le precisano, in particolare, per la sezione che riguarda la scelta di ricorrere al riuso per sviluppare nuovo software. Nelle precedenti linee guida, l'unica opzione di riuso analizzata era il riuso di una intera applicazione, previo adattamento e personalizzazione. In queste linee guida, si indica viceversa che, al momento di valutare l'opzione del riuso di software già esistente, l'acquirente debba valutare sia l'opzione di riusare una intera applicazione esi-

[1] Il software di tipo *custom* è quello che viene sviluppato ad hoc da un fornitore su una commessa specifica di un cliente, sulla base di requisiti forniti dal cliente stesso, e si differenzia in ciò dal software Commercial Off The Shelf (COTS) che viene invece prodotto per un utilizzo diffuso e generalizzato, senza una specifica commessa e senza una interazione diretta con gli utenti nella fase di sviluppo del prodotto. Questa modalità di acquisto del software è particolarmente utilizzata dalle Pubbliche Amministrazioni, che devono spesso ricorrere a commesse specifiche per soddisfare esigenze di digitalizzazione verticali, non risolvibili acquistando prodotti COTS. Perciò, la Pubblica Amministrazione è proprietaria di molto software *custom* e investe ogni anno ingenti somme per svilupparne altro, così come per mantenere e fare evolvere quello di cui è già in possesso.

stente, sia l'opzione di riusare solo alcuni componenti riusabili già disponibili in appositi cataloghi della P.A., da assemblare poi con altri sviluppati per l'occasione.
L'esperienza ha mostrato che i fattori chiave dell'efficienza del riuso sono la qualità del processo di produzione e la qualità di quanto viene riusato, e ciò è tanto più vero nel caso di uno sviluppo di software basato sull'assemblaggio di componenti riusabili. Pertanto, tra le raccomandazioni inserite in queste linee guida ve ne sono alcune di carattere generale che riguardano l'organizzazione del processo di produzione del software, affinché sia regolato e controllato, ed altre dedicate specificatamente al *testing* del software e alla misura della sua qualità durante il processo produttivo, al fine di aumentare la qualità del software che viene sviluppato per il riuso.
Le raccomandazioni che vengono qui fornite riguardo il processo di produzione del software riprendono ed espandono quelle già contenute nelle citate *Linee Guida alla qualità degli appalti ICT* pubblicate dal CNIPA, adattandole al processo di sviluppo per il riuso trattato in questo documento.
Molto del software *custom* che viene sviluppato dalla (o per conto della) Pubblica Amministrazione è realizzato integrando componenti *custom* e funzioni fornite da prodotti commerciali proprietari, di mercato (detti "da banco", o Commercial Off the Shelf - COTS), ovvero da software *open source*, non sviluppato *ad hoc* per la Pubblica Amministrazione. Nel caso di nuove applicazioni software realizzate integrando software *custom* con funzioni COTS o funzioni di pacchetti *open source*, le linee guida si rivolgono alle sole componenti *custom* poste *on top* al prodotto COTS o al software open source.
Non vi sono, nelle linee guida, indicazioni dettagliate in merito agli aspetti giuridici e amministrativi impattati dai progetti di riuso. Le implicazioni giuridiche che derivano dall'adozione generalizzata nella P.A. di uno sviluppo di software per assemblaggio di componenti riusabili, che deve basarsi sulla disponibilità di cataloghi di componenti riusabili, sono infatti troppo vaste per poter essere trattate in un documento con finalità meramente tecniche e metodologiche. Tuttavia, aspetti come la limitazione delle responsabilità nel processo di sviluppo e negli effetti del software riusato, e della proprietà del software, andrebbero meglio definiti in appositi contesti normativi, al fine di dare reale efficacia al processo di sviluppo di software descritto in questo documento.
Le linee guida non trattano in dettaglio neppure la semantica dei cataloghi del software riusabile della P.A. che dovrebbero essere costruiti per ospitare – a regime – le informazioni necessarie per riusare i componenti software riusabili. Si tratta infatti di un argomento che da solo richiede una trattazione specifica. Qui si individua tuttavia come possibile fonte per la descrizione dei contenuti dei cataloghi la specifica OMG "RAS" (Reusable, Asset Specification").

2.2 DESTINATARI E UTILIZZO DELLE LINEE GUIDA

I destinatari di queste linee guida sono le pubbliche amministrazioni che intendono acquisi-

re nuovo software di tipo *custom,* sia realizzato con risorse interne, sia sviluppato tramite affidamento a un fornitore specializzato.
Le raccomandazioni fornite da queste linee guida sono quindi una possibile fonte per definire i requisiti di un servizio di sviluppo software *custom*, nell'ambito di un contratto con un fornitore (in questo caso potrebbero essere parte di un capitolato tecnico), ovvero di un progetto condotto con risorse interne.
Va osservato che le raccomandazioni contenute in queste linee guida non sono integralmente applicabili ad ogni progetto di sviluppo di nuovo software *custom*. Le linee guida sono infatti un *framework* di riferimento, che va ritagliato e adattato secondo le specificità di ogni contesto. Non sempre un nuovo software deve necessariamente possedere tutte le caratteristiche di riusabilità qui individuate (la cui realizzazione ha comunque un costo). D'altra parte, quando applicabili, queste linee guida sono state scritte per essere utilizzate a prescindere dalle dimensioni, architettura, tecnologia, linguaggio con il quale il software è realizzato.
Analogamente, è necessario anche precisare che le modalità di sviluppo software descritte in queste linee guida non sono le uniche possibili per una Pubblica Amministrazione, che può scegliere di sviluppare nuovo software *custom* seguendo un processo diverso da quello qui descritto, secondo le proprie esigenze e la convenienza economica.
Adottando il processo di sviluppo software basato su componenti riusabili descritto in questo documento, le amministrazioni dovrebbero però ottenere benefici in termini di contenimento dei costi di produzione del software, miglioramento della sua qualità, accelerazione dei tempi di sviluppo. Questi effetti benefici si amplificano nel tempo, via via che aumenta il numero e la qualità di componenti riusabili – realizzati secondo le raccomandazioni qui fornite – disponibili nei cataloghi della P.A., secondo un principio già evidenziato nelle architetture SOA (Service Oriented Architecture) e nei web services utilizzati nel paradigma della cooperazione applicativa definito nel Sistema Pubblico di Connettività (SPC) al fine di far interoperare tra loro le pubbliche amministrazioni in rete.
I benefici che si propongono di raggiungere queste linee guida vanno al di là della singola amministrazione, e sono diretti a migliorare la capacità di spesa complessiva per l'ICT nella Pubblica Amministrazione che, nel caso del riuso di software, deve necessariamente arrivare a una visione integrata e unitaria delle sue strategie di acquisto del software, visti i volumi che è in grado di movimentare.
Infine, si segnala come queste raccomandazioni non si applicano alla reingegnerizzazione, al fine del loro riuso, degli applicativi dei sistemi *legacy* di cui è ancora ricca la Pubblica Amministrazione centrale. Per "incapsulare", o comunque rendere riusabili in nuovi contesti questi applicativi, sono state definite specifiche metodologie, alle quali si rimanda.[2]

[2] Esperienze nella reingegnerizzazione di applicativi legacy sono state condotte in alcune grandi pubbliche amministrazioni, come ad esempio quelle del comparto finanziario.

2.3 ORGANIZZAZIONE DEL TESTO

Le linee guida sono organizzate in capitoli, che contengono, oltre la Premessa, le referenze e questo capitolo di introduzione, i seguenti argomenti:

Cap. 3 – Contiene un'analisi del perimetro attuale del concetto di riuso, confrontato coni concetti limitrofi come portabilità, qualità, diffusione, reingegnerizzazione etc.. e una ipotesi di ridefinizione del concetto di riuso alla luce delle moderne architetture software e dell'evoluzione tecnica del settore IT, con un focus sul concetto di "riusabilità". In questo capitolo vengono anche trattati i temi del riuso nei processi di acquisto della Pubblica Amministrazione e dei fattori abilitanti il riuso.

Cap. 4 – Tratta alcuni elementi innovativi che possono estendere il concetto di riuso, come i *web services* e i componenti funzionali multiuso riusabili, concetto già presente nelle architetture SOA (Service Oriented Architecture).

Cap. 5 – Contiene una sintesi della situazione attuale del riuso nella Pubblica Amministrazione, che analizza i limiti del riuso come attualmente praticato e la sua scarsa efficienza.

Cap. 6 – Presenta la strategia proposta per superare i limiti attuali all'efficienza del riuso di software *custom* nella PAC, basata sullo sviluppo di componenti software riusabili, tratteggiata sinteticamente nei suoi aspetti chiave, che verranno approfonditi poi in successivi capitoli dedicati.

Cap. 7 – Definisce le caratteristiche tecniche che devono possedere i componenti software custom di nuovo sviluppo per poter essere considerati "riusabili".

Cap. 8 – Definisce i requisiti di un processo di produzione del software orientato a realizzare componenti riusabili e a riusare componenti nel processo produttivo, basato sull'utilizzo dei appositi cataloghi di componenti software riusabili e su una dinamica costruttiva di tipo get-put.

Cap. 9 – Tratta dei requisiti di massima dei cataloghi dei componenti software riusabili che le varie P.A. dovrebbero costruire per ospitare i componenti riusabili via via acquisiti (grazie al ricorso al processo di produzione get-put descritto in questo documento).

Cap. 10 – Definisce i contenuti di massima della documentazione – che deriva dal processo produttivo get-put del software – che va particolarmente curata per poter poi facilmente riusare un componente software (Piano del riuso, libretto del riuso, Piano della qualità).

Cap. 11 – Contiene un'analisi dell'organizzazione che, nelle P.A., può favorire il riuso di componenti software, come prescritto in queste linee guida, e dare maggiore efficienza al processo di riuso nella PAC.

Cap. 12 – Contiene un'analisi dell'impatto dello sviluppo di software riusabile sui costi di un progetto di sviluppo software.

In Appendice 1 è descritto sinteticamente il Reuse Maturity Model, un riferimento metodologico per valutare il grado di "capacità" e affidabilità con la quale una organizzazione che sviluppa software gestisce il tema del riuso. Il metodo è derivato dal CMMI (Capability Maturity Model Integrated), sviluppato dal Software Engineering Institute della Carnegie Mellon University di Pittsburgh, USA, e molto diffuso nel settore IT.

In Appendice 2 sono discussi aspetti di carattere generale relativi al processo di valutazione del software, la cui corretta pianificazione ed esecuzione è indispensabile per produrre software destinato al riuso.

In Appendice 3 è presentato un metodo di misurazione basato sui punti funzione che può permettere di quantificare la "quantità" di software riusato in un progetto di sviluppo di nuovo software.

In Appendice 4 è riportata la struttura commentata degli elementi base di un capitolato tecnico relativo ad una iniziativa di sviluppo per il riuso.

3. Il perimetro del riuso: quello attuale e ipotesi per una sua ridefinizione

3.1 Definizioni

3.1.1. Definizione di riuso ed evoluzione del concetto di riuso

Si intende come "riuso" di un software il complesso di attività svolte per poterlo utilizzare in un contesto diverso da quello per il quale è stato originariamente realizzato, al fine di soddisfare esigenze simili o anche solo parzialmente simili a quelle che portarono al suo primo sviluppo. Il prodotto originario viene "trasportato" nel nuovo contesto arricchendolo, se necessario, di ulteriori funzionalità e caratteristiche tecniche che possono rappresentare un "valore aggiunto" per il nuovo soggetto utilizzatore.

Un aspetto fondamentale del riuso nel contesto della Pubblica Amministrazione è che l'Amministrazione che "riusa" riceve il software gratuitamente dall'Amministrazione cedente, e lo acquisisce sostenendo solo le spese di suo adattamento, ma non quelle di progettazione e realizzazione.

La pratica del riuso non è certo una novità nel settore ICT e nemmeno nella Pubblica Amministrazione. Il riuso è infatti già utilizzato da tempo nello sviluppo di software con metodi industriali, ma si è negli ultimi tempi raffinato ed evoluto: dal riuso "fisico", che consisteva nell'inglobare nel programma principale codice sorgente proveniente da librerie di "subroutines", si è passati al riuso "concettuale", che si basa sulla progettazione a oggetti e sui patterns (schemi di progettazione di componenti assemblabili) per estendere il riuso ai modelli progettuali e alle architetture funzionali del software, per arrivare oggi al riuso "logico funzionale" permesso dai *web services* e dalle architetture orientate ai servizi (e.g. l'architettura SOA – Service Oriented Architecture), che consentono di riusare funzioni (servizi) applicative senza necessariamente avere il software riusato residente nelle proprie installazioni: basta infatti "richiamare" tali funzioni (installate in locazioni remote) con opportune procedure e protocolli capaci di attivare servizi "remoti" e distribuiti.

Nell'architettura del modello di cooperazione applicativa del Sistema Pubblico di Connettività (SPC) definita dal CNIPA per supportare lo scambio telematico e l'interoperabilità tra pubbliche amministrazioni, si fa largo riferimento a questo ultimo tipo di soluzioni. In definitiva, oggi si può sviluppare software riutilizzando schemi progettuali, idee, componenti software, parti di documentazione ma anche assemblando e facendo cooperare servizi applicativi distribuiti.

3.1.2 RIUSO *VS* DIFFUSIONE E REINGEGNERIZZAZIONE

È utile distinguere tra i termini "riuso" e "diffusione": il riuso, come sopra osservato, aggiunge valore ad un software nel trasportarlo da un ambiente ad un altro. Esistono, d'altra parte, software pensati per un largo utilizzo, che vengono "diffusi" presso molti utenti senza modificarne o adattarne le caratteristiche. È il caso, ad esempio, dei prodotti di *Office Automation* di mercato, che vengono appositamente progettati per essere utilizzati in vari contesti. In questo caso si può parlare di "diffusione", una pratica di acquisto del software che può consentire risparmi (ad esempio nel costo d'acquisto delle licenze per l'utilizzo di software proprietari, grazie alle economie di scala garantite dalla larga "diffusione" del prodotto) ed effetti benefici di standardizzazione delle modalità lavorative degli utenti, ma che non rientra strettamente nella logica del riuso.
Se, viceversa, gli interventi effettuati sul software per trasportarlo da un ambiente ad un altro ne modificano sostanzialmente le caratteristiche funzionali e/o tecniche, si parla in letteratura di "reingegnerizzazione" del software e non di riuso.
Il riuso si caratterizza quindi come una soluzione intermedia tra diffusione e reingegnerizzazione, con il vincolo, comunque, che il software da riutilizzare sia di proprietà del soggetto cedente.

3.1.3 QUALITÀ DEL SOFTWARE *VS* RIUSO

Il riuso del software genera, come valore aggiunto, una sua migliore qualità. Infatti, quanto più un software viene riutilizzato, tanto più sono le verifiche e i test cui è sottoposto dai vari soggetti utilizzatori. Ciò diminuisce la densità di errori residui (latenti) che il software contiene, migliorando progressivamente la sua qualità, secondo una logica ben nota alle comunità di sviluppatori di software *open source*.

3.1.4 QUALITÀ DEI DATI VS RIUSO

Come definito dallo standard ISO/IEC 1926, parte interante di una applicazione software è anche lo schema concettuale dei dati che il software tratta, anch'esso oggetto di possibile riuso. In questo caso, è indispensabile che il soggetto cedente fornisca a quello riusante un insieme strutturato di informazioni sui dati trattati dal software, e in particolare:

- un **glossario** dei dati trattati, (auspicabilmente comune a tutte le organizzazioni coinvolte nel riuso), che fissi il significato dei termini di 'dominio applicativo' utilizzati dal software;
- un **repository dei dati** che descriva tutti gli schemi concettuali utilizzati nei database e che descriva anche i dati scambiati, all'interno e all'esterno dell'organizzazione, in formato XML ed i termini utilizzati negli schemi concettuali, fornendo una definizione per ogni termine ed eventualmente la loro organizzazione in termini di lessico;
- un **repository dei metadati** che definisca l'interdipendenza tra i dati e le applicazioni e i processi che su di essi si basano.

Ai fini della riusabilità di un software la qualità di questi elementi connessi ai dati è un fattore fondamentale. Va da sé che è necessario primariamente garantire la consistenza e la qualità delle informazioni. Ne consegue che un'accurata attività di pulizia, integrazione e consolidamento dei dati è fondamentale per rendere possibile l'integrazione e il riuso di prodotti software e servizi ad essi connessi, e per poter utilizzare interfacce e modalità operative standard. Delle metodiche per valorizzare la qualità dei dati non si può trattare nell'ambito di queste linee guida, e si rimanda a lavori specifici, tra i quali va segnalata l'iniziativa ISO (promossa da Uninfo per l'Italia) per la pubblicazione di un modello di qualità dei dati (da pubblicare come standard ISO/IEC 25012, alla data emesso in forma di committee draft).

3.1.5 Estensione del riuso

Secondo lo standard ISO/IEC 9126, sono da considerarsi "oggetti software" non solo il codice sorgente vero e proprio, ma anche altre entità ad esso correlate:

1. documenti di specifica dei requisiti
2. documenti di progettazione funzionale e tecnica
3. documentazione d'uso, manutenzione e gestione
4. procedure di test e casi di test
5. procedure di installazione e configurazione
6. schemi di workflow che rappresentano la procedura con cui opera il software
7. schemi di organizzazione di basi di dati.

Il riuso può riguardare ognuno di questi elementi. Ciò è spesso dimenticato quando si sviluppa software. Ne consegue che molti software applicativi di proprietà della Pubblica Amministrazione sono corredati da entità correlate non aggiornate, non complete, difficilmente modificabili per adattarle ad altri contesti. Un processo di sviluppo software che si propone di realizzare componenti riusabili deve quindi provvedere alla riusabilità anche di tutti gli elementi sopra ricordati.

3.1.6 Definizione di riusabilità

Dopo aver precisato il concetto di riuso anche alla luce delle evoluzioni tecniche del settore e le possibili relazioni del riuso del software con altri fattori rilevanti in un progetto di sviluppo software, introduciamo ora il concetto di "riusabilità" del software. Definiamo quindi la "riusabilità" di un software applicativo come la sua capacità ad essere utilizzato in sistemi informativi e contesti diversi da quello originario, con costi di adattamento contenuti. Non è ovviamente possibile progettare un software che, potenzialmente, possa essere riutilizzato con minimi adattamenti in tutti i domini applicativi e/o i contesti operativi. Si può però ipotizzare di progettare software che posseggano determinate caratteristiche di base che ne agevolano l'adattamento in un certo numero di domini e contesti (eventualmente noti a priori). La riusabilità può essere quindi definita come il grado di facilità con la quale un software può essere adattato ad altri contesti, eventualmente noti a priori.

Con queste premesse, anticipando le indicazioni che verranno date nel seguito di questo documento, un software è riusabile più facilmente se:

- possiede un insieme di caratteristiche tecniche di base che ne permettono l'adattamento a vari contesti, in tempi e costi contenuti;
- è corredato da una documentazione progettuale, completa e prodotta con notazioni standard, tale che soggetti diversi da quelli che lo hanno originariamente realizzato possano facilmente modificarlo e adattarlo;
- è corredato da una documentazione d'uso costruita in modo da poter essere anch'essa modificata e adattata con facilità ad altri contesti;
- è corredato da una adeguata ed esaustiva documentazione sui test cui deve essere sottoposto ai fini del riuso in altri contesti ed è corredato di casi di test riusabili.

3.1.7 RIUSABILITÀ *VS* PORTABILITÀ

Frequentemente, si associa la facilità di riuso di un software esclusivamente alla sua "portabilità", quasi a farne dei sinonimi. Per "portabilità" si intende la possibilità di installare e far funzionare un dato software su "piattaforme" differenti da quella per la quale è stato originariamente progettato e realizzato. Senza entrare in questa sede nella non semplice impresa di individuare sia la tipologia (sistemi operativi, middleware etc..), sia la specifica tecnologia di "piattaforma" per le quali un software riusabile/portabile dovrebbe essere progettato, si può in ogni caso ritenere che la portabilità è in molti casi un importante fattore abilitante il riuso. Tuttavia, ai fini della facilità di riuso di un software è importante che tale software possegga, oltre la portabilità, anche diverse altre caratteristiche, quelle che ne facilitano la modifica e l'adattamento per rispondere ai requisiti del nuovo contesto in cui viene riusato.
Se analizzato dal solo punto di vista economico, d'altra parte, il contesto in cui viene effettuato il riuso determina spesso anche il valore aggiunto portato dal riuso, a volte compensando eventuali vincoli dovuti a una scarsa portabilità del software: ad esempio, può darsi il caso per cui un determinato software sia utilizzabile solo con uno specifico sistema operativo o middleware, di larghissima diffusione nella Pubblica Amministrazione, per il quale la P.A. ha in essere contratti o accordi quadro con il produttore, e che perciò il suo riuso possa risultare molto vantaggioso anche se il software stesso non è portabile. Così come un software potrebbe essere portabile in molti contesti ma avere scarsa diffusione nella P.A. e quindi avere un costo unitario alto e rendere pertanto il riuso basato su tale prodotto poco efficiente dal punto di vista economico.

3.2 I FATTORI ABILITANTI IL RIUSO

3.2.1 FATTORI TECNICI

L'esperienza ha dimostrato che i principali fattori tecnici che influenzano il riuso sono:

1. la compatibilità architetturale tra l'ambiente di origine e quello in cui verrà riusato il software;

2. la compatibilità tra i requisiti utente nei domini applicativi cedente e ricevente;
3. le caratteristiche tecniche del software sviluppato;
4. la completezza della documentazione progettuale, che deve facilitare l'adattabilità, l'evoluzione e la personalizzazione del prodotto;
5. la metodologia adottata per lo sviluppo, che deve essere basata su un processo di produzione orientato al riuso.

I primi due fattori dovrebbero essere valutati come presupposto al riuso di un prodotto software, in quanto ne possono determinare il maggiore o minore costo e la fattibilità.
Anche il terzo fattore, le caratteristiche tecniche del software, deve essere attentamente valutato per capire il costo e la facilità di poter effettuare il riuso. In questo documento vengono definiti gli attributi base che deve possedere un software riusabile, a prescindere dal contesto di riuso. Questi attributi dovrebbero essere sempre inseriti tra i requisiti associati al software in ogni nuovo progetto di sviluppo che si proponga di realizzare componenti riusabili.
Gli ultimi due fattori aumentano la efficacia ed efficienza del riuso, e dovrebbero essere fissati come requisiti "di processo" per i fornitori dai committenti nei progetti di sviluppo di nuovo software. Va osservato qui che lo standard ISO/IEC 9126 *Software engineering – Product quality* afferma che la documentazione associata al prodotto software (di progetto, d'uso, di manutenzione, di gestione) fa parte integrante del prodotto. È quindi sempre buona norma fissare dei requisiti anche per tale documentazione nei contratti di acquisizione di prodotti software, siano essi *custom* che COTS.
I fattori 1 e 2 vengono qui di seguito brevemente trattati, rimandando alle linee guida pubblicate dal CNIPA nel 2005 per una ulteriore trattazione. I fattori 3, 4 e 5 sopra individuati verranno approfonditi nei successivi capitoli di questo documento, come elementi base della strategia per il riuso qui esposta, fornendo anche delle sintetiche indicazioni progettuali per la produzione di software in grado di soddisfare i requisiti base di riusabilità.

3.2.2 Compatibilità dei requisiti tra dominio cedente e ricevente

Una delle maggiori difficoltà che si incontra nei processi di sviluppo, almeno per quanto riguarda lo svolgimento delle cosiddette attività di alto livello (analisi del dominio, analisi dei requisiti, progettazione architettonica), risiede nella mancanza di una terminologia comune e condivisa.
Ne deriva un "rischio" progettuale, ampiamente documentato dall'ingegneria dei requisiti, che dipende dall'aver attribuito allo stesso concetto nomi diversi o dall'aver usato lo stesso termine con diverse accezioni, in documenti diversi o addirittura nello stesso documento di analisi.
Questa eterogeneità nell'uso dei vocabolari di domini diversi può dar luogo a descrizioni errate dei sistemi, che possono indurre a valutazioni nelle quali le analisi del dominio sembrano apparentemente uguali, ma sono di fatto profondamente diverse.
È intuitivo che, dal punto di vista del riuso, questo problema terminologico assume una rilevanza molto particolare, in quanto aumenta oltre misura lo sforzo di analisi e integrazio-

ne di applicazioni provenienti da domini diversi, soprattutto nel caso di applicazioni custom. Sull'argomento, il CNIPA ha pubblicato un "Quaderno" che ha l'obiettivo di rendere più omogenea la terminologia da adottare nel settore informatico, in modo da uniformare il più possibile i bandi di gara, i concorsi e, in generale, tutti i documenti relativi a questo settore.

3.2.3 COMPATIBILITÀ ARCHITETTURALE E TECNOLOGICA

L'uso di più tecnologie nel contesto dell'informatizzazione della Pubblica Amministrazione è un fatto inevitabile, in quanto la tecnologia tende naturalmente ad evolversi e diversificarsi e sono numerosi i fornitori di tecnologia che operano sul mercato, tra loro in concorrenza, ma spesso anche auspicabile, in quanto garantisce la pluralità delle soluzioni e la possibilità di scelta dell'acquirente, consentendoli di acquisire ciò che meglio si adatta al contesto.

L'eterogeneità tecnologica, tuttavia, produce una vasta gamma di difficoltà tecniche dovute essenzialmente ad aspetti di incompatibilità che si manifestano a vari livelli, nel caso di riuso del software:

1. *Linguaggio di programmazione*. Linguaggi di programmazioni diversi ostacolano il riuso a livello di codice sorgente. Anche se è possibile sviluppare un progetto utilizzando moduli già sviluppati e scritti in linguaggi di programmazione diversi, questa soluzione aumenta, in termini assoluti, la complessità del sistema software. Tuttavia, lo sviluppo di software basato su diversi linguaggi di programmazione è oggi possibile su alcune piattaforme che supportano tale possibilità, come, ad esempio, Microsoft.Net, che permette l'utilizzo di più tools di sviluppo (C++, C#, Visual Basic) nella stessa applicazione.
2. *Librerie esterne*. L'uso di librerie esterne diverse per fornire le stesse funzionalità è di fatto un ostacolo al riuso. Un esempio di queste difficoltà si ritrova negli ambienti grafici in ambiente Linux (OpenMotif, Gnome, KDE), nei parser XML in Java, e così via. Il riuso di un componente che usa una di queste librerie all'interno di un'applicazione in cui ne viene utilizzata una diversa è molto difficile. Una soluzione che mitiga questo problema consiste nella definizione di interfacce astratte che permettono di gestire le dipendenze da queste librerie, ma questo approccio ha l'effetto collaterale di aumentare la complessità totale del sistema.
3. *Ambiente operativo*. Sistemi operativi diversi mettono a disposizione API diverse e, parzialmente, anche categorie di funzionalità diverse (ad esempio, la possibilità o meno di gestire applicazioni multi-threaded). Per le funzionalità più comuni, in linea generale è possibile operare per svincolarsi dalle specifiche API attraverso l'uso di opportuni Adapters o framework di emulazione.
4. *Strati applicativi*. Comunemente usate per lo sviluppo di applicazioni complesse (piattaforme di sviluppo ed esecuzione, application server, database management systems) ogni piattaforma mette a disposizione primitive diverse e molto spesso impone

modelli architetturali differenti. In alcuni casi, le piattaforme si riferiscono a standard (che molto spesso sono solo "di fatto"), come nel caso degli *application server* che si riferiscono a Java 2 Enterprise Edition (J2EE) di Sun o nel caso dei database che realizzano il modello relazionale e sono accessibili attraverso API standardizzate (es. JDBC). Anche quando si fa riferimento a modelli comuni, tuttavia, è normale riscontrare situazioni in cui si devono sfruttare specificità proprietarie del prodotto discostandosi dallo standard. Ciò può verificarsi per svariati fattori, ad esempio per una specifica non sufficientemente precisa dello standard (ogni produttore è portato a implementarlo in modo leggermente diverso facendo sì, ad esempio, che la medesima sequenza di chiamate alla stessa API fornisca risultati diversi se eseguita su due piattaforme diverse) o per un'eccessiva limitatezza dello standard che, particolarmente quando non è sufficientemente consolidato, non copre determinate funzionalità (come, ad esempio, nelle prime specifiche di J2EE in cui non era specificato come gestire le associazioni fra componenti EJB).

In conclusione, l'eterogeneità tecnologica deve essere opportunamente considerata nel momento in cui si valuta l'opportunità del riuso di un'applicazione software, in quanto può generare costi aggiuntivi nella gestione e manutenzione dell'applicazione. Analogamente, va ovviamente valutata attentamente l'opportunità di sviluppare applicazioni software basate su più tecnologie tra loro eterogenee, in quanto, se si desidera in futuro riusare queste applicazioni può essere più complesso garantire la loro interoperabilità e installabilità in nuovi contesti.

3.3 Riuso e architetture software

Come detto in precedenza, queste linee guida si propongono di contribuire all'aumento dell'efficienza del riuso favorendo a regime il riuso di singoli componenti software, anziché solo di interi sistemi applicativi. Per comprendere il livello di granularità al quale può essere vantaggiosamente collocato il riuso è significativo individuare preliminarmente quali sono gli elementi che compongono di norma l'architettura di un sistema applicativo e individuare a quale livello di aggregazione questi elementi potrebbero essere singolarmente riusati.

Introduciamo quindi il concetto di "architettura" software. Un'architettura software è una particolare configurazione di tutti gli elementi che compongono un sistema software.[3]

L'architettura di un sistema software può essere vista come una "pila" formata da più livelli (o strati) "logico-funzionali", in ognuno dei quali sono raggruppati dei "componenti"

[3] È questo un concetto esteso di configurazione software, generalmente riferita alla sola organizzazione dei moduli di codice sorgente, che diventa, secondo tale visione, peraltro coerente con lo standard ANSI/IEEE 1471-2000, una parte dell'architettura più generale del sistema software da riusare. Il riuso è infatti più agevole se tutti gli elementi che compongono un sistema software (e non solo il codice) sono stati progettati per essere riusati.

software che forniscono una tipologia omogenea di funzioni specializzate (o servizi), destinate ai livelli superiori. La pila degli strati è detta spesso "stack".
Per "componente" si intende un elemento software che realizza funzioni specifiche (ha un ruolo preciso nell'architettura), ha una propria autonomia funzionale con un input e un output definiti ed è in grado di comunicare con altri componenti. Un componente può essere un modulo, un'aggregazione di più moduli (ad esempio, un sottosistema o un servizio applicativo) o un intero sistema software. Dal punto di vista tecnico, un componente può essere visto come un "circuito integrato" software che comunica con l'esterno attraverso una serie di "piedini".
Negli strati più bassi della pila sono inclusi i sistemi operativi, i software di base o anche componenti hardware. Sopra questi vi sono dei componenti di interfaccia che mettono in contatto questi strati con quelli dove viene realizzata la logica funzionale specifica del sistema software. In questi strati di livello più alto sono posizionati i "business components", che possono essere divisi tra componenti standard (o semi standard) di mercato, che assicurano funzioni di interesse generalizzato per una larga fascia di utenti e componenti che, eventualmente poggiandosi sui precedenti, svolgono funzioni specifiche per una data classe di utenti.
Chiameremo questi due strati di business livelli "abilitante" e "funzionale.
Riepilogando:

- il livello *Base* della pila logico funzionale di un sistema software è costituito da componenti privi di logica funzionale (applicativa), ad es. i Sistemi Operativi (OS), Application Server (AS), Database (DB), i Middleware;
- il livello *Abilitante* comprende componenti che forniscono funzioni generiche utili a diversi fini applicativi (ad esempio, sistemi di directory, quali LDAP; applicazioni non verticalizzate, quali ERP/CRM, sistemi di Business Intelligence etc);
- il livello *Funzionale* comprende componenti sviluppati su specifici requisiti del committente.

La successiva figura riporta una tipica pila di un sistema software, posto su uno strato hardware.

Figura 1 – Schema di architettura a livelli di un sistema software

In qualsiasi strato della pila non c'è una regola fissa per definire la "dimensione" dei componenti o lo "spazio" applicativo che coprono (l'insieme di servizi che offrono). In ingegneria del software si sottolinea l'importanza che tali componenti siano tra loro indipendenti (si parlino attraverso interfacce) e siano molto coesi internamente (siano focalizzati nel fornire un servizio specifico) e poco accoppiati tra loro (per dare il servizio non debbano necessariamente ricorrere anche ad altri componenti).
I componenti possono essere sviluppati ad hoc, su requisiti di uno specifico committente (e sono detti "custom") o essere prodotti di mercato, detti COTS, Commercial Off The Shelf.[4]
Anche se non si può generalizzare, si rileva come negli strati bassi dell'architettura di un sistema software siano spesso presenti in prevalenza prodotti COTS (a volte *open source* anziché proprietari) mentre negli strati più alti è maggiore la percentuale di prodotti *custom*. In effetti, questa distribuzione risponde alla diversa specializzazione propria di ciascun strato: quelli più alti rispondono ad esigenze specifiche di particolari utenti, quelli più bassi forniscono funzioni generalizzate, standardizzabili e "pacchettizzabili".
In ogni livello possono essere fatti cooperare componenti COTS e *custom*.
A volte, i componenti COTS, per poter essere utilizzati nella pila, devono essere preventivamente parametrizzati e/o personalizzati, per soddisfare specifiche esigenze di un dato cliente. Queste parametrizzazioni e personalizzazioni possono essere in parte considerate a loro volta un prodotto *custom*.
Si deve rilevare come gli strati "abilitante" e "funzionale" dello stack dell'architettura di un sistema software possano essere ulteriormente suddivisi in 3 sotto strati, con riferimento alla tipologia di funzioni che i componenti collocati in questi strati offrono: funzioni di "infrastruttura", ""processo", "servizio". Ognuno di questi tre sotto-strati ha necessità del sottostante per funzionare.
La figura che segue rappresenta questo schema logico funzionale, con riferimento al contesto del riuso.

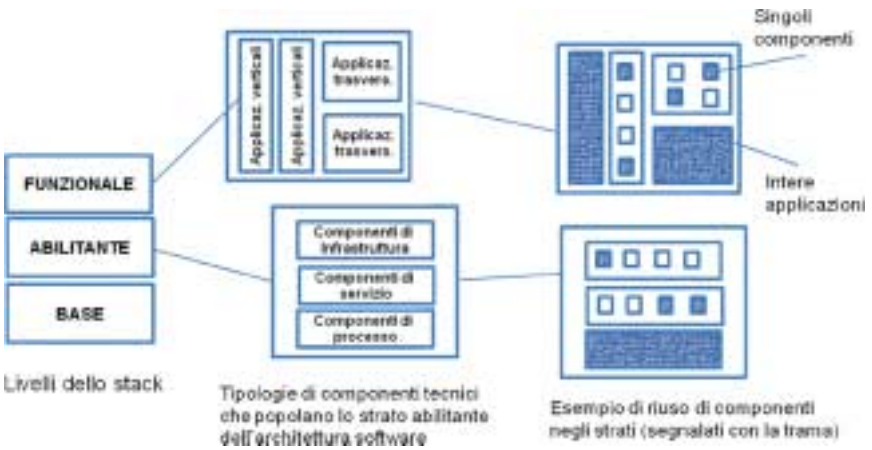


Figura 2 – Generalizzazione della modularità di una architettura software e contesto del riuso

[4] Per i prodotti COTS si fa riferimento allo standard ISO/IEC 12119 *Software Engineering - Software product evaluation - Requirements for quality of Commercial Off The Shelf software product (COTS) and instructions for testing.*

Ai fini del riuso, è importante che il livello Abilitante e quello Funzionale siano realizzati mediante tecnologie e metodologie che ne garantiscano l'isolamento logico-funzionale.
Un progettista di sistemi software oggi dispone di molti componenti già sviluppati da riusare (sia prodotti proprietari che open source), in grado di fornire servizi elementari specifici e ricorrenti. Più in dettaglio, i tre sotto-livelli di una architettura software sopra citati possono comprendere:

1. componenti software che realizzano *funzioni di infrastruttura*, quali, ad esempio:
   a. autenticazione, autorizzazione e accesso,
   b. tracciatura e log,
   c. firma (elettronica o digitale),
   d. pagamento elettronico,
   e. monitoraggio di applicazioni e processi,
   f. posta elettronica (anche certificata),
   g. gestione documentale a norma CNIPA (tra cui il protocollo informatico);

2. componenti software che realizzano *funzioni di processo* quali, ad esempio:
   a. il processo che governa l'emissione di un certificato di residenza,
   b. il processo di approvazione di una richiesta di acquisto,
   c. il processo che governa la gestione dello svolgimento di una gara di appalto on-line;
2. componenti software che realizzano *funzioni di servizio*, quali, ad esempio:
   a. gestione del personale,
   b. contabilità analitica o economica,
   c. gestione degli acquisti o degli asset,
   d. erogazione di servizi al cittadino e alle imprese.

3.4 IL RIUSO NEI PROCESSI PRODUTTIVI DEL SOFTWARE E NELLE STRATEGIE DI ACQUISTO DI SOFTWARE

Il riuso, come pratica per migliorare l'efficienza del processo di produzione del software, è ben noto da diversi anni alle organizzazioni produttrici di software, che vi hanno fatto sempre largamente ricorso come fattore di competitività economica e qualitativa. Infatti, il riuso:

- riduce la dimensione del software da sviluppare – che rappresenta il maggior *cost driver* dello sviluppo – diminuendo quindi il costo di produzione,[5]

[5] La riduzione del costo non è in proporzione lineare con la quantità di software riusato rispetto alle dimensioni di quello da sviluppare. Una quota di progettazione rimane comunque necessaria, per decidere cosa riusare e come inserire il software riusato nel nuovo sistema da sviluppare. Questa considerazione è ripresa nel noto metodo per la stima dei costi del software Constructive Cost Model (COCOMO II), nella formula degli *Early prototyping* e *Post architecture* model in CoCoMo II.

- riduce i tempi di sviluppo,
- migliora – progressivamente – la qualità di quanto prodotto.

Il riuso di semilavorati per velocizzare il processo produttivo è una pratica diffusa in tutti i processi di produzione industriali. Nel caso della produzione di software, il riuso dovrebbe essere, in teoria, particolarmente efficiente, in quanto i semilavorati software non soffrono dei vincoli fisici degli altri materiali: non devono essere stoccati in attesa del riuso, possono essere adattati e modificati a piacere e con sforzo contenuto, sono in genere adattabili e integrabili in varie architetture, sono replicabili infinite volte etc..
Le architetture logico funzionali dei sistemi software, d'altra parte, fin dai tempi dell'affermarsi della progettazione strutturata sono fatte di moduli, che svolgono funzioni specifiche o compiti elementari e/o ripetitivi, che sono generalizzabili, nel senso che le si ritrova in quasi tutti i prodotti. Questo concetto è alla base delle routines, dei moduli, delle librerie di servizi applicativi ma anche dei moderni *patterns* che aiutano l'opera dei progettisti.
Le odierne architetture applicative "Service Oriented" (dette architetture SOA), sono state ideate per realizzare servizi complessi attraverso l'assemblaggio di componenti elementari già esistenti, che possono essere anche tecnologicamente eterogenei.
In questi anni, d'altro canto, molte delle tecnologie di sviluppo software si sono predisposte a favorire il riuso: la progettazione e programmazione object-oriented e per componenti, i linguaggi di programmazione che lavorano a oggetti, tutto ciò che oggi esiste per "incapsulare" e rendere coesi e poco accoppiati i moduli che compongono un software, va nella direzione di agevolare il riuso.
Come detto, il riuso non consente solo vantaggi economici: un altro elemento qualificante il riuso è la capacità di migliorare progressivamente la qualità del software riusato. Infatti, la affidabilità dei componenti già sottoposti ad uso operativo è ovviamente maggiore rispetto a quelli realizzati *ex novo* ed ancora da "provare" sul campo.
Anche i clienti, in fondo, riusano da sempre il software. Per l'acquirente, ovviamente, il riuso ha un significato in parte diverso da quello del produttore. In genere, gli acquirenti, se devono acquisire un nuovo software, valutano queste possibili forme di approvvigionamento: lo sviluppo ex novo del software (*custom*), il riuso di un software già esistente (previo eventuale adattamento), un prodotto proprietario già pronto (da pagare di solito a licenza d'uso o a copia), un software *open source* già pronto.
Vale la pena di osservare che la scelta della soluzione dovrebbe essere effettuata tenendo conto, tra l'altro, di diversi fattori economici, che sono sinteticamente ricordati nella tabella che segue nella pagina successiva.
La valutazione deve essere analitica, in quanto non necessariamente ricorrere a un software di larga diffusione, ancorché proprietario, è meno vantaggioso rispetto al riuso: i prodotti COTS (i c.d. Commercial Off The Shelf) hanno infatti dei costi unitari relativamente bassi (il cliente paga solo una quota minima dei costi effettivamente sostenuti per lo sviluppo del prodotto, grazie alle economie di scala generate dalla sua larga diffusione); sono disponibili da subito (un software *custom* richiede tempo per essere progettato e poi realizzato); inoltre, questi software sono in genere affidabili, vengono fatti evolvere continuamente per

Tipo di approvvigionamento	Tipologia di costi
Sviluppo custom	Costo dello sviluppo del software (Analisi e Specifica requisiti, Progettazione tecnica, Codifica, Test e Integrazione)
	Eventuale costo delle componenti proprietarie utilizzate dal software custom (RDBMS, Middleware, Componenti specializzati etc..)
	Costo manutenzione del software dopo il rilascio
	Costo dell'assistenza all'uso
Riuso software già esistente	Costo del riuso (Analisi requisiti, adattamenti e personalizzazioni, Test e Integrazione)
	Eventuale costo delle componenti proprietarie utilizzate dal software custom (RDBMS, Middleware, Componenti specializzati etc..)
	Costo di manutenzione del software dopo il rilascio
	Costo dell'assistenza all'uso
Prodotti proprietari	Costo delle licenze e/o delle copie del prodotto
	Costo di manutenzione del software dopo il rilascio
	Costo dell'assistenza all'uso
Prodotti open source	Costo di manutenzione del software dopo il rilascio
	Costo dell'assistenza all'uso

rimanere in linea con lo stato dell'arte della tecnologia, sono serviti da una catena di assistenza, hanno dei costi contenuti (speso a forfait) per la manutenzione e l'upgrade.[6]
Se si decide di ricorrere al riuso di un software *custom* vi sono poi degli ulteriori elementi che potrebbero ridurre l'efficienza del riuso:

a) i requisiti di ogni nuovo progetto di sviluppo software applicativo (che realizza funzioni utili nel dominio specifico del cliente) sono di norma in buona parte diversi da quelli dei progetti precedenti, e pertanto è quasi sempre difficile poter riusare un'intera applicazione sviluppata per soddisfare altre esigenze. I costi di adattamento e personalizzazione del software da riusare al nuovo contesto potrebbero essere molto alti;

b) spesso le applicazioni software disponibili per il riuso sono state sviluppate non interamente ad hoc, ma integrano sviluppi *custom* con personalizzazioni di prodotti

[6] In merito all'utilizzo dei prodotti COTS per realizzare funzioni applicative per la P.A. è utile aggiungere qui una precisazione. Una strategia che intenda favorire il riuso del software non può prescindere dal contesto di maturità e diffusione delle tecnologie tramite le quali le applicazioni software sono o possono essere realizzate. Tale contesto comprende non solo un eterogeneo insieme di tecnologie, anche molto diverse tra loro, che stanno convergendo verso alcuni aspetti comuni (quali l'interoperabilità "a servizi" basata sul modello SOA, ma anche una vasta gamma di sistemi software (tra cui i prodotti COTS) realizzati e distribuiti da fornitori nazionali o internazionali. I prodotti COTS, e le tecnologie di base soggette a licenza d'uso e prodotte da fornitori nazionali o internazionali, hanno spesso caratteristiche di qualità elevate e sono dotati di funzionalità molto avanzate che vengono ereditate dalle soluzioni basate su di esse, con possibili vantaggi in termini di qualità della soluzione complessiva. Il loro uso può, in determinate condizioni, abbattere i costi di realizzazione delle soluzioni applicative, e contribuire alla realizzazione degli obiettivi di contenimento della spesa pubblica.

COTS e con funzioni offerte direttamente da prodotti COTS. In molti casi, questi strati di software non sono tra loro indipendenti, così che per riusare una applicazione *custom* il soggetto riusante deve acquistare le licenze d'uso dei prodotti COTS su cui il software *custom* si poggia.

Inoltre, non bisogna sottovalutare l'aspetto della rapida obsolescenza del software. L'evoluzione delle tecnologie di base rende spesso le soluzioni applicative in breve tempo inutilizzabili e inefficienti, talvolta semplicemente "fuori moda". Riadattare un software di 10 anni fa (ma anche solo di 5 anni fa) per riusarlo oggi, ad esempio, potrebbe essere estremamente inefficiente e costoso.

In definitiva, il riuso di una applicazione software già esistente deve partire da una analisi approfondita dei costi da sostenere per il riuso e della sua convenienza anche in prospettiva, nel tempo.[7]

L'esperienza ha dimostrato che per aumentare la convenienza della scelta del riuso si dovrebbe sia disporre di più software da riusare, sia di software le cui caratteristiche costruttive abbattono i costi del riuso (sostanzialmente quelli di analisi e specifica dei requisiti, parametrizzazione e personalizzazione, licenze d'uso di prodotti COTS necessari per far funzionare il software da riusare).

Per aumentare la disponibilità di software da riusare, almeno nel contesto della Pubblica Amministrazione centrale dove molte sono le applicazioni "verticali", abbastanza monolitiche e progettate per esigenze specifiche poco esportabili in altri contesti, si dovrebbe prevedere di aumentare progressivamente il livello di granularità del riuso, nel senso di prevedere di riusare non solo intere applicazioni ma anche singoli moduli, che svolgono funzioni generalizzabili o generalizzabili con poco sforzo.

Naturalmente, perché ciò sia possibile, è necessario che le pubbliche amministrazioni dispongano od abbiano accesso a librerie di componenti funzionali riusabili, dotati di specifiche caratteristiche tecniche che ne agevolino la riusabilità in altri contesti, eventualmente in parte noti a priori.

In definitiva, per aumentare l'efficienza del riuso nella P.A. ed in particolare in quella centrale (senza trascurare comunque le possibili ricadute nel riuso tra P.A. centrale e locale), sarebbe utile che le pubbliche amministrazioni chiedessero ai propri fornitori di progettare e sviluppare le nuove applicazioni software *custom* – laddove possibile in ragione del contesto tecnico e organizzativo del progetto – come assemblaggio di componenti (moduli) preesistenti e di sviluppare i nuovi moduli necessari a completare l'architettura del sistema in maniera tale che tali moduli siano a loro volta singolarmente riusabili, in quanto dedicati a svolgere compiti generalizzabili o comunque generalizzabili con piccoli adattamenti (componenti "multiuso", nel senso che possono essere usati in vari contesti).

In pratica, così facendo, le amministrazioni si troverebbero ad acquisire, al termini di ogni progetto di sviluppo, sia l'applicazione software richiesta con la specifica commessa, sia un

[7] Sarebbe in tal senso opportuna una analisi che utilizzi come parametro il Total Costo of Ownership (TCO) che considera anche costi che si producono nel tempo, dopo la conclusione dello sviluppo.

certo numero di nuovi semilavorati riusabili. Con tali componenti le amministrazioni potrebbero (come già fanno del resto i produttori di software) costruire un catalogo di propri "componenti" riusabili, da utilizzare come mattoni elementari per costruire altro nuovo software *custom* nei loro futuri progetti. Si avvierebbe in pratica un circolo virtuoso con benefici sia economici, sia nella qualità del software.
Si intende che questa particolare modalità di sviluppo di software *custom* va ad aggiungersi (e non le elimina) all'elenco delle opzioni che una Amministrazione deve valutare al momento di decidere dell'acquisto di un nuovo software (Make or Buy or Reuse).
I costi specifici di questa modalità di sviluppo che vanno valutati sono:

- costi di ricerca, selezione e adattamento dei componenti riusabili già esistenti;
- costi aggiuntivi per sviluppare i nuovi componenti riusabili, la cui realizzazione richiede una certa cura;
- costi di sviluppo del software progressivamente sempre più bassi, in quanto cresce il numero di componenti riusabili di cui si dispone e diminuisce il costo del loro riuso e adattamento in altri progetti.

Le amministrazioni solo "riusanti" avrebbero invece fin da subito costi di riuso inferiori, grazie alla maggiore facilità di adattamento dei componenti sviluppati per essere riusabili.
Dovrebbe essere poi superfluo sottolineare come questo tipo di strategie abbia una valenza che va oltre i contorni delle singole amministrazioni, essendo mirata alla razionalizzazione della pratica di sviluppo del software nell'intera Pubblica Amministrazione, vista come un unico acquirente logico. Nel successivo capitolo verrà sinteticamente tratteggiata la situazione del riuso nella Pubblica Amministrazione, in modo da fornire alcuni dati quantitativi di partenza per meglio inquadrare i problemi da risolvere e la soluzione proposta.

4. Elementi di novità del riuso

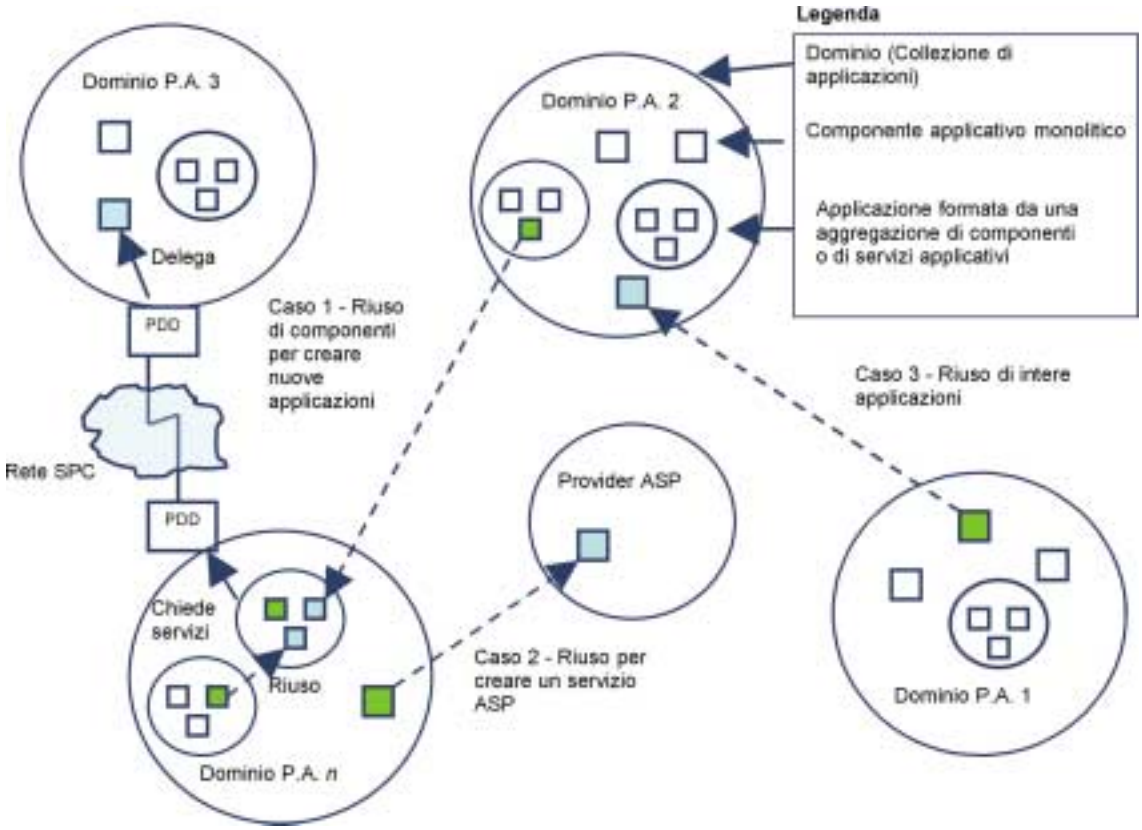
4.1 *Web Services* e cooperazione applicativa

Il riuso di software è una forma di cooperazione tra soggetti diversi (il cedente e il riusante). In senso tradizionale, prevede un trasferimento "fisico" di un componente software da un soggetto all'altro, il suo adattamento e la sua installazione nell'ambiente di destinazione. Questa modalità di cooperazione tra soggetti, per quanto efficace, comporta dei costi legati all'adattamento, installazione e gestione nel nuovo ambiente dei componenti riusati.
Una nuova forma di cooperazione tra amministrazioni pubbliche, nella quale è possibile anche prevedere forme di riuso, è il ricorso a *shared services* (anche detti servizi erogati in modalità ASP – Application Service Provider) e ai *web services*.
Nel caso degli *shared services*, un soggetto (eventualmente terzo rispetto alle amministrazioni utilizzatrici) eroga un servizio (fa da "provider") mettendo a disposizione via rete TLC le funzioni che gli utenti richiedono. L'allestimento del servizio (nelle componenti applicative e tecnologiche, quindi incluso il data center) è a carico di questo provider. Le amministrazioni che aderiscono al servizio pagano un canone per il suo utilizzo, ma non hanno costi diretti per l'acquisto, gestione e manutenzione di componenti applicative e tecnologiche.
Nel caso di *web services*, diversi soggetti, collegati da una rete TLC, possono creare un *workflow* applicativo, nel quale i vari passi del processo elaborativo sono affidati a diverse applicazioni, residenti nei domini dei soggetti stessi. Le applicazioni si scambiano i dati attraverso protocolli e interfacce. In tal modo, i soggetti partecipanti al *workflow* possono cooperare senza dover affrontare interventi di omogeneizzazione tecnologica, ricorrendo a opportune tecniche di disaccoppiamento tra servizi informatici e tecnologia con la quale sono realizzati. Tutti i soggetti coinvolti nel processo elaborativo distribuito realizzato attraverso i *web services* devono provvedere a creare e gestire le proprie risorse elaborative (e.g. un data center, o dei server), ma possono dimensionarle per coprire solo la quota di processo applicativo di loro competenza.
Fattore abilitante l'uso di *shared services* e *web services* è la disponibilità di una rete di connettività adeguata e di connessi servizi per l'interoperabilità, che assicurino la sicurezza, efficienza e affidabilità degli scambi di informazioni e di servizi in rete. Questa rete è l'obiettivo dell'iniziativa "SPC" (Sistema Pubblico di Connettività) attuata dal CNIPA, che ha messo a disposizione delle amministrazioni pubbliche un ampio insieme di servizi di connessione e di supporto all'IT di alta qualità e a costi contenuti. SPC comprende anche un'offerta di ser-

vizi di interoperabilità e cooperazione applicativa per le amministrazioni, basati su un paradigma logico funzionale denominato *SPCoop*.
Lo scenario tecnologico ed organizzativo sopra descritto non fa venir meno l'esigenza del riuso e la sua efficienza. Infatti, il riuso può servire per creare nuove applicazioni – o parti componenti di esse – da esporre come servizio web sulla rete ovvero per creare nuovi *shared services*.
Questo scenario è rappresentato nella successiva figura, che riporta uno scenario operativo di riferimento, articolato su più entità organizzative che sviluppano e/o espongono in rete componenti applicative. In particolare, sono evidenziati quattro domini che comunicano tra loro mediante la rete SPC, utilizzando come interfaccia delle apposite porte di dominio. Tra i domini sono indicate altre relazioni che esemplificano gli aspetti principali del riuso.

- Una prima forma di riuso è costituita dal riuso di un'intera applicazione (rappresentata in figura da un rettangolo). L'applicazione è replicata integralmente dal dominio in cui è stata realizzata nel dominio in cui è riusata. Questa modalità di riuso è la più semplice e, per alcuni versi, la più efficace in quanto l'intera applicazione è riusata in un altro dominio senza dover operare modifiche sostanziali. È applicabile tipicamente quando l'intera applicazione supporta un processo specifico replicato tra diverse amministrazioni.
- La seconda forma di riuso si ha quando un componente che fa parte di un'applicazione complessa che aggrega più componenti, viene riusato per la realizzazione di un'altra applicazione. In questo caso l'aspetto più significativo è la capacità di estrarre un componente da un'aggregazione e di inserirlo in un'altra, a costi inferiori rispetto a quelli necessari per una sua scrittura *ex novo*. La nuova applicazione può appartenere sia al dominio che ha sviluppato originariamente il componente, sia ad un altro. Questa forma di riuso è applicabile quando due applicazioni prevedono nelle loro architetture logico funzionali componenti che svolgono compiti simili (spesso di "servizio"). Un caso tipico è quello di componenti di natura "infrastrutturale".
- La terza forma di riuso prevede l'inserimento di un componente software all'interno di un sistema applicativo messo a disposizione di amministrazioni clienti da un fornitore di servizi in modalità ASP. In questo caso, i soggetti che intendono usufruire dei servizi messi a disposizione dal fornitore vi accedono via web, anche attraverso la sua porta di dominio, senza dover conoscere i dettagli realizzativi del componente software che utilizzano. Per molti servizi della Pubblica Amministrazione, specie di "back office" o comunque di interesse generalizzato, questa soluzione è la più efficiente e la più efficace. Questo caso è tipico quando il processo è gestito direttamente da un soggetto esterno all'Amministrazione richiedente, per motivi che possono essere normativi, organizzativi, o di convenienza.

Ovviamente, l'efficienza del riuso in questo scenario dipende dalla disponibilità di software riusabile nei cataloghi delle pubbliche amministrazioni che cooperano in rete. Come verrà



approfondito in questo documento, l'efficienza del riuso può essere massimizzata dall'adozione di un diverso approccio allo sviluppo del software *custom* nella P.A., finalizzato a produrre sistemi software fatti di componenti anche singolarmente riusabili in successivi progetti. Componenti quindi modulari, coesi, tra loro scarsamente accoppiati, in grado di collaborare con altri componenti – anche tecnologicamente eterogenei – utilizzando i classici principi della progettazione a oggetti. Vale la pena di osservare, poi, che i componenti riusabili non sono solo moduli di codice sorgente, ma anche patterns e schemi progettuali, templates di documentazione etc..

Tutto questo materiale, per poter essere riusato, deve essere visibile nella rete della P.A. e va quindi pubblicato in appositi cataloghi, interrogabili in modo anche applicativo, in cui i componenti contenuti o referenziati sono descritti con linguaggi standard (e.g. XML e le specifiche OMG per la descrizione di asset riusabili). Questi cataloghi devono essere connessi in una rete e devono essere, logicamente, un'unica fonte di informazione per chi deve decidere se e cosa riusare per sviluppare nuove applicazioni software.

In questa visione della Pubblica Amministrazione in rete, ogni Amministrazione può quindi contribuire a una "fabbrica" del software distribuita, che produce software riutilizzabile per il contesto dell'intera P.A. Questo software potrà essere riutilizzato sia dalla stessa Amministrazione che l'ha originariamente sviluppato sia da altre, come componente di una nuova applicazione.

4.2 COMPONENTI SOFTWARE MULTIUSO

Il concetto di software "multiuso" è recente, ed è l'evoluzione della pratica "industriale" di sviluppo di software come assemblaggio di componenti semilavorati già pronti.
Lo sviluppo di software come assemblaggio di componenti già esistenti trova il suo fondamento nella concettualizzazione dell'architettura logica di un sistema software, già ipotizzata con la progettazione strutturate e la progettazione object oriented.
Secondo tali principi una architettura software può essere schematizzata come un insieme di "moduli", ognuno dedicato a realizzare una funzione (o parte di una funzione). Più funzioni aggregate realizzano un "servizio" per un utente esterno. In una architettura i singoli moduli non sono monadi isolate, ma interagiscono per realizzare servizi, scambiandosi dei messaggi (dei dati), secondo un determinato protocollo condiviso. Ogni modulo è però una *black box*, nel senso che i dettagli tecnologici con i quali è stato sviluppato non hanno importanza ai fini del suo funzionamento nell'architettura.
Le recenti architetture "SOA" (Software Oriented Architecture) hanno fatto evolvere questo schema concettuale, ponendo l'accento sui servizi che i componenti possono realizzare e sulla costruzione di nuovi sistemi software come aggregazione di componenti funzionali elementari, in parte già disponibili in specifiche librerie.
Va da sé che l'efficienza di questa modalità di costruire sistemi software aumenta con l'aumentare del numero di componenti funzionali già pronti all'uso a disposizione del progettista dell'architettura del sistema. Più sono questi componenti, minore sarà il numero di funzioni per realizzare le quali dovranno essere sviluppati ex novo dei componenti software e, quindi, minore sarà il costo di sviluppare il sistema e il tempo necessario allo sviluppo. Nello stesso tempo, l'efficienza del riuso dei componenti aumenta se essi devono subire pochi adattamenti per essere inseriti nella nuova architettura.
Lo sviluppo per componenti può dare grande efficienza alla progettazione e realizzazione di sistemi software: se è vero infatti che in ogni nuovo sistema alcune funzioni sono destinate a soddisfare esigenze specifiche che non sono facilmente ritrovabili uguali in altri contesti, in qualsiasi sistema software esiste anche un buon numero di funzioni assolutamente generalizzabili e replicabili con contenuti adattamenti da un contesto ad un altro.
Se nella pratica della produzione industriale del software lo sviluppo per assemblaggio di componenti si sta ormai affermando, non così è ancora nella Pubblica Amministrazione italiana, che ogni anno commissiona una ingente quantità di software *custom* ma, finora, non ha ancora sfruttato a sufficienza le potenzialità delle architetture di componenti.
La Pubblica Amministrazione sta da tempo esplorando le potenzialità del riuso del software, ma si è finora concentrata sul riuso di intere applicazioni. Ciò ha diminuito di molto l'efficienza potenziale del riuso in quanto la maggior parte dei software *custom* della P.A. sono stati sviluppati per servire esigenze specifiche verticali e sono quindi riusabili solo in contesti simili a quelli originali, oppure devono subire ingenti lavori di adattamento per essere riusati. Perciò, il riuso nella Pubblica Amministrazione, quand'anche praticato, riesce a recuperare solo parte delle funzioni del software riusato, con alti costi di adattamento.

Questo problema è per la verità meno sentito nel caso delle amministrazioni locali, che sono in numero molto più elevato delle amministrazioni centrali e nelle quali le esigenze similari sono molto più numerose. Il riuso di intere applicazioni, nel loro caso, riesce a essere abbastanza efficiente e a generare economie di scala. Semmai, si deve rilevare come, per i motivi sopra ricordati, sia difficile il passaggio di software dalle amministrazioni centrali a quelle locali, o viceversa, a meno dei soliti alti costi di adattamento.
In definitiva, allo stato attuale, si deve osservare come, soprattutto nel caso di riuso tra pubbliche amministrazioni centrali e nel caso di riuso tra pubbliche amministrazioni centrali verso locali, o viceversa, il riuso non sia ancora una pratica sufficientemente efficiente. E ciò a causa, in buona parte, delle caratteristiche con le quali i software *custom* destinati alla Pubblica Amministrazione vengono commissionati e realizzati: sono infatti in genere applicazioni monolitiche, difficilmente scomponibili in componenti funzionali singolarmente riusabili e non posseggono (non sono stati progettati a tal fine) quelle caratteristiche tecniche che ne facilitano il riuso.
Per aumentare l'efficienza del riuso occorre quindi ricorrere a quel concetto "evoluto" di riuso che abbiamo prima introdotto e che prevede la possibilità di costruire nuovi sistemi software assemblando singoli componenti, appositamente costruiti per essere assemblati in architetture logico-funzionali e tecniche anche – per quanto possibile – non note a priori. Ovvero, riusano un software che può essere definito "multiuso", nel senso che svolge funzioni di carattere generale che possono essere utili in contesti diversi e architetture differenti. Si tratta, ad esempio, di funzioni di ricerca, trattamento immagini, gestione documentale, business intelligence, autenticazione etc.. Il CNIPA, nel Piano Triennale per l'ICT nella Pubblica Amministrazione 2009-11, ha definito una prima classificazione di questi componenti funzionali, per tipologia di funzioni che offrono.
Ma esistono componenti funzionali riusabili già disponibili? In effetti sono già disponibili molti componenti software "prefabbricati", sia *open source*, sia di proprietà privata (il cui uso viene concesso pagando un determinato prezzo), sia di proprietà pubblica (il cui riuso è gratuito per le pubbliche amministrazioni). Riusarli nei progetti di sviluppo di nuovo software non è però ancora semplice: mancano cataloghi pubblici dove cercare questi componenti, manca uno standard condiviso per la descrizione di tali componenti – quali funzioni offrono, come interagiscono con altri componenti, quale formato dei messaggi utilizzano, in quali ambienti tecnologici funzionano etc – sebbene OMG abbia recentemente emesso uno standard per la descrizione degli "asset software riusabili" che può diventare il riferimento in questo settore, mancano pratiche standard di sviluppo software per assemblaggio di componenti.
In definitiva, ad ostacolare la diffusione dello sviluppo di software per assemblaggio di componenti funzionali multiuso vi sono problemi organizzativi (va definita la topologia dei cataloghi), semantici (vanno definiti i meta dati con cui descrivere i componenti), tecnici (vanno definite le caratteristiche tecniche che devono possedere i componenti per essere riusabili), procedurali (va definito come procedere alla selezione dei componenti riusabili e al loro adattamento nel nuovo progetto di sviluppo). Queste linee guida intendono fornire un contributo tecnico e metodologico alle pubbliche amministrazioni per superare le limitazioni su esposte.

5. La situazione attuale del riuso nella Pubblica Amministrazione

5.1 I DATI DI PARTENZA

Alcuni dati di particolare significatività dovrebbero indurre a considerare con grande attenzione le strategie di riuso del software applicativo nella Pubblica Amministrazione italiana: questa è infatti proprietaria di oltre 12 milioni di punti funzione[8] di software sviluppato *ad hoc* (detto anche *software custom*) per soddisfare proprie specifiche esigenze – probabilmente uno dei patrimoni software più grandi al mondo. La spesa che la Pubblica Amministrazione stima di dover sostenere per l'acquisto di nuovo software applicativo nel 2008 è di oltre 560 milioni di euro, quella per la gestione e manutenzione di software già esistente è di circa 190 milioni di euro.[9]

In base a questi dati, si deve convenire che il software applicativo nella Pubblica Amministrazione italiana costituisce un asset significativo, che cresce continuamente di dimensioni e valore, nonché di costi di gestione. Inoltre, grazie a questi numeri, la Pubblica Amministrazione italiana si pone come uno dei principali committenti di software oggi presenti sul mercato, nonché luogo di esercizio e sperimentazione di una grande varietà di tecnologie e soluzioni informatiche e può perciò giocare un ruolo importante come fattore di crescita economica e di innovazione nel Paese e nel settore ICT in particolare.

Con tali premesse, è indispensabile che la Pubblica Amministrazione avvii iniziative tese a razionalizzare la spesa di sviluppo e gestione del suo patrimonio software. Iniziative, d'altra parte, che sono chieste in varie disposizioni normative succedutesi nel tempo, tra le quali il Codice dell'Amministrazione Digitale, che dedica al riuso ampio spazio, e dal dPCM 31 maggio 2005 di attuazione del comma 192 della Legge finanziaria del 2005 (L. 311 del 30 dicembre 2004).

Il citato dPCM, in sintesi, impone esplicitamente (art. 2 comma 1) alla P.A. di definire progetti di riuso del software applicativo, considerato come uno dei possibili fattori di contenimento dei costi di sviluppo e gestione del software e di miglioramento della sua qualità. Il medesimo dPCM affida al CNIPA il compito di supportare questi progetti. Il CNIPA ha crea-

[8] I punti funzione sono intesi secondo le definizioni della ISO (v. ISO/IEC 14143) e di IFPUG (www.ifpug.org), e sono in parte calcolati come punti funzione "equivalenti".

[9] Fonti: consuntivo annuale CNIPA per l'anno 2006 (I function point sono in parte calcolati come FP "equivalenti") e Piano Triennale CNIPA per il 2008-2010.

to, a tal fine, un Centro di competenza specifico, che offre supporto metodologico e tecnico alle amministrazioni che intendono avviare un progetto di riuso.
Il Centro ha già pubblicato, tra l'altro, le linee guida al riuso di software già esistente e di proprietà della Pubblica Amministrazione, in cui sono state definite le modalità di gestione dei progetti di riuso di software già esistente basate su un ciclo progettuale che prevede che le amministrazioni, con l'assistenza del CNIPA, inseriscano in un catalogo, gestito presso il CNIPA stesso, le informazioni di base relative alle applicazioni software di loro proprietà che ritengono possano essere di interesse anche per altre pubbliche amministrazioni. Successivamente, le altre amministrazioni interessate ad una di quelle applicazioni si accordano con quella proprietaria, sempre con il supporto del CNIPA, per ottenere in riuso il prodotto e quindi lo adattano, a proprio carico, al nuovo contesto di destinazione, sia per gli aspetti tecnici che funzionali.
Questo modo di gestire il riuso di applicazioni software esistenti ha finora consentito dei vantaggi economici per la Pubblica Amministrazione, pur se utilizzato in un numero ancora relativamente limitato di occasioni.
In sostanza, nonostante gli sforzi – anche normativi – finora fatti, il riuso come fattore di razionalizzazione dell'efficienza operativa e di contenimento della spesa di acquisto e gestione del software nella Pubblica Amministrazione sembra ancora essere ai primi passi. Nei progetti di sviluppo di nuovo software della P.A. si riusa poco software già esistente e, quando si riusa, i costi del riuso sono ancora alti.

5.2 LIMITAZIONI ATTUALI ALL'EFFICIENZA DEL RIUSO NELLA P.A.

La parziale inefficienza del riuso nella P.A. ha diverse motivazioni, di ordine organizzativo, culturale, tecnico, giuridico.

a) Limitazioni legate alla specificità dei domini funzionali nella P.A
   La forma di riuso attualmente perseguita nella Pubblica Amministrazione è quella che prevede il recupero e riadattamento di intere applicazioni già esistenti, sviluppate originariamente per altre amministrazioni ma che offrono funzioni simili a quelle necessarie all'Amministrazione che ne richiede il riuso; il riuso di una porzione cospicua di applicazione è possibile però solo in un numero limitato di casi, in cui combaciano gran parte delle macro-esigenze delle amministrazioni cedenti e riusanti; riadattare (personalizzare) le specifiche funzionali da un contesto ad un altro ha dei costi alti;
b) Vincoli tecnologici insiti nel software da riusare
   Le applicazioni software sono spesso realizzate a partire da pacchetti applicativi proprietari, dai quali non sono sempre agevolmente separabili: per riusarle, occorre acquistare le licenze d'uso di tali pacchetti.
   Molte applicazioni sono vincolate a usare solo determinati middleware, sistemi operativi, RDBMS, il cui uso (se soggetto a licenza o acquisto) impone al soggetto riusante altri costi.

Il software open source è ancora raro nella P.A.

Il software open source utilizzato dalle P.A. è ancora poco: i pacchetti proprietari possono essere più convenienti e affidabili in molti casi, ma, ad ogni nuovo progetto, andrebbe eseguita una valutazione preliminare sul tipo di software da realizzare o acquistare, comparando costi e benefici delle possibili alternative, anche considerando quelli legati ad un eventuale futuro riuso. Riusare software open source vuol dire, infatti, evitare di dover sostenere costi di acquisto licenze per le funzioni offerte dai pacchetti proprietari.

Molti software custom candidati al riuso sono vecchi di anni e quindi parzialmente inefficienti rispetto allo "stato dell'arte" della tecnologia.

c) Scarsa circolazione delle informazioni sul software riusabile

Quasi nessuna P.A. ha un catalogo del proprio software applicativo (tanto meno di quello riusabile) – con alcune importanti eccezioni, come il comparto Tesoro e Finanze e alcuni Enti pubblici non economici, come INPS e INAIL).

I cataloghi dovrebbero descrivere le caratteristiche funzionali e quelle tecniche del software, con particolare riguardo alla riusabilità (permettendo di stimare i costi del riuso, inclusi quelli dell'acquisto delle licenze di eventuali prodotti proprietari indispensabili per il funzionamento del software riusato).

I pochi cataloghi esistenti di software della P.A. non sono tra loro collegati, né sono indicizzati in repertori, né sono interrogabili in maniera applicativa da soggetti esterni.[10]

Le P.A. non considerano come opzione in un nuovo progetto di valutare anzitutto se sia già disponibile del software per realizzare parte di quello nuovo.

Né si pongono come esplicito obiettivo di popolare, con il prodotto dello sviluppo, i cataloghi di software riusabile (per agevolare i futuri sviluppi).

d) Il software disponibile per il riuso non è stato progettato e realizzato per essere riusato

Non è quasi mai stato progettato "a componenti" riusabili (indipendenti e coesi), nell'ottica che anche i singoli componenti possano essere riusati.

Se è difficile riusare intere applicazioni, può essere infatti più facile riusare singoli componenti che svolgono funzioni specifiche, estrapolabili dal contesto funzionale dell'applicazione e utili anche in contesti diversi. Aumentando, in fase di progettazione, la granularità e modularità dell'architettura funzionale del software da realizzare, si aumenta la probabilità che alcune sue componenti vengano riusate. In tal senso, i cataloghi di software riusabile della P.A. dovrebbero divenire cataloghi di componenti software riusabili.

Le applicazioni software oggi disponibili nella P.A. non sono viceversa quasi mai scomponibili in componenti singole tra loro "separabili" e singolarmente riusabili.

[10] La disponibilità di tale catalogo è elemento propedeutico imprescindibile alla valutazione della scelta del riuso come opzione per la realizzazione di nuovo software nella Pubblica Amministrazione, come richiesto nel D.Lgs 7 marzo 2005 n. 82 (Codice Amministrazione Digitale - Capo V, Sez. I, art. 68, comma 1).

Questa situazione, di fatto, rende improduttivi, almeno in parte, importanti investimenti pubblici, costringendo quasi tutti i nuovi progetti di sviluppo software della P.A. a "ripartire da zero".

La disponibilità di componenti riusabili è infatti la base della produzione industriale in tutti i settori dell'ingegneria. Solo disponendo di un numero adeguato di componenti riusabili, reperibili con facilità, descritti adeguatamente nei cataloghi delle P.A., adattabili e modificabili senza sforzo per rispondere a esigenze diverse, l'efficacia del riuso nella P.A. – e dello stesso processo produttivo – può aumentare. A partire da questi componenti, infatti, sarebbe possibile progettare e realizzare molto nuovo software con facilità, per assemblaggio di componenti già esistenti, innescando poi un circolo virtuoso del riuso che può abbattere i costi dello sviluppo del software e aumentare di molto la sua qualità. Infatti, il software riusato è intrinsecamente di maggiore qualità (è stato "verificato" più volte per vari contesti) ed è in grado di trasmettere questa sua qualità all'applicazione nelle quale viene inserito.

Manca ancora una comunità di sviluppatori trasversale alle P.A. che, sul modello del mondo open source, provveda a migliorare continuamente e far evolvere il software disponibile per il riuso.

Le caratteristiche tecniche del software applicativo esistente non sono quasi mai quelle "ottimali" per il riuso. Manca una definizione standard degli attributi che connotano un software "riusabile".

Conoscere questi attributi permette agli sviluppatori di progettare e realizzare un software che sia riusabile e permette di valutare la riusabilità di un software già esistente, decidendo se vale la pena di riusarlo.

e) La documentazione progettuale e di manutenzione associata al software da riusare è carente

Viene raramente adottato dai produttori un processo di produzione del software orientato a costruire software riusabile: questo processo realizza i documenti progettuali, i casi di test e ogni altra documentazione a corredo del software pensando a un suo possibile riuso, usando notazioni standard, curandone la completezza, assicurando la consistenza e coerenza dei documenti nonché il loro costante aggiornamento.

Spesso mancano o sono carenti le procedure di configurazione e installazione, le procedure per la parametrizzazione e la personalizzazione, i documenti progettuali quali la architettura logico funzionale e i diagrammi delle classi, i casi di test etc. La personalizzazione/modifica di tale documentazione per adattarla ad altri contesti è quindi molto difficile e onerosa per produttori diversi da quello originale.

Si riusano molto poco gli elementi "non software" di una applicazione: schemi progettuali, patterns, casi di test, schemi e strutture di dati e di documentazione, possono essere riusati a loro volta, abbattendo i costi del riuso.

Perciò, anche questi elementi dovrebbero essere prodotti dai tecnici pensando a un loro possibile riuso, anche solo parziale.

f) La normativa sul riuso è ancora ambigua o carente su molti aspetti, quali, ad esempio:
   - la proprietà del software (manca anche una definizione di opere derivate),
   - la limitazione di responsabilità nel riuso.

g) Manca una metrica che consenta di valutare in maniera oggettiva la "quantità" di riuso presente in un progetto di sviluppo software. Questa metrica permetterebbe di negoziare nei contratti per i nuovi sviluppi l'adozione della tecnica del riuso, valutandone gli effettivi benefici anche dal punto di vista meramente economico.

h) È mancato un coordinamento complessivo degli acquisti di software delle Pubbliche Amministrazioni, che hanno perciò proceduto agli acquisti "in ordine sparso", per soddisfare sempre specifiche esigenze, senza pensare a possibili ricadute trasversali delle proprie spese.
   Servirebbe inquadrare ogni nuovo progetto di sviluppo software nei suoi costi e benefici complessivi per l'intera P.A., laddove possibile.

In definitiva, è mancata una strategia di acquisto del software nella Pubblica Amministrazione, basata su leve tecniche e giuridico-amministrative, che favorisca il riuso di quanto acquistato, nell'ottica di una visione unitaria del patrimonio software pubblico.

6. Una strategia per aumentare l'efficienza del riuso del software nella Pubblica Amministrazione

6.1 Gli elementi base della strategia

Sulla base delle esperienze di progetti di riuso condotte nella Pubblica Amministrazione, sia centrale che locale, e delle considerazioni espresse nel precedente paragrafo, è possibile delineare i contenuti di un progetto "riuso" della Pubblica Amministrazione, che si ponga l'obiettivo del contenimento della spesa pubblica, della maggiore qualità delle applicazioni software e dei servizi che su di esse si poggiano, del miglioramento della efficienza operativa delle strutture pubbliche, sia per quanto riguarda il *front office* verso cittadini ed imprese che nel *back office* che supporta il funzionamento degli uffici pubblici.
La strategia deve affrontare le cause dei problemi che limitano il riuso nella P.A., come sopra individuate. In sintesi, una tale strategia può essere riassunta nelle seguenti affermazioni.

1. Se la cooperazione tra amministrazioni ai fini del riuso non è ancora soddisfacente, occorre aumentare la circolazione di informazioni tra le amministrazioni.
   Uno dei maggiori problemi che ostacola il riuso dei software *custom* è la loro scarsa "notorietà": mentre i prodotti COTS sono pubblicizzati dai produttori al fine di trovare loro degli acquirenti, i proprietari dei software *custom*, e ciò vale anche nel caso della P.A., non pubblicano cataloghi del software di cui dispongono. Perciò il riuso di questi software è spesso attivato da una sorta di "passaparola" o, talvolta, da eventi esterni, come i bandi pubblici di e-government che finanziano progetti di riuso di software *custom* già esistenti, che agevolano la circolazione di informazioni, anche se spesso per un tempo limitato a quello della durata del bando.
   Ogni Amministrazione dovrebbe quindi definire un catalogo del software di sua proprietà, descritto in modalità standard (ad esempio utilizzando specifiche OMG definite per la descrizione degli asset riusabili, basate su XML), dovrebbe pubblicarlo sul SPC, rendendolo interrogabile attraverso la propria porta di dominio. I cataloghi dovrebbero essere indicizzati a cura di un soggetto terzo (ad esempio il CNIPA), per facilitare la ricerca dei contenuti. I cataloghi dovrebbero contenere sia intere applicazioni che singoli componenti che svolgono funzioni specifiche.
   Secondo lo SWEBOK (*Software Engineering Book of Knowledge*) pubblicato dallo IEEE e lo standard IEEE 1517-99, riusare vuol dire primariamente creare biblioteche dei beni riusabili.

I cataloghi del software riusabile non devono contenere solo le informazioni tecniche necessarie per decidere del riuso degli oggetti che referenziano, ma anche quelle economiche che evidenziano i possibili costi associati al riuso (ad es. licenze da acquistare), quelle necessarie a riusare effettivamente il software (ad es. personalizzazioni, parametrizzazioni da prevedere), i vincoli tecnici e di portabilità;[11]

2. Se il riuso di intere applicazioni è difficile, in quanto raramente i domini funzionali di due amministrazioni si sovrappongono, occorre aumentare il livello di granularità del riuso, prevedendo il riuso a livello anche di singoli componenti, anziché di intere applicazioni.
   Per facilitare il riuso di componenti occorre che il software di nuovo sviluppo sia progettato e realizzato, per quanto possibile, a componenti, in possesso di quelle caratteristiche (attributi) di riusabilità (e più in generale di qualità) che ne facilitano il successivo riuso. Questi componenti "riusabili", via via che vengono realizzati dalle P.A., andranno ad alimentare i cataloghi del software riusabile aumentando le possibilità di riuso.
   Nella progettazione va introdotto il concetto di "architetture" di servizi applicativi, secondo il paradigma della SOA e della cooperazione applicativa basata su web services definita in SPC.
3. Se la facilità del riuso è oggi bassa e i costi di riuso sono ancora alti (costi di qualificazione, adattamento, personalizzazione etc..), a causa delle caratteristiche del software che viene riusato, occorre aumentare la qualità dei componenti software candidati al riuso, dotandoli poi di caratteristiche tecniche e architetturali tali da rendere più conveniente e agevole il riuso.
   Queste linee guida individuano 12 specifiche caratteristiche di "riusabilità" del software, e forniscono raccomandazioni per la valutazione sia della riusabilità del software, sia della "quantità" di riuso effettuato in un progetto di sviluppo software.
   Per facilitare il riuso va aumentata l'attenzione alle attività di testing durante lo sviluppo; poiché il software da produrre potrà essere riutilizzato anche in altri contesti, dovrà essere messa particolare cura nel progettare ed effettuare i test, non solo al fine di scoprire ed eliminare i difetti nel prodotto durante il ciclo di produzione, ma anche per fornire al soggetto riusante un framework di supporto operativo al testing di qualificazione e una documentazione di test esaustiva, il tutto a sua volta "riusabile"; in sostanza, anche i test, come il software, devono essere progettati e documentati per un loro futuro riuso.
4. Ogni Amministrazione dovrebbe adottare, per sviluppare nuovo software di tipo custom (e farlo adottare ai propri fornitori), un processo di produzione di tipo get-

[11] Gli attuali cataloghi del software riusabile contengono solo la descrizione di massima delle applicazioni disponibili, ma non forniscono le informazioni indispensabili per la loro estrazione, nella versione riusabile, dagli archivi informatici dell'amministrazione cedente, né le informazioni necessarie al riuso del software in altri contesti.

put, che preveda come prima azione la valutazione della disponibilità di software della P.A. già esistente e riusabile e la sua convenienza rispetto ad altre soluzioni (e.g. sviluppo ex novo o acquisto di pacchetti), e come azione finale la pubblicazione dei componenti riusabili sviluppati sul catalogo dell'Amministrazione committente.

Attraverso questo ciclo si può rapidamente aumentare l'efficienza del riuso (i componenti che vanno a popolare il catalogo sono in fondo i "mattoni" con i quali costruire parte di altre applicazioni, in misura progressivamente sempre maggiore, come avviene per i *web services* e le architetture SOA).

La documentazione progettuale e ogni altro artefatto corredato al software previsto nel processo di produzione devono essere realizzati in modo esaustivo e con notazioni standard, che ne facilitino la comprensione e l'utilizzo anche in altri contesti.

I casi di test dovrebbero essere progettati in modo da essere a loro volta riusabili.

Il software dovrebbe essere codificato in maniera comprensibile e standard, per quanto possibile.

Il software dovrebbe essere corredato da procedure di installazione "riusabili".

L'approccio al riuso sopra descritto è simile a quello delle comunità *open source* – ma anche a quello adottato dalle metodologie di sviluppo "agili", come ad esempio eXtreme Programming. Volàno di questo nuovo riuso può quindi essere una "comunità" di sviluppatori e utilizzatori del software applicativo nella Pubblica Amministrazione, che operi sul software applicativo di proprietà della P.A. per migliorarlo costantemente e renderlo sempre più "riusabile" e con caratteristiche di "qualità".

In queste linee guida vengono date indicazioni per attuare questa strategia.

Tuttavia, nessuna strategia si può poggiare solo su indicazioni metodologiche e tecniche. Un errore che si commette spesso quando si affronta il tema del riuso è quello di ritenere che la tecnologia, da sola, possa determinare il riuso di un software: ad esempio utilizzando un dato linguaggio di programmazione, una certa architettura tecnica e così via. L'efficacia del riuso, viceversa, necessita, accanto a un uso appropriato delle tecnologie, anche della definizione di un contesto organizzativo che supporti ed agevoli il riuso, della diffusione della cultura del riuso nei committenti, a livello manageriale e non solo tecnico, nonché della definizione di soluzioni giuridico-amministrative atte a premiare il riuso negli appalti pubblici. Solo chiudendo il cerchio con questi altri tasselli sarà possibile ricavare dal riuso l'efficacia attesa.

In particolare, occorre fornire indicazioni per regolare, nei documenti contrattuali, aspetti come:

- la proprietà del software – va chiarito cosa è compreso e cosa no nell'acquisizione di software *custom*, ad esempio definendo se fanno parte dell'acquisto anche le cosiddette "opere derivate", il *know how*, i tools utilizzati per lo sviluppo, schemi progettuali ripetibili come *patterns* etc; tali elementi, pur quando non sono oggetto di trasferimento di piena proprietà, devono comunque essere resi disponibili dal produttore all'acquirente, che ne potrà fare uso per proprie finalità;

- le limitazioni di responsabilità nel riuso – va chiarito se, nel caso di malfunzioni in un software riusato, la responsabilità va addebitata al soggetto che riusa o a quello che ha sviluppato originariamente il componente.[12]

6.2 GLI ULTERIORI ELEMENTI DELLA STRATEGIA

L'efficacia delle raccomandazioni di cui al paragrafo precedente potrà essere amplificata dalla adozione da parte delle pubbliche amministrazioni di ulteriori strumenti a supporto dello sviluppo per il riuso:

a) l'obbligo, per ogni software sviluppato, di essere accompagnato da un **libretto del riuso**, una sorta di manuale operativo per il riuso predisposto dallo sviluppatore al momento della realizzazione del software come ulteriore output dello sviluppo, e da aggiornare poi nel tempo (al variare della configurazione originale); nel libretto vanno riportate le informazioni necessarie sia a valutare la riusabilità del software nei vari contesti, sia a effettuare l'operazione di riuso (inclusa la installazione in nuovi ambienti) e di eventuale adattamento del software;

b) la creazione presso le amministrazioni di **centri di competenza sul riuso**, strutture specializzate che si occupino di supportare i progetti di sviluppo software nella valutazione dell'opzione del riuso, nella selezione dei componenti candidati al riuso, nella pubblicazione sul catalogo della amministrazione del software riusabile sviluppato. Il centro dovrà anche gestire il catalogo del software riusabile dell'Amministrazione e fornire indicazioni per aumentare costantemente nel tempo il livello di riusabilità degli oggetti presenti nei cataloghi del software riusabile, agendo sul codice sorgente e sulla documentazione associata, con un'attività di manutenzione e aggiornamento successiva al primo rilascio;

c) la definizione di una **licenza di libero utilizzo del software ai fini del riuso**, per semplificare gli adempimenti formali per la cessione di software tra amministrazioni; la licenza potrebbe essere rilasciata dall'Amministrazione cedente, nel momento in cui aggiunge al catalogo del software riusabile un nuovo componente software, in modo che chiunque lo voglia riutilizzare a favore della P.A. lo possa fare senza dover sottostare ad ulteriori adempimenti formali, ma nello stesso tempo senza lucrare sul componente riusato; ad esempio, un'Amministrazione potrebbe emettere un bando di gara in cui si invitano i partecipanti a riutilizzare software presente in un catalogo a cui i fornitori, una volta aggiudicatasi la gara, potrebbero accedere liberamente, per finalità di servizio verso l'Amministrazione stessa;

[12] Ad esempio, va definito, nel caso di riuso di componenti software prelevati da un catalogo per creare una nuova applicazione, come è delimitato l'ambito di responsabilità tra i vari costruttori che hanno contribuito allo sviluppo, considerando chi ha fornito i componenti, chi li ha assemblati, chi li ha testati, integrati, modificati.

d) la definizione di una **metrica** condivisa e oggettiva per **misurare la quantità di riuso** in un progetto di sviluppo software, che possa essere anche un elemento contrattuale per valutare le implicazioni economiche del riuso in un progetto.

L'insieme di questi elementi, e degli ulteriori contenuti in queste linee guida, costituisce un f*ramework* di strumenti tecnici, metodologici e giuridico-amministrativi che possono, nel loro insieme, supportare e sostenere l'avvio di un circuito virtuoso del riuso nella Pubblica Amministrazione, in grado di attivare la produzione di nuovo software per il riuso, popolare cataloghi di software riusabile e produrre in futuro sempre più nuovo software ottenuto assemblando e riusando i componenti software provenienti da questi cataloghi.
Le raccomandazioni fornite in questo documento tengono conto dello stato dell'arte attuale della tecnologia e non sarebbero state adottabili fino a qualche tempo fa: l'attuale sviluppo dei linguaggi di programmazione, delle librerie di componenti, dei meccanismi di integrazione tra i moduli, la disponibilità di patterns, frameworks e toolkits per progettare e/o realizzare software, la maturità raggiunta da paradigmi architetturali come la cooperazione applicativa (definita nell'ambito SPC dal CNIPA) e la SOA sono i presupposti indispensabili per pensare ad un nuovo modo di fare riuso.

6.3 GLI IMPATTI DELLA STRATEGIA SUI COSTI DELLO SVILUPPO SOFTWARE

Lo sviluppo di software riusabile può avere dei costi aggiuntivi rispetto a uno sviluppo tradizionale, costi legati sia alla necessità di dare al software caratteristiche tali da renderlo riusabile anche in contesti diversi da quello del primo cliente (contesti eventualmente noti solo in parte a priori), sia alla gestione del processo produttivo get-put (gestione dei cataloghi. ricerca nei cataloghi, loro popolamento con software riusabile, qualificazione dei componenti da riusare etc..).
Questi costi sono tuttavia sicuramente compensati, a regime, dai risparmi ottenibili sviluppando nuovo software con componenti prelevati dai cataloghi, con bassi costi di adattamento e integrazione. Inoltre, via via che il catalogo viene popolato da componenti riusabili, aumenta l'incidenza del riuso nei nuovi progetti e potenzialmente diminuiscono i costi di sviluppo di nuovo software.
Inoltre, per quanto riguarda molte delle raccomandazioni contenute in queste linee guida (accuratezza della documentazione progettuale, dei test, attenzione alla qualità dei componenti etc..), si tratta in realtà solo di una sistematizzazione delle "normali" modalità di produrre software di "qualità", a regola d'arte, come dovrebbe avvenire in qualunque progetto di sviluppo software.
Per queste considerazioni, si ritiene comunque la modalità d sviluppo software definita in questo documento conveniente per la Pubblica Amministrazione che, tra l'altro, dispone di una massa critica di progetto d sviluppo software che, se organizzati e resi tra loro coerenti, potrebbero portare in breve tempo a grandi benefici economici e qualitativi.

7. I requisiti del software riusabile

7.1 I MODELLI DI RIFERIMENTO

La definizione delle caratteristiche del software riusabile si basa, oltre che sul materiale prodotto dal gruppo di lavoro, sui modelli di riferimento contenuti negli standard ISO/IEC 9126 e ISO/IEC 12119, di cui viene data qui di seguito una sintetica presentazione, cui seguirà la individuazione delle caratteristiche che connotano un software riusabile.

7.1.1 IL MODELLO ISO/IEC 9126

Lo standard ISO/IEC 9126 *Software engineering – Product quality*, contiene un insieme articolato di requisiti (un "modello") per definire e valutare la qualità del software, applicabile sia al software sviluppati *ad hoc* sia ai prodotti COTS. Visto lo scopo che si pone, lo standard si concentra esclusivamente sulla definizione di questo modello, senza fornire indicazioni sui modelli e le best practices di produzione del software, o sulle modalità d valutazione della qualità del software, argomenti che rimanda ad altri standard ISO.
È stato pubblicato la prima volta nel 1991, una seconda tra il 2001 ed il 2004. Lo standard è in corso di ulteriore revisione da parte di ISO, nell'ambito del progetto SQuaRE (*Software Quality and Requirements Engineering*).
Non sono previste certificazioni di qualità del prodotto software rispetto a questo standard.
Il modello di qualità del software definito da ISO/IEC 9126 è gerarchico a 4 livelli:

1. tre "punti di vista" sulla qualità del software (interno-codice sorgente, esterno-prestazioni tecniche, utilizzo da parte degli utenti), che un progetto di sviluppo deve prendere in considerazione e soddisfare,[13]
2. gli attributi (o caratteristiche, definite come requisiti qualitativi, non direttamente misurabili) che qualificano un software, secondo i tre punti di vista,
3. per ogni attributo, le sottocaratteristiche misurabili (requisiti quantitativi, verificabili) che lo specificano e che dovranno essere misurate per valutare il livello di qualità effettivamente raggiunto nel software,
4. le metriche per effettuare le misure.

[13] Si noti che lo standard ISO/IEC 9126 non considera il punto di vista manageriale, interessato a costi e tempi di sviluppo. Questo punto di vista è ripreso, almeno in parte, nello standard UNI ISO 10006, *Linee guida per la qualità nella gestione dei progetti*.

Le caratteristiche qualitative che associa ai tre punti di vista sulla qualità sono in totale 16 (6 per la qualità interna, 6 per la qualità esterna e 4 per la qualità in uso), mentre le sottocaratteristiche sono complessivamente 27. Le metriche sono fornite in 3 Technical Reports (ISO/IEC 9126-2, 3 e 4) in relazione ai 3 punti di vista ed alle sottocaratteristiche che misurano.
Il riuso non è esplicitamente considerato nello standard ISO/IEC 9126 tra le caratteristiche e sottocaratteristiche di qualità del software, ma tra queste sono comprese diverse proprietà che un software dovrebbe possedere per essere riusabile.
Verranno quindi riprese in questo documento sia le sotto-caratteristiche di qualità che si ritiene debbano essere previste in un software riusabile, sia le metriche che lo standard ISO/IEC 9126 utilizza per la loro misura e valutazione. La scelta di individuare i requisiti di riusabilità direttamente al livello di sotto-caratteristiche deriva dalla esigenza di fornire ai Capitolati elementi immediatamente misurabili e valutabili, cui sia possibile associare delle metriche, come avviene ad esempio nel modello ISO/IEC 9126.
Le sottocaratteristiche riprese dallo standard sono: interoperabilità, modificabilità, analizzabilità, adattabilità, installabilità, coesistenza, testabilità, apprendibilità.
Il processo di valutazione della qualità del software (applicabile sia a prodotti *custom* che COTS) è definito nello standard ISO/IEC 14598, *Information Technology, Software Product Evaluation* Questo standard è stato in parte recepito anche in Italia da UNI ed è in corso di revisione (sarà ripubblicato nel 2007 come ISO/IEC 25040).

7.1.2 IL MODELLO ISO/IEC 12119

Solitamente, i prodotti COTS sono forniti inseriti in una confezione (*package*) che illustra all'acquirente le caratteristiche del prodotto. Spesso, questa è la principale fonte con la quale il produttore (o chi commercializza il prodotto) comunica con l'acquirente/utente per informarlo circa le caratteristiche di ciò che sta per acquisire e utilizzare. È perciò importante che queste informazioni mettano in grado chi le legge di capire se il prodotto soddisfa le proprie esigenze.
Lo standard ISO/IEC 12119 *Software product evaluation, requirements for COTS products and instruction for testing,* definisce:

1. i requisiti di qualità per i prodotti COTS (le caratteristiche di qualità che devono possedere),
2. i requisiti per la documentazione dei test effettuati sui prodotti COTS prima del rilascio sul mercato, inclusi i requisiti per effettuare i test, per la definizione dei casi di test e per la documentazione dei risultati dei test,
3. indicazioni su come valutare la conformità dei prodotti COTS rispetto a dei requisiti di qualità.

Lo standard riguarda quindi solo la valutazione che un acquirente può dare di un prodotto COTS esaminandolo dopo la sua immissione sul mercato. Non riguarda direttamente aspetti del processo produttivo, cui il cliente non ha diretto accesso.

Un prodotto COTS è conforme allo standard ISO/IEC 12119 se:

- rispetta i requisiti di qualità definiti nello standard,
- la descrizione sul *package* è conforme a quanto richiesto dallo standard,
- la documentazione allegata (di installazione, uso, manutenzione etc..) è conforme allo standard,
- prima del rilascio sul mercato è stato "testato" secondo quanto definito nello standard, producendo e rendendo disponibile ai clienti la documentazione di test definita nello standard,
- le anomalie rilevate durante i test sono state risolte prima del rilascio sul mercato del prodotto.

Lo standard ISO/IEC 12119 definisce questi elementi come rilevanti per la qualità dei prodotti COTS:
1) requisiti della descrizione del prodotto sul package:
tra questi requisiti include l'identificazione univoca del prodotto, del fornitore, della versione (o della data di rilascio), la descrizione del possibile utilizzo del prodotto, le caratteristiche del sistema *hardware* e *software* richiesto per l'uso, indicazioni sulle modalità di ottenere supporto e manutenzione sul prodotto.
Il package deve contenere affermazioni sulle caratteristiche del prodotto, nello schema 9126:

- *Funzionalità* offerte dal prodotto, sicurezza ed interoperabilità; se vi sono solo alcuni valori di input ammissibili per l'utente, devono essere chiaramente identificati; vanno descritte le condizioni che limitano le funzionalità e il modo di limitare utilizzi indesiderati;
- *Affidabilità* del prodotto, in particolare devono essere presenti e descritte le procedure per il recupero dei dati dopo malfunzionamenti e deve essere definita la capacità di gestire i malfunzionamenti (*fault tolerance*); vanno descritti anche gli accorgimenti che si possono mettere in atto per limitare i problemi;
- *Usabilità*: i tipi di interfacce offerte all'utente (ad es. menu, *web browser*, funzioni di *help*); le conoscenze base richieste all'utente per l'uso del prodotto; se effettuabili, devono essere descritte le modalità di personalizzazione e parametrizzazione del prodotto e allegati gli strumenti necessari a farle;
- *Efficienza*: configurazione necessaria per l'ambiente operativo di utilizzo; consumo di risorse;
- *Manutenibilità*: modalità per modificare il software, a fronte di problemi o nuove esigenze;
- *Portabilità*; il sistema *hardware/software* necessario e sufficiente per farlo funzionare.

2) requisiti per la documentazione allegata al prodotto:

- *Completezza*; deve includere la descrizione della procedura di installazione, se è previsto che l'utente possa effettuarla; se la documentazione è in più documenti, va fornita una guida ed un indice;

- *Correttezza*: non devono esserci ambiguità od errori;
- *Consistenza*: non vi devono essere contraddizioni nella documentazione; ogni termine usato deve avere lo stesso significato in ogni documento allegato;
- *Comprensibilità*: deve essere facilmente comprensibile;
- *Apprendibilità*: deve facilitare l'apprendimento dell'uso del software.

3) requisiti di qualità del prodotto:

- *Funzionalità*: tutte le funzionalità descritte nel *package* devono essere richiamabili; il software deve funzionare come descritto nella documentazione allegata e le condizioni di attivazione delle funzioni devono corrispondere a quanto descritto nel manuale d'uso;
- *Affidabilità*: il prodotto non deve consentire l'immissione di dati in ingresso dannosi o incorretti; non deve danneggiare o perdere dati;
- *Efficienza:* l'utente deve essere avvisato di possibili tempi di risposta lunghi per operazioni che esegue il prodotto a fronte di sue richieste;
- *Usabilità*; domande, messaggi del prodotto devono essere facilmente comprensibili dall'utente; messaggi di errore devono aiutare a correggere il problema, con l'ausilio della opportuna documentazione; la tipologia dei messaggi (errori, *warnings*, domande, risposte etc..) deve essere chiaramente individuabile; i formati dei campi di input e dei reports devono essere facilmente comprensibili dall'utente; l'esecuzione di una operazione deve essere annullabile (deve essere reversibile la modifica di dati); un utente deve poter apprendere come chiedere servizi al prodotto leggendo la documentazione e le funzioni di *help*;
- *Manutenibilità*; ciò che deve essere fatto per manutenere il prodotto deve essere conforme alle descrizioni sul package e nella documentazione allegata;
- *Portabilità*; l'utente deve poter effettuare l'installazione seguendo le istruzioni allegate; gli ambienti hardware e software descritti nella documentazione devono essere sufficienti a completare l'installazione; il prodotto deve poter essere rimovibile dal computer.

4) requisiti per la documentazione dei test pre-rilascio:

- *Scopo del test*: deve essere descritta la finalità dei vari test svolti;
- *Consistenza*: la documentazione non deve contenere ambiguità o differenze semantiche nelle sue varie parti;
- *Completezza*: la documentazione deve contenere il piano dei test, le caratteristiche dell'ambiente di test, i casi di test usati, elementi per la tracciabilità dei test, i risultati dei test effettuati;
- *Identificabilità*: ogni documento usato nel test deve essere univocamente identificato.

5) requisiti per la completezza dei test pre-rilascio:

- Tutte le funzioni e le caratteristiche del prodotto (come descritte sul package e nei manuali allegati) devono essere state testate, incluse le procedure di installazione.

7.2 LE CARATTERISTICHE BASE DEL SOFTWARE RIUSABILE

Sulla base di un'analisi dei modelli presentati nei precedenti paragrafi (gli standard ISO/IEC 9126 e ISO/IEC 12119) è possibile individuare un insieme minimo di caratteristiche di qualità applicabili al software di tipo *custom*, che si ritengono possano agevolarne la riusabilità. Questo insieme di requisiti non è esaustivo della qualità di un software né intende sostituirsi ai modelli standard di qualità del software: per ogni software rimane invariata, in tal senso, l'esigenza di soddisfare i requisiti ISO/IEC 9126 o ISO/IEC 12119, o gli altri che derivano da ulteriori modelli indicati dal cliente, eventualmente stabiliti espressamente in un Capitolato tecnico o che sono impliciti nella *regola d'arte* che sempre sovrintende lo sviluppo del software.[14]
Perciò qui di seguito verranno individuate, tra le varie caratteristiche che un prodotto software dovrebbe possedere in generale, solo quelle che più specificatamente agevolano il riuso.
Con tali premesse, le caratteristiche per il riuso, identificate in queste linee guida, sono:

1. modularità
2. interoperabilità
3. modificabilità
4. conformità a standard di codifica (comprensibilità e leggibilità)
5. conformità a standard di progettazione
6. adattabilità (a diversi contesti tecnologici e di utilizzo)
7. analizzabilità
8. configurabilità
9. installabilità
10. coesistenza
11. testabilità
12. apprendibilità.

Secondo l'approccio correntemente seguito in letteratura, che deriva dal citato standard ISO/IEC 9126, le caratteristiche (anche quelle non presenti nello standard) saranno riferite, laddove possibile, a tre viste qualitative:

1. la vista della qualità "interna", relativa alle caratteristiche che il codice software possiede in modo intrinseco, indipendentemente dall'ambiente di utilizzo e dall'utente. La qualità interna corrisponde alle modalità con le quali gli sviluppatori traducono i requisiti di qualità definiti dal committente in aspetti tecnici intrinseci al software e agli

[14] Si vuole dire che qualsiasi software, per essere riusato, dovrà essere anche affidabile ed efficiente, ma tali caratteristiche dovrebbero essere oggettivamente ed implicitamente proprie di qualsiasi oggetto software, anche se non progettato esplicitamente per un futuro riuso. A questi requisiti "standard" se ne aggiungono qui degli altri, specifici del riuso.

altri manufatti ad esso associati (documentazione di analisi, documentazione di progettazione, procedure di test, procedure di installazione, dati di esercizio);

2. la vista della qualità "esterna", relativa al software in esecuzione considerato come una scatola nera. La qualità esterna è tipicamente misurata e valutata impiegando metriche esterne durante la conduzione dei test in ambiente simulato, con dati simulati.
3. la vista operativa, "in uso", che corrisponde alla capacità del software di soddisfare le finalità per le quali è stato acquisito ed in particolare la sua capacità di supportare le esigenze dell'utente che ne fa uso (ad esempio aumentandone la efficienza lavorativa e la soddisfazione di lavorare con il software). La qualità "in uso" va verificata durante il periodo di esercizio, nel corso del quale l'oggetto software può essere anche qualificato per il riuso in altri contesti, eliminando difetti e completandolo secondo le indicazioni che emergono con l'utilizzo operativo in un ambiente reale.

Ogni caratteristica di qualità per il riuso sarà descritta fornendo le seguenti informazioni:

1. un identificativo in italiano e in inglese, laddove necessario per evitare ambiguità dovute alla traduzione dallo standard internazionale da cui è stato derivato;
2. la fonte di riferimento per la descrizione, se esistente;
3. una descrizione sintetica della caratteristica;
4. il punto di vista qualitativo che soddisfa secondo il modello ISO/IEC 9126; quando possibile sarà anche indicata l'operatività durante l'esercizio del software;
5. le metriche utilizzabili per la misura del livello con cui la caratteristica è posseduta da un oggetto software, misura da effettuarsi sia durante la fase di sviluppo che al momento del collaudo con tecniche metodi rigorosi;
6. le raccomandazioni progettuali e tecniche (sintetiche) su come dare al software un sufficiente livello di possesso della caratteristica.

Le metriche individuate in questo documento hanno la finalità di permettere il test del software per valutarne in modo oggettivo il livello di riusabilità. Sono rivolte quindi alle amministrazioni committenti, affinché le recepiscano nei propri piani metrici da allegare ai capitolati tecnici, e agli stessi sviluppatori, affinché ne tengano conto nella progettazione dei casi di test. I valori di soglia proposti nelle metriche hanno finalità puramente indicative, in quanto i valori effettivi devono essere scelti in funzione degli elementi di contesto del progetto (costo del progetto, livello di qualità richiesto, criticità del software nel suo dominio applicativo, tempi di sviluppo attesi etc..).
Le raccomandazioni progettuali hanno come scopo fornire alcune indicazioni che possono essere recepite nelle soluzioni architetturali nella progettazione del software.
Qui di seguito vengono trattate con maggior dettaglio le caratteristiche sopra individuate.

7.2.1 MODULARITÀ

Identificativo. Modularità.
Fonte di riferimento. Gruppo di lavoro.

Descrizione. La modularità non fa parte delle caratteristiche definite nello standard ISO/IEC 9126, ma è comunque un elemento importante su cui porre l'attenzione nel caso di sviluppo di sistemi software sia della categoria 2a che 2b definite in questo documento. Qui si intende la modularità di un software prevalentemente nella sua accezione funzionale.
Un software è modulare quando le funzioni che offre sono fornite da "componenti" singolarmente individuabili (e tra loro sufficientemente indipendenti) nella sua architettura logico funzionale. Ognuno di questi componenti (ad es. classi, metodi, oggetti, packages, routines, moduli etc..) può quindi essere realizzato, verificato e modificato in maniera indipendente dagli altri. Pertanto, le proprietà che connotano un software modulare sono le seguenti:

- i componenti in cui è scomposto il software devono rappresentare nella sua architettura logico funzionale soluzioni a problemi specifici, che operano indipendentemente da altri componenti (i componenti offrono "servizi" agli altri componenti, con un ruolo specifico e autoconsistente nell'architettura logico funzionale del sistema software);
- i componenti devono avere un basso livello di accoppiamento tra loro, nel senso che devono essere indipendenti e separabili in base a criteri di tipo funzionale, senza compromettere l'operatività del singolo componente che non deve dipendere per il suo funzionamento da un altro componente; se possibile non vi devono essere poi connessioni dirette tra componenti;[15]
- le comunicazioni all'interno del sistema software sono gestite solo tramite il passaggio di parametri tra funzioni e/o moduli;
- i componenti devono poter essere aggregati per fornire servizi più complessi;
- le malfunzioni (e i difetti) che dovessero affliggere un componente non devono propagarsi ad altri componenti, inficiandone il funzionamento;
- una modifica apportata a un componente non deve necessariamente comportare un eccessivo impegno per modificare di conseguenza altri componenti;
- un componente deve poter essere trasportato in un altro contesto, per fornire i suoi specifici servizi, senza dover necessariamente ricreare nel nuovo ambiente dei vincoli con gli altri componenti con cui collaborava nel sistema software originario.

La modularità diminuisce la complessità di un software e ne favorisce il successivo riuso (è più facile modificarlo per adattarlo, è più facile testarlo etc..). Inoltre, la modularità è, in un certo senso, il volano stesso della riusabilità del software: se le applicazioni vengono sviluppate per componenti (a moduli indipendenti, in grado di fornire specifici servizi) si possono comporre più facilmente nuove applicazioni assemblando moduli già esistenti.
Punto di vista qualitativo. ISO/IEC 9126: interno.
Metriche. Le metriche che possono essere utilizzate per valutare la modularità sono:

1. la proporzione AC/NC tra i componenti auto-consistenti (ovvero i moduli – chiamati in letteratura one in / one out – che non fanno riferimento a istruzioni e dati esterni)

[15] Si intende qui per connessione diretta la dipendenza del software da interfacce applicative messe a disposizione da un altro software e/o dai servizi offerti da un altro software e/o da specifici schemi da base dati.

AC ed il numero di componenti totali NC. I valori ammissibili variano tra 0 e 1. Un possibile ragionevole valore di soglia è pari a 0,8;

2. la proporzione VG/V tra numero di variabili globali VG ed il numero di variabili usate V (valore di soglia tipico in letteratura pari a 0.05);
3. la coesione delle classi, misurata attraverso il numero medio di sottografi non connessi (valore di soglia tipico in letteratura pari a 2) del grafo delle chiamate ed uso di variabili interne;
4. lo scarso accoppiamento tra le classi, misurata attraverso il numero medio di chiamate esterne fatte dalle classi (valore di soglia tipico in letteratura pari a 2).

Raccomandazioni progettuali. È auspicabile, in generale, che la soluzione sia progettata in modo tale da poterne estrapolare/sostituire/modificare anche solo un sottoinsieme funzionale, senza che questo ne comprometta l'utilizzabilità in altri contesti e altri sistemi. In genere, ciò si ottiene utilizzando nella fase progettuale un opportuno livello di astrazione per disegnare l'architettura del sistema software.

7.2.2 INTEROPERABILITÀ

Identificativo. Interoperabilità, *interoperability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della Funzionalità. L'interoperabilità è anche trattata da un'iniziativa del governo inglese, chiamata e-GIF (e-*Government Interoperability Framework*).
Descrizione. La capacità di un sistema software di interagire con uno o più sistemi specificati, scambiando dati mediante un determinato insieme di funzionalità. I dati scambiati sono definiti da un formato standard accettato dai sistemi che interagiscono tra loro e la comunicazione avviene tramite un protocollo concordato.
Punto di vista qualitativo. ISO/IEC 9126: interno, esterno.
Metriche. L'interoperabilità è misurabile considerando gli altri sistemi software noti con i quali il sistema in sviluppo deve poter dialogare, e i relativi formati di scambio dati e protocolli. Le principali metriche applicabili alla misura della interoperabilità sono le seguenti.

1. Sia B il numero dei formati dei dati dei sistemi software con i quali l'applicazione deve poter scambiare dati; sia A il numero dei formati dei dati correttamente implementati (ovvero che abbiano superato i relativi test) all'interno dell'applicazione. La proporzione A/B misura l'interoperabilità rispetto al formato dei dati. I valori ammissibili variano tra 0 ed 1, che rappresenta il valore desiderabile.
2. Sia B il numero dei protocolli di comunicazione dei sistemi software con i quali l'applicazione deve poter colloquiare; sia A il numero dei protocolli di comunicazione correttamente implementati (ovvero che abbiano superato i relativi test) all'interno dell'applicazione. La proporzione A/B misura l'interoperabilità rispetto ai protocolli di comunicazione. I valori ammissibili variano tra 0 ed 1 che rappresenta il valore desiderabile.

Raccomandazioni progettuali. Durante la progettazione dei sistemi *custom* deve essere posta molta attenzione nella definizione delle interfacce verso l'esterno, utilizzando formati basati su linguaggi di scambio (ad esempio XML) che favoriscano l'interazione con altre applicazioni. Nel caso di sistemi basati su servizi e porte applicative, l'interoperabilità è garantita dall'architettura stessa purché si basi su soluzioni standard e allo stato dell'arte, ad esempio mediante l'utilizzo di *web services*.
Nel caso di prodotti COTS, le caratteristiche di interoperabilità devono essere esplicitamente indicate, in termini di protocolli di comunicazione, standard di rappresentazione dei dati.

7.2.3 MODIFICABILITÀ

Identificativo. Modificabilità, *changeability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della Manutenibilità.
Descrizione. La capacità di un prodotto software di essere modificato con facilità (e.. per adattarlo a requisiti e contesti diversi da quelli originari).
Punto di vista qualitativo. ISO/IEC 9126: interno.
Metriche. Le principali metriche che possono essere utilizzate per valutare la modificabilità sono:

1. la densità dei commenti, come rapporto NCOMM/LOC tra il numero di commenti NCOMM e linee di codice LOC (valore di soglia tipico in letteratura pari a 20-30%);
2. la coesione delle classi, misurata attraverso il numero medio di sottografi non connessi (valore di soglia tipico in letteratura pari a 2) del grafo delle chiamate ed uso di variabili interne;
3. la semplicità delle classi, misurata con il numero medio dei metodi per classe e la lunghezza media dei cammini di ereditarietà (valori di soglia tipici in letteratura pari a 15-20 e 4-5);
4. lo scarso accoppiamento tra le classi, misurata attraverso il numero medio di chiamate esterne fatte dalle classi (valore di soglia tipico in letteratura pari a 2);
5. la standardizzazione della codifica, intesa come il rapporto tra numero di deviazioni dallo standard DS e le linee di codice DS/LOC (valore di soglia tipico in letteratura pari a 0,01).

Raccomandazioni progettuali.
Categoria 2a. Va posta attenzione, nella progettazione e realizzazione della soluzione, alla standardizzazione della progettazione e della codifica (ad esempio utilizzando design patterns e notazioni di uso comune per la rappresentazione del comportamento statico e dinamico del software), alla ricerca di una dimensione dei moduli non estesa (per poter intervenire facilmente solo sulle componenti da modificare, senza ripercussioni su porzioni più ampie del sistema), al basso livello di accoppiamento tra i moduli.
Categoria 2b. Tipicamente tale caratteristica viene ereditata dal prodotto COTS sottostante. Diventa quindi di fondamentale importanza che il prodotto COTS possegga le caratteristiche individuate per la seconda categoria.

7.2.4 CONFORMITÀ AGLI STANDARD DI CODIFICA (COMPRENSIBILITÀ)

Identificativo. Conformità agli standard di codifica.
Fonte di riferimento. Gruppo di lavoro.
Descrizione. Conformità a standard di codifica emessi da enti di standardizzazione, oppure *de facto*, industriali, imposti dal mercato. L'esigenza della conformità ad uno standard di codifica deriva, nel caso di sviluppo di software per il riuso, dalla diffusa comprensibilità e leggibilità che è necessaria in un codice software destinato al riuso, al fine di agevolare un suo possibile riadattamento e modifica per renderlo utilizzabile in contesti diversi da quello originario. Deve essere perciò semplice e rapido capire dove intervenire per apportare cambiamenti al codice, anche se non si fa parte del team di sviluppatori che hanno realizzato la prima versione di quel software. Per inciso si osserva che dovrebbero essere standard anche le modalità di realizzazione e descrizione dei casi di test associati al software.
Punto di vista qualitativo. ISO/IEC 9126: interno.
Metriche. Le metriche utilizzabili per misurare questa caratteristica sono misure di aderenza a standard e linee guida di codifica e sono rilevabili tramite ispezioni, ad esempio calcolando il rapporto tra numero di deviazioni dallo standard DS e le linee di codice esaminate DS/LOC. Si possono prevedere anche ispezioni finalizzate a valutare la leggibilità di un codice, valutando ad esempio il rapporto tra numero di linee di codice e commenti. Un valore soglia ragionevole è che meno del 1% del codice violi un qualche standard.
Raccomandazioni progettuali.
Categoria 2a. La comprensibilità può essere migliorata dalla presenza di commenti, associati ad ogni elemento significativo del codice (variabili, algoritmi, metodi, classi etc..), che ne chiarisca il ruolo nella struttura logica dell'applicazione e l'impatto che deriva da una sua modifica. La comprensibilità è migliorata anche dall'uso di una *naming convention* standard, con nomi auto-esplicativi dati a variabili, metodi, classi etc.., e da una semplicità di scrittura del codice che eviti cicli di decisione troppo annidati e che frazioni gli algoritmi in strutture logiche semplici e comprensibili con facilità. Alcuni metodi di sviluppo del software, come ad esempio *streme Programming*, impongono ai progettisti/programmatori di scomporre le strutture logiche dell'architettura funzionale del progetto in sottosistemi di piccole dimensioni, attraverso giochi di ruolo (Planning Game) e l'uso di carte CRC dove descrivere le funzioni che ogni "pezzo" di software realizza. Ciò rende i componenti software auto-esplicativi rispetto al ruolo che svolgono nell'architettura del sistema e consente di diminuire la quantità di documentazione progettuale e la quantità di commenti da inserire nel codice.
Categoria 2b. Lo standard è meno rilevante in quanto legato a requisiti di prodotto generalmente consistenti.

7.2.5 CONFORMITÀ A STANDARD DI PROGETTAZIONE

Identificativo. Conformità a standard di progettazione.
Fonte di riferimento. Gruppo di lavoro.

Descrizione. Conformità a standard di progettazione emessi da enti di standardizzazione, oppure *de facto*, industriali, imposti dal mercato. Gli standard emessi da enti di standardizzazione sono di regola da preferire a quelli *de facto*, in quanto mentre nel primo tipo di standard le modifiche vengono decise pubblicamente e sottoposte ad un processo pubblico di revisione, nel caso degli standard *de facto* la proprietà resta di un soggetto privato che può decidere di modificare lo standard in modo autonomo. Ciò crea una potenziale dipendenza da un soggetto privato che va valutata attentamente, caso per caso.
Dal punto di vista tecnico, la standardizzazione può riguardare diversi aspetti della progettazione:

- le forme di rappresentazione della progettazione (i modelli), eventualmente imponendo il ricorso a determinati schemi progettuali e modelli di larga diffusione se non standardizzati,
- la sintassi e semantica utilizzate nella progettazione, che devono essere omogenee e consistenti,
- le scelte architetturali, riferite agli elementi dello stack dell'architettura progettuale del sistema software.

In questo ultimo caso, di particolare rilevanza, ai fini della riusabilità, hanno le scelte relative a:

- piattaforme di sviluppo ed esecuzione del software (ad es. gli application server),
- meccanismi previsti per l'accesso ai dati,
- protocolli di comunicazione.

Ciascuna scelta mette a disposizione primitive diverse e molto spesso impone modelli architettonici diversi dalle altre. In alcuni casi, le scelte si riferiscono a degli standard (molto spesso solo *de facto*), come, ad esempio, l'application server Java 2 Enterprise Edition (J2EE) o meccanismi di accesso a basi di dati relazionali tramite API standardizzate (e.g., JDBC/ODBC). Anche in questi casi, tuttavia, è abbastanza frequente riscontrare situazioni in cui è necessario, o quantomeno utile, sfruttare specificità proprietarie del prodotto discostandosi dallo standard. È opportuno contenere queste deviazioni deliberate, evidenziandole e documentandole, al fine di rendere il prodotto più portabile.
In sintesi, occorre valutare il *trade-off* fra il perseguire una forte omogeneità tecnologica, che porta a vantaggi relativi alla riusabilità dei componenti nonché alla facilità di circolazione delle conoscenze e competenze, rispetto alla scelta di soluzioni tecnologiche eterogenee, che consente di selezionare la miglior tecnologia disponibile al momento per ciascun caso, rendendo però più problematico il riuso e implicando la necessità di gestire la coesistenza fra tecnologie diverse.
La omogeneità del lessico e della semantica utilizzati nei documenti progettuali, infine, sono anch'essi rilevanti, perché la mancanza di un vocabolario comune genera frequenti incomprensioni.
Punto di vista qualitativo. ISO/IEC 9126: interno.

Metriche. Le metriche utilizzabili per misurare questa caratteristica sono misure di aderenza a standard e linee guida di progettazione e sono rilevabili tramite ispezioni, ad esempio calcolando il rapporto tra numero di deviazioni dallo standard DS e diagrammi progettuali esaminati DS/Diag (un valore di soglia ragionevole è inferiore al 1%).
Raccomandazioni progettuali. Utilizzare Design Patterns – che sono dei veri e propri vocabolari concettuali – Frameworks e Toolkits. Rappresentare il comportamento statico e dinamico del sistema software utilizzando notazioni *standard de facto* o comunque di larga diffusione. Utilizzare tools per disegnare e gestire i diagrammi di progettazione. In particolare, come indicazione di massima, si raccomanda l'utilizzo di UML per la parte relativa alla modellazione dei requisiti e dei componenti del sistema ed il modello Entità-Relazioni per la fase di progettazione concettuale di una base di dati.
Per le scelte relative agli aspetti architettonici si raccomanda l'aderenza agli standard propri della Pubblica Amministrazione (e.g. in SPC) e agli standard ufficiali e *de facto*, motivando e documentando le eventuali deviazioni.

7.2.6 ADATTABILITÀ

Identificativo. Adattabilità, *adaptability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della *Portabilità*.
Descrizione. La capacità di un prodotto software di essere adattato ad ambienti differenti noti a priori, eventualmente referenziati in un capitolato tecnico, senza dover ricorrere ad azioni o mezzi diversi da quelli contemplati a questo scopo dal software stesso (funzioni di personalizzazione e configurazione in dotazione del software stesso).
Punto di vista qualitativo. ISO/IEC 9126: interno, esterno.
Metriche. Le metriche che possono essere utilizzate per valutare la adattabilità sono:

1. la proporzione INFUN/FUN tra il numero di funzionalità indipendenti dall'ambiente hardware e/o software di base INFUN ed il numero totale di funzionalità FUN. I valori ammissibili variano tra 0 ed 1, dove 1 rappresenta il valore ottimale (valori diversi possono essere fissati in funzione di specifici progetti e contesti);
2. la proporzione INFUN/FUN tra il numero di funzionalità indipendenti dalla organizzazione aziendale INFUN ed il numero totale di funzionalità FUN. I valori ammissibili variano tra 0 ed 1 dove 1 rappresenta il valore ottimale (valori diversi possono essere fissati in funzione di specifici progetti e contesti);
3. la proporzione INDAT/DAT tra il numero di strutture dati indipendenti dall'ambiente hardware e/o software di base INDAT ed il numero totale di strutture dati DAT. I valori ammissibili variano tra 0 ed 1 dove 1 rappresenta il valore ottimale (valori diversi possono essere fissati in funzione di specifici progetti e contesti);
4. la proporzione FDIFUN/DFUN tra il numero di funzionalità dipendenti dall'ambiente hardware e/o software di base che possono essere facilmente adattate ad altri ambienti FDIFUN ed il numero totale di funzionalità dipendenti da altri ambienti DFUN, I valori ammissibili variano tra 0 ed 1 dove 1 rappresenta il valore ottimale

(valori diversi possono essere fissati in funzione di specifici progetti e contesti, comunque senza scendere sotto 0,7).

Raccomandazioni progettuali.

Aspetti generali. In ottica di riuso, la previsione della necessità di dover in futuro adattare il software in via di sviluppo a nuove esigenze è praticamente inevitabile, a causa dei rapidi cambiamenti nella tecnologia, delle possibili differenze nei requisiti utente e della necessità di adeguamento a strutture hardware e software preesistenti nel contesto dove il software sarà riusato. Si consideri di dover adattare il software sia ad altri ambienti operativi (hardware e software di base) sia ad altri contesti organizzativi, sia ad altri insiemi di dati.

Le principali soluzioni progettuali da considerare sono di seguito riportate.

1. *Incapsulamento*: le funzioni richieste sono svolte all'interno di oggetti che espongono interfacce standardizzate. Non è, pertanto, necessariamente portabile il codice (sia sorgente che eseguibile), ma lo sono i risultati dell'elaborazione.
2. *Virtualizzazione*: le crescenti capacità di calcolo dei sistemi informatici hanno reso possibili vari meccanismi di esecuzione in più ambienti operativi (anche eterogenei) all'interno del medesimo hardware. È così possibile creare istanze multiple, conformi ai requisiti del sistema che si vuole eseguire.
3. *Astrazione*: sul mercato sono presenti librerie che forniscono il disaccoppiamento tra i vari livelli architettonici presentando interfacce quanto più possibile consistenti e traducendo tali interfacce nelle specifiche funzionalità offerte dai moduli sottostanti. Tale approccio può semplificare la scrittura del software, ma ne limita in varia misura sia le prestazioni che le possibilità di utilizzo dei moduli su cui si basa.

7.2.7 Analizzabilità

Identificativo. Analizzabilità, *analysability*.

Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della *Manutenibilità*.

Descrizione. Idoneità del prodotto software a essere esaminato per fini diagnostici diretti a individuare malfunzioni e difetti, o per individuare le parti da modificare.

Punto di vista qualitativo. ISO/IEC 9126: interno.

Metriche. Le metriche che possono essere utilizzate per valutare la analizzabilità sono:

1. la complessità ciclomatica (per misurare la complessità strutturale del grafo di controllo del software), da intendersi come valore medio dei valori misurati sui metodi (valore di soglia tipico in letteratura pari a 10);
2. la densità dei commenti, intesa come rapporto NCOMM/LOC tra il numero di commenti NCOMM e linee di codice LOC (valore di soglia tipico in letteratura pari al 20-30%);
3. la coesione delle classi, intesa come numero medio di sottografi non connessi del grafo delle chiamate (valore di soglia tipico in letteratura pari a 2);
4. la semplicità delle classi, misurata con il numero medio dei metodi per classe e la lunghezza media dei cammini di ereditarietà;

5. l'accoppiamento tra le classi, misurata come numero medio di chiamate esterne fatte dalle classi (valore di soglia tipico in letteratura pari a 2);
6. la standardizzazione della codifica, misurata come il rapporto DS/LOC tra numero di deviazioni dallo standard (DS) e le linee di codice (LOC) del software (da misurarsi con ispezioni) – (valore di soglia tipico in letteratura pari a 0.01).

Raccomandazioni progettuali.
In termini generali, per essere analizzabile, un sistema software non deve essere eccessivamente complesso, e le sue funzioni devono essere ben suddivise a livello architettonico.
Categoria 2a. L'analizzabilità di un sistema software risulta favorita dalla modularità del software, dalla disponibilità di documentazione progettuale che ne descriva l'architettura logico funzionale, dalla coesione dei moduli e dal loro scarso livello di accoppiamento, dall'utilizzo di standard di codifica da parte degli sviluppatori, dall'uso di commenti appropriati inseriti nel codice, dal basso livello di complessità strutturale del codice sorgente.
Categoria 2b. Tipicamente questa caratteristica è ereditata dal prodotto COTS sottostante. Diventa quindi di fondamentale importanza la verifica della caratteristica per il prodotto COTS più che per il componente stesso.

7.2.8 CONFIGURABILITÀ

Identificativo. Configurabilità.
Fonte di riferimento. Gruppo di lavoro.
Descrizione. La capacità di un prodotto software di essere configurato con facilità per rispondere a differenti esigenze e/o condizioni ambientali note a priori.
Punto di vista qualitativo. ISO/IEC 9126: interno, esterno.
Metriche. Il livello di configurabilità di un sistema software può essere valutato in base al numero medio di parametri di configurazione associati a ciascuna funzione principale svolta dal sistema. Il numero complessivo dei parametri determina lo spazio delle configurazioni che il sistema può occupare. Più è grande questo spazio, maggiori sono le caratteristiche di configurabilità. Per contro, va rilevato che un elevato numero di parametri può rendere il processo di configurazione troppo complesso e rischioso, così da diminuire gli eventuali benefici che derivano dalla configurabilità.
Raccomandazioni progettuali.
Aspetti generali. La configurabilità, sebbene non inserita tra le caratteristiche del modello ISO/IEC 9126, è una caratteristica molto importante ai fini della riusabilità del software. Infatti, non potendo al momento dello sviluppo prevedere tutti i possibili riusi del manufatto, è necessario dare al software il maggiore livello possibile di configurabilità, per soddisfare esigenze sia tecniche che funzionali diverse da quelle conosciute. Un componente software configurabile include quindi un insieme di strumenti o di funzioni dedicate alla configurabilità, che agevolano il lavoro di configurazione.
Categoria 2a. La caratteristica di configurabilità è critica per questa tipologia di soluzioni. Infatti tale caratteristica, dovendo necessariamente essere tenuta in considerazione fin dalle

prima fasi di progettazione di un componente software (e quindi influendo significativamente sui tempi e sui costi di realizzazione), rischia a volte di essere scarsamente presente in soluzioni prodotte con tempi o costi troppo contenuti.
Categoria 2b. Tipicamente tale caratteristica viene ereditata dal prodotto COTS sottostante. Diventa quindi di fondamentale importanza la verifica di tale caratteristica per il prodotto COTS più che per il componente stesso.

7.2.9 Installabilità

Identificativo. Installabilità, *installability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sotto caratteristiche della *Portabilità*.
Descrizione. La capacità di un prodotto software di essere installato con facilità in un insieme predefinito di ambienti operativi.
La installabilità del software dipende da quattro elementi fondamentali:

1. la disponibilità, a corredo, di strumenti operativi utili allo scopo (ad esempio dei wizard),
2. il grado di automazione del processo di installazione,
3. il numero limitato dei passi operativi necessari,
4. la disponibilità di un manuale di installazione completo e comprensibile.

Punto di vista qualitativo. ISO/IEC 9126: esterno e in uso.
Metriche. Le metriche che possono essere utilizzate per valutare la installabilità sono:

1. il numero di funzionalità offerte dal prodotto software a supporto della sua installazione (ad esempio wizard di installazione e di configurazione);
2. la proporzione MSTEP/STEP tra il numero di passi di installazione chiaramente ed esaustivamente spiegati in un manuale di installazione MSTEP, rispetto al numero STEP di passi di installazione da effettuare;
3. la proporzione ASTEP/STEP tra il numero di passi di installazione che possono essere automatizzati ASTEP ed il numero totale di passi STEP necessari per l'installazione;
4. la proporzione RSTEP/STEP tra il numero di passi di installazione che possono essere facilmente ripetuti RSTEP ed il numero totale STEP di passi necessari per l'installazione;
5. la proporzione AOP1/AOP2 tra il numero di ambienti operativi nel quale il software può essere installato per i quali il software dispone di funzioni di installazione (AOP1) ed il numero di ambienti operativi su cui può essere installato (AOP2).

I valori ammissibili per le misure rilevabili variano tra 0 ed 1. Il valore ottimale sarebbe 1, ma sono accettabili anche valori più bassi, in funzione dello specifico progetto. Per il punto 4), in particolare, il valore di soglia pari 0,7 può essere considerato ragionevole.
Raccomandazioni progettuali. Per quanto riguarda le procedure di installazione, vanno previsti opportuni programmi di attività che rilevano le caratteristiche dell'ambiente e prov-

vedono a modificarlo (creando opportune directory e file di configurazione) affinché il sistema possa essere correttamente installato.
Nel caso di sistemi *custom*, l'installabilità su più ambienti operativi si ottiene introducendo un livello architettonico nella progettazione che scherma il sistema software dall'ambiente operativo nel quale sarà installato. La dimensione di questo strato determina la complessità e i costi da sostenere per far sì che il sistema possa essere installato su più piattaforme (per ognuna di esse, in pratica, lo strato deve essere completamente riscritto).
Per i prodotti COTS questo requisito può essere verificato solo a posteriori, in quanto non è possibile intervenire successivamente a questo livello.

7.2.10 COESISTENZA

Identificativo. Coesistenza, *coexistence*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della *Portabilità*
Descrizione. La capacità di un prodotto software di coesistere con altri software indipendenti (noti a priori) in un ambiente comune, condividendo risorse comuni.
Punto di vista qualitativo. ISO/IEC 9126: interno, esterno.
Metriche. Le metriche che possono essere utilizzate per valutare la coesistenza sono:

1. la proporzione NEPROD/PROD tra il numero di prodotti software (noti a priori) con i quali l'applicazione coesiste senza errori NEPROD ed il numero totale PROD di prodotti software indicati nelle specifiche con i quali l'applicazione deve coesistere. I valori ammissibili variano tra 0 ed 1. Il valore ottimale sarebbe 1, ma sono accettabili anche valori più bassi, in funzione dello specifico progetto.
2. Numero di errori riscontrati sulle applicazioni coesistenti.

Raccomandazioni progettuali. Un software che possiede questa caratteristica di qualità ha la capacità di condividere risorse hardware e software con altre applicazioni nel medesimo ambiente operativo, senza che la presenza del prodotto sia bloccante per il funzionamento di altri prodotti previsti sul sistema. Il software dovrebbe permettere un'alta configurabilità delle risorse stabilmente occupate (ad esempio, porte per servizi) e un utilizzo delle risorse condivise non esclusivo.

7.2.11 TESTABILITÀ

Identificativo. Testabilità, *testability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della *Manutenibilità*.
Descrizione. La capacità di un prodotto software di essere sottoposto con facilità a verifiche che valutino sia il grado di rispetto dei requisiti (durante ed al termine del processo di sviluppo) sia la correttezza delle modifiche apportate al prodotto dopo la consegna e in fase di riuso (per correggere errori, per aggiungere funzioni, per migliorare le prestazioni e la qualità, per adeguarlo a variazioni dell'ambiente operativo, per adattarlo a nuovi contesti etc…). Per poter essere riusato in un nuovo ambiente, un software deve essere di norma sottoposto ad adattamenti e quindi a successive verifiche che accertino che abbia conserva-

to anche dopo il riuso le caratteristiche originarie richieste dal dominio ricevente e l'affidabilità complessiva.
La costruzione di casi e procedure di test è operazione complessa e costosa, ma è indubbio che quanto più un software acquisito per il riuso è facilmente testabile, tanto meno sarà costoso riusarlo. È opportuno, perciò, preferire soluzioni che riusano oggetti ben corredati di casi di test riusabili e adattabili a diversi contesti
Punto di vista qualitativo. ISO/IEC 9126: interno, esterno.
Metriche. Le misure del livello di testabilità di un sistema software rilevano la complessità del codice e del progetto architetturale (nella progettazione di dettaglio). Le metriche utilizzabili per rilevare queste proprietà sono quelle classiche utilizzate per verifiche di complessità strutturale e architetturale (si veda per queste quanto detto a proposito delle altre sotto caratteristiche della manutenibilità trattate in questo capitolo). Oltre a queste, dovrebbero essere considerate anche le misure che influenzano direttamente il processo di test, quali:

1. percentuale di funzioni elementari (o requisiti funzionali) definite nel documento di progettazione funzionale con almeno un caso di test associato nel Piano di test; percentuale di requisiti non funzionali con almeno un caso di test associato nel Piano di test. Un valore di soglia ragionevole per queste misure è pari a 0.99;
2. percentuale dei casi di test automatizzabili, espressa tramite la proporzione ACT/CT tra i casi di test che è possibile eseguire in modo automatico senza modifiche del modulo da testare ACT ed il numero totale di casi di test a cui il modulo è stato sottoposto CT. I valori ammissibili variano tra 0 ed 1, dove 1 rappresenta il valore ottimale. Un valore di soglia ragionevole è pari a 0.99.

Raccomandazioni progettuali.
Per i software *custom* orientati ad essere riusati, la testabilità è un requisito essenziale, poiché agevola la verifica nel tempo delle eventuali modifiche/adattamenti apportate al sistema per utilizzarlo in altri contesti. La testabilità è legata sia a scelte architettoniche e progettuali (modularità, bassa complessità architetturale, alto livello di coesione e basso livello di accoppiamento) sia alla definizione di un insieme adeguato di casi di test e di procedure di test, associati ai requisiti del prodotto.
È opportuno predisporre e associare al prodotto casi di test riusabili a loro volta, elaborati in modo da permettere di variare facilmente i dati inseriti per la prova (per renderli coerenti con le esigenze di contesto del dominio ricevente), senza la necessità di riprogettare ex-novo i test. Sono quindi utili strumenti di gestione e automazione di test che permettono di gestire la complessità del processo di testing, e di considerare adeguati livelli di copertura, tracciabilità, gestione del livello di rischio e le eventuali diverse versioni dei casi di test. In ogni caso, quanto prodotto per il test (casi di test, test automatici, eventualmente strumenti se utilizzati) è parte integrante della fornitura e va consegnato con essa al committente.
Nel caso di una cessione semplice, quando l'applicazione e la sua manutenzione viene presa completamente in carico dall'Amministrazione ricevente, la stessa dovrà provvedere a mantenere allineati i casi di test.

Per i prodotti COTS, la testabilità si misura verificando la dimensione e l'accuratezza dei casi di test allegati nella documentazione a corredo del prodotto.

7.2.12 APPRENDIBILITÀ

Identificativo. Apprendibilità, *learnability*.
Fonte di riferimento. ISO/IEC 9126. È una delle sottocaratteristiche della *Usabilità*.
Descrizione. La capacità di un prodotto software di essere appreso con facilità dagli utenti nelle sue modalità di utilizzo.
Punto di vista qualitativo. ISO/IEC 9126: esterno.
Metriche. Le metriche per valutare la apprendibilità prevedono test di usabilità che coinvolgono gli utenti finali. In particolare, le metriche che possono essere utilizzate per valutare l'apprendibilità di un software sono:

1. tempo medio degli utenti per l'apprendimento di una specifica funzione;
2. efficacia del manuale utente/documentazione *on-line*, misurata tramite la proporzione CT/T tra i task che sono stati eseguiti correttamente dall'utente consultando la documentazione CT ed i task oggetto di test T. I valori ammissibili variano tra 0 ed 1. Un valore di soglia ragionevole è pari a 0.95;
3. organizzazione del manuale utente/documentazione *on-line*, misurata tramite la proporzione CT/T tra i task per i quali la documentazione fornita comprende una descrizione completa delle modalità di utilizzo CT ed i task oggetto di test T. I valori ammissibili variano tra 0 ed 1. Un valore di soglia ragionevole è pari a 0.99.

Raccomandazioni progettuali.
Un software è "apprendibile" se è stato progettato secondo metodologie *user centered design*, ovvero coinvolgendo l'utente finale fin dalle prime fasi del progetto di sviluppo. Il ruolo dell'utente a supporto del team di sviluppo deve essere quello di:

- verificare che i requisiti da lui forniti siano stati ben compresi e descritti nel documento di specifica,
- definire i casi di test che dovranno essere utilizzati per la verifica sul software,
- valutare il manuale utente che viene prodotto a corredo del software,
- effettuare i test di usabilità che prevedono il coinvolgimento diretto dell'utente.

In letteratura, si associa comunemente una maggiore apprendibilità di un prodotto software all'adozione di un processo di produzione prototipale e incrementale, che permette di procedere per gradi insieme agli utenti e agli stakeholders in genere, per giungere progressivamente ad una visione comune delle funzionalità da realizzare e delle interfaccia da produrre. Per migliorare l'efficacia delle interfacce, in particolare, è utile organizzare l'interfaccia dell'applicazione seguendo standard noti all'utente, in modo da minimizzare il tempo di apprendimento dell'uso delle funzionalità del sistema.

8. Il processo di sviluppo del software riusabile

8.1 Aspetti generali del processo di sviluppo

La particolarità del processo di produzione del software raccomandato in queste linee guida consiste in questi due elementi:

a) la produzione avviene attraverso un processo (noto in letteratura come "Get-Put") finalizzato a costruire l'architettura del sistema software per assemblaggio di componenti funzionali riusabili pre-esistenti (come avviene in gran parte dei processi di produzione industriali);

b) il medesimo processo produttivo realizza – per completare l'architettura del sistema software – ulteriori componenti funzionali a loro volta riusabili, in quanto dotati di specifiche caratteristiche che ne facilitano il riuso in nuovi progetti.

Attraverso questo processo, a regime, la Pubblica Amministrazione dovrebbe disporre di cataloghi di componenti riusabili via via sempre più ricchi, dai quali prelevare i componenti funzionali necessari a costruire nuovo software *custom*, minimizzando così i costi di sviluppo e migliorando progressivamente la interoperabilità e la qualità dei software della Pubblica Amministrazione, grazie alle virtù intrinseche – di standardizzazione, di portabilità e di miglioramento qualitativo – indotte dal riuso.

Si possono realizzare prodotti riusabili sia utilizzando un processo di sviluppo di tipo iterativo e incrementale che a cascata, anche se è comunemente accettato in letteratura che l'utilizzo di un modello di produzione iterativo e incrementale diminuisce la densità di difetti residui nel prodotto rilasciato al cliente e massimizza la conformità del prodotto ai requisiti. Caratteristiche, queste, che sono senz'altro propedeutiche al successivo riuso del software in altri contesti.

Quanto alle metodologie di sviluppo, le principali organizzazioni che operano nel settore dispongono di adeguate e consolidate pratiche che, se correttamente messe in atto, permettono al software di possedere le caratteristiche di riusabilità fissate in questo documento nonché le ulteriori caratteristiche che deve comunque possedere un software fatto "a regola d'arte".

Riferimenti per definire un processo di produzione del software affidabile sono reperibili nel Capability Maturity Model Integrated (CMMI) e nello standard ISO/IEC 15504 *Information Technology - Software process assessment, oltre che, con minor dettaglio, negli standard ISO 90003 e ISO/IEC 12207.* Un riferimento più specifico per organizzare i processi di pro-

duzione del software orientati alla riusabilità è nel *Reuse Maturity Model* (RMM), di cui viene fornito un maggior dettaglio in Appendice 1.
Al d là di tali riferimenti, nell'ambito delle singole fasi che compongono tipicamente un processo di produzione del software, sono numerose le raccomandazioni che possono essere date per aumentare il livello di riusabilità dei prodotti in uscita dalla fase. Queste raccomandazioni vengono qui di seguito riportate, per fase cui si applicano. Preliminarmente, si ricorda l'importanza, ai fini del riuso, dell'utilizzo di modelli standard di documentazione progettuale e di documentazione degli ulteriori output del processo produttivo.

8.2 MODELLI STANDARD DI DOCUMENTAZIONE DEL SOFTWARE

La previsione della possibilità che la documentazione prodotta nel corso di un progetto di sviluppo di software possa essere a sua volta riusata in altri progetti suggerisce, per la sua stesura, l'adozione di un linguaggio standard (nella sintassi e semantica) al fine di favorirne la corretta interpretazione. Si consiglia l'adozione di un linguaggio standard di modellazione visuale, che sia in grado di rappresentare, ai vari livelli di astrazione, i requisiti di un'applicazione a partire dai macro processi di business fino alle scelte tecniche e architettoniche.
La modellazione grafica, all'interno di un processo di sviluppo, riveste infatti un ruolo fondamentale. Oltre a specificare, visualizzare e documentare tutto ciò che viene prodotto durante lo sviluppo, essa consente di:

- garantire l'uniformità di lettura della documentazione prodotta attraverso tutte le fasi che vanno dall'analisi dei requisiti all'installazione nell'ambiente di esercizio;
- fornire una chiave semantica che evidenzi le decisioni più importanti e strategiche prese sul software in sviluppo, che non si evincono direttamente dal codice sorgente;
- semplificare la lettura e comprensione delle componenti statiche e dinamiche dell'architettura del sistema software;
- agevolare la modifica della documentazione progettuale prodotta.

Da ciò consegue l'esigenza di adottare un linguaggio di modellazione che sia ampiamente supportato da strumenti automatici che, oltre a offrire la possibilità di modellazione grafica, consentano la possibilità di esportazione della documentazione, per favorire la portabilità degli elaborati prodotti. Le soluzioni scelte dovrebbero essere in grado di consentire la fruizione tramite web browser e in formato compatibile con i più diffusi editor di testo. È consigliabile, infine, preferire soluzioni che, integrando i vari strumenti grafici e di testo utilizzati, consentano la produzione semi-automatica della documentazione conclusiva in modo da favorirne la standardizzazione del contenuto.

8.3 Dettaglio sulle fasi del processo produttivo

In termini generali, le fasi di lavoro che vanno previste in un processo di sviluppo del software di tipo get-put finalizzato alla produzione di oggetti riusabili sono le seguenti.

1. raccolta, analisi e specifica dei requisiti,
2. analisi delle alternative *make or buy* o *riuso*,
3. progettazione tecnica,
4. realizzazione (codifica del software e/o integrazione di software riusato con software nuovo, realizzazione delle strutture dati, produzione della documentazione d'uso, manutenzione, gestione operativa),
5. collaudo,
6. installazione e rilascio in esercizio,
7. messa a disposizione nel catalogo degli oggetti software riusabili prodotti.

Il testing deve essere considerata un'attività presente in ognuna delle fasi sopra individuate, ovviamente con intensità, finalità e oggetto della verifica differenti da fase a fase.
Delle fasi sopra individuate, con esclusione del collaudo (regolato da altre considerazioni che esulano dagli scopi di questo documento) viene qui di seguito riportato un maggior dettaglio, con una focalizzazione sugli aspetti che possono influenzare il riuso. In conclusione del capitolo vengono fornite indicazioni relative alle attività di testing, da considerare come uno dei driver della riusabilità del software, in quanto ne migliora la qualità e riduce i costi di adattamento e verifica in caso di riuso (il processo di valutazione del software è poi richiamato con maggior dettaglio nell'Appendice 2), e alla attività di gestione della configurazione, altro fattore rilevante ai fini della riusabilità del software.

8.3.1 Raccolta, analisi e specifica dei requisiti

Un requisito è una dichiarazione documentata attestante una condizione o una capacità che un software deve possedere per soddisfare una richiesta di un utente, riguardante la risoluzione di un problema, il raggiungimento di un obiettivo, il rispetto di un contratto, una norma, o di altri documenti formalmente definiti. Un requisito è dunque una descrizione astratta di un servizio che il software deve offrire ovvero di un vincolo sulla realizzazione di tale servizio.
La "specifica" dei requisiti che il cliente pone a un software è la base per il contratto cliente-fornitore, ma anche l'input per le attività di disegno e codifica del software, che seguono temporalmente la specifica dei requisiti nell'iter del processo produttivo.
Nel caso di sviluppo di nuovo software con un processo get-put, una corretta specifica dei requisiti è fondamentale per diversi motivi:

a) al fine di documentare in modo completo ciò che il software fa, in modo da consentire, dopo il suo sviluppo, ai soggetti interessati al suo riuso, se quel software può essere loro utile;

b) durante lo sviluppo, per permettere ai progettisti di prendere una decisione in merito a quali componenti già esistenti riutilizzare nel processo produttivo, verificando quali funzioni devono essere realizzate;

c) sempre durante lo sviluppo, una volta selezionati i componenti da riusare nel processo produttivo, per poterli integrare correttamente nell'architettura logico funzionale e tecnica dell'applicazione.

I requisiti vanno raccolti presso gli utenti (intervistando se possibile più categorie di utenti, con metodi appropriati), e poi analizzati, per verificarne la fattibilità, la consistenza, la coerenza e congruenza (nel caso di requisiti forniti da più soggetti). Dopo che cliente e sviluppatore ne hanno negoziato la fattibilità e le eventuali priorità, i requisiti devono essere specificati in un documento, che li descrive in modo non ambiguo, chiaro, esaustivo, in maniera tale che il cliente possa validare tali requisiti in modo formale, permettendo di avviare le fasi successive del processo produttivo. Eventuali modifiche ai requisiti sono ovviamente sempre possibili in corso d'opera, ma vanno a loro volta negoziate e riportate nel documento di specifica, indicando le motivazioni della modifica e gli impatti sul progetto.

Come previsto dalla maggior parte delle attuali tecniche di analisi, i requisiti del software sono classificati in due macro categorie, i requisiti funzionali e quelli non funzionali.

I *requisiti funzionali* descrivono i servizi che il software deve erogare agli utenti, evidenziando le diverse modalità di utilizzo (interazioni) da parte dei possibili attori[16] e gli scenari di utilizzo.

I *requisiti non funzionali* comprendono un'ampia categoria di esigenze che possono essere espresse dagli utenti e dai committenti del software, tra le quali, ad esempio, le "prestazioni" e l'efficienza (vincoli sul tempo di risposta, sull'occupazione di memoria, ecc.), la sicurezza, la usabilità, l'affidabilità, la tecnologia da utilizzare (un dato middleware, un sistema operativo, un RDBMS, un linguaggio di programmazione etc..). Si suggerisce di far riferimento allo standard ISO/IEC 9126-1 per la definizione dei requisiti non funzionali applicabili al software.

Alle categorie di requisiti sopra individuate si aggiunge talvolta quella dei requisiti "inversi", ovvero ciò che il software non deve mai fare (descrizione che può essere comunque compresa in uno scenario di utilizzo del software).

Nel caso di sviluppo di software riusabile, devono essere esplicitamente definiti tra i requisiti anche quelli che ne caratterizzano la riusabilità, individuati in queste stesse linee guida.

In ogni caso, si raccomanda di inserire tra i requisiti anche questi elementi, importanti per facilitare il riuso del software:

- i criteri e le modalità di accettazione del software (come sarà verificato e collaudato),
- i criteri e le modalità di installazione del software nell'ambiente di destinazione

[16] Per *attore* si intende qualsiasi soggetto esterno all'applicazione: utenti umani, organizzazioni e istituzioni, altre applicazioni, sistemi hardware, sistemi software.

(richiesta di procedure di installazione guidate, di documentazione a supporto, identificazione di eventuali vincoli tecnologici e di coesistenza con altri prodotti etc..),

- le specifiche per la documentazione delle modalità di utilizzo del software da parte degli utenti (il manuale d'uso e di gestione operativa).

Sia la definizione dei requisiti funzionali che di quelli non funzionali possono avere ricadute sulla riusabilità del software: l'ampiezza del dominio funzionale coperto può infatti agevolarne il riuso, così come eventuali vincoli tecnologici associati al software ne possono limitare il riuso o renderlo più costoso (e.g. nel caso di uso di componenti applicativi e di sistema "certificati" solo su determinate piattaforme – la scelta di mantenere tali vincoli dovrebbe essere quindi sempre motivata dall'analisi dei requisiti). D'altro canto, prevedere una larga portabilità del software (su diverse piattaforme, sistemi operativi etc..), una sua facilità di coesistenza con altri software nel medesimo ambiente operativo, la sua facile manutenzione e modificabilità, sono fattori che aumentano la facilità di riuso.
Sempre ai fini della riusabilità, nella specifica dei requisiti l'attenzione va posta in particolare sulla definizione del "dominio" dell'applicazione, inteso come l'insieme di funzioni che devono essere messe a disposizione degli utenti. Infatti, se il software in via di realizzazione deve poter essere riusato anche in altri contesti, è opportuno considerare in modo estensivo il dominio applicativo. Si possono avere due casi:

- se sono già noti altri domini al cui interno l'applicazione potrà essere riusata, allora l'individuazione delle funzioni (e delle caratteristiche tecniche) dell'applicazione da realizzare dovranno considerare anche le esigenze di tali domini;
- se non si conoscono a priori i domini in cui l'applicazione potrà essere riusata occorre, per quanto possibile, generalizzare al massimo i requisiti che provengono dal dominio noto, in modo da "catturare" potenzialmente anche i requisiti (almeno in parte) di altri domini che, in astratto, potrebbero essere interessati al riuso dell'applicazione.

Non va dimenticato che la specifica dei requisiti deve descrivere anche i dati che il sistema software dovrà elaborare. La descrizione dei dati deve identificare, tra l'altro:

- i dati elaborati dal software, i loro attributi e la loro associazione con le funzionalità,
- i volumi di dati trattati,
- i dati da caricare al primo avvio del sistema,
- criteri di sicurezza dei dati,
- eventuali collegamenti con base dati esterne,
- glossario / dizionario dati,
- vincoli sui formati dei dati

Come detto, l'output della fase di raccolta e analisi dei requisiti è un documento che descrive in modo formale (o semi-formale) tali requisiti. Ai fini del riuso, è necessario che i requisiti raccolti e analizzati vengano descritti in modo standard, esaustivo e comprensibile, eventualmente facendo ricorso a notazioni standard o comunque di larga diffusione.

Alcuni standard forniscono un indice e una descrizione di massima di questo documento (ad es. lo standard ISO/IEC 12207). In ogni caso, il documento di specifica dei requisiti deve contenere, almeno questi elementi rilevanti per il riuso:

- la descrizione del contesto lavorativo per il quale è stato richiesto il software,
- la descrizione dei processi lavorativi del cliente che vengono impattati dal software da realizzare,
- la individuazione degli utenti del software, con la descrizione delle loro competenze ed esigenze specifiche,
- la descrizione dei requisiti (funzionali, non funzionali, relativi ai dati, inversi),
- un glossario e/o dizionario dei termini utilizzati.

È fondamentale che i requisiti siano descritti in questo documento in maniera non ambigua ed esaustiva. In particolare, occorre che:

- i requisiti siano identificati in modo univoco (ad es. con un codice o un acronimo) e correlati alla loro fonte (utente, contesto, tecnologia) e a chi li ha approvati (l'unità organizzativa del cliente);
- ne siano sempre censite le modifiche (chi le chiede, quando e perché, per garantire la tracciabilità dei requisiti e delle loro varianti);
- i requisiti siano tra loro collegati per evidenziare se la modifica ad uno di essi ha conseguenze su altri;
- i requisiti siano classificati per tipologia (ad esempio, funzionali, non funzionali, opzionali o irrinunciabili, tecnologici etc..).

È poi raccomandabile che i requisiti descritti nel documento di specifica siano identificati da queste informazioni, che ne assicurano, tra l'altro, la tracciabilità:

- Identificativo (un acronimo o numero progressivo).
- Tipologia di requisito (funzionale, non funzionale, inverso).
- Descrizione (sintetica) del requisito.
- Collegamento del requisito ad eventuali diagrammi esplicativi (ad es. casi d'uso UML e diagrammi di sequenza).
- Se requisito funzionale, dati in ingresso ed uscita alla funzione (identificazione dei dati, loro attributi, fonte degli input e archivio di destinazione per gli output, volumi in gioco).
- Se requisito funzionale, identificazione degli utenti della funzione.
- Criteri per verificare il soddisfacimento del requisito (i casi di test da utilizzare per la verifica).
- Fonte del requisito (chi lo ha espresso e quando).
- Razionale del requisito (perché è stato richiesto).
- Storia (e motivazioni) delle modifiche apportate nel tempo al requisito.

Una particolare attenzione va posta nel definire già da questa fase del processo produttivo i casi di test con cui sarà verificato che quanto prodotto soddisfi i requisiti. Ai fini di agevolare il riuso, anche questi casi di test dovrebbero essere a loro volta riusabili.
È importante associare ai singoli requisiti anche il *razionale*, ovvero la spiegazione del perché il requisito è stato definito nella forma corrente con cui è presente nel documento di specifica. Questa accortezza sarà particolarmente utile quando i requisiti verranno modificati, nel corso dello sviluppo o anche dopo il rilascio del software al cliente, ad esempio nel caso di un loro riuso in altri contesti. La presenza del razionale aiuta infatti a comprendere l'impatto che avrà sul sistema software una eventuale modifica di quel determinato requisito. È utile spesso anche tracciare una matrice di corrispondenza tra requisiti definiti e casi d'uso descritti nel documento di specifica.

8.3.2 Analisi delle alternative make or reuse/buy

Dal punto di vista concettuale, il riuso presenta alcune analogie con l'acquisto di un pacchetto software di mercato, e richiede quindi azioni valutative analoghe: occorre individuare e validare il componente da riutilizzare/acquistare e confrontare il costo del riuso/acquisizione con quello dello sviluppo *ex novo*.
Dalla specifica dei requisiti si è ottenuta la scomposizione (non ancora di dettaglio) del sistema software da realizzare in una architettura logico funzionale, che individua quali funzioni l'applicativo dovrà rendere disponibili agli utenti.
A questo punto del processo produttivo si hanno quindi le prime informazioni necessarie a valutare la possibilità di riusare componenti software già esistenti per realizzare alcune funzioni chieste dal cliente.
Per poter effettuare la scelta, è indispensabile avere le appropriate informazioni sui componenti candidati al riuso. In accordo con quanto affermato in queste linee guida, la ricerca dei componenti candidati al riuso deve essere effettuata nei cataloghi del software riusabile delle amministrazioni, ma può essere effettuata anche tra i prodotti proprietari e i pacchetti COTS, valutando in tal caso la convenienza economica della loro scelta. In particolare, per i componenti candidati al riuso si deve poter verificare quanto segue:

- il software da riusare deve essere descritto in modo esaustivo e aggiornato nel catalogo dal quale lo si intende prelevare,
- devono essere esplicitate presso il soggetto cedente eventuali modalità e responsabilità relative all'aggiornamento, manutenzione, evoluzione del componente da riusare (nel caso, prevedere un contratto di riuso tra soggetto cedente e soggetto riusante in cui queste modalità sono definite, prendendo come riferimento gli schemi di contratti di riuso resi disponibili dal CNIPA sul suo sito web).

Per determinare il costo di riuso di un componente, vanno considerati i costi necessari al suo adattamento ed integrazione nell'ambiente in cui andrà inserito e quelli per l'acquisto di eventuali licenze d'uso di prodotti proprietari dai quali il software da riusare non è separabile o senza i quali non funziona.

Nel caso in cui il software da riusare sfrutti pacchetti commerciali per fornire parte delle funzionalità applicative, va considerato che i produttori di tali pacchetti non offrono quasi mai la disponibilità del codice sorgente, per cui occorre considerare che potrebbe essere difficile apportare modifiche all'applicazione software che si intende riusare, a meno di ricorrere a skill specialistici od utilizzare linguaggi dichiarativi (che non richiedono di sviluppare codice).
Nel caso si stia valutando la possibilità di riusare prodotti COTS o componenti custom basati su prodotti COTS, va considerato anche che:

- la documentazione associata al software commerciale si può discostare notevolmente da quella per software custom, eventualmente progettata e realizzata esplicitamente per il riuso;
- vi possono essere problemi di non allineamento delle procedure e dei costi della manutenzione delle componenti custom riusate e delle componenti COTS cui la parte custom è vincolata.

Più in generale, quando si riusa un software in cui parte delle funzionalità sono rese disponibili da prodotti COTS, è consigliabile verificare la qualità di tali componenti COTS utilizzando le indicazioni contenute nello standard ISO/IEC 12119.
Quando si prende la decisione make or reuse/buy, si deve motivare la scelta fatta in un documento che elenca, oltre che i costi connessi al riuso di quel componente, quali altri componenti riusabili sono stati esaminati (indicando da quali cataloghi provengono) e la motivazione dettagliata per la quale il componente è stato selezionato per il riuso, ovvero determinati componenti "candidati" al riuso non sono stati considerati riutilizzabili.

8.3.3 CLASSIFICAZIONE DEI COMPONENTI SOFTWARE AI FINI DEL LORO RIUSO

Ipotizzando che i cataloghi dei software *custom* riusabili esistano e siano accessibili, come si può scegliere tra i tantissimi software disponibili nelle installazioni dei loro proprietari?
Proviamo a definire i principali elementi che caratterizzano un software e che devono costituire la griglia di decisione per chi vuole avviare un progetto di riuso. Questi elementi, sinteticamente, rispondono alle seguenti domande.

1. "Cosa fa" il software (le funzioni che offre all'utente); per utilizzare un gergo ormai entrato nell'uso comune dei progettisti, i "casi d'uso" e gli "scenari d'uso" del software sono sicuramente il primo elemento da valutare in caso di selezione di un software ai fini del riuso. Questi elementi costituiscono il c.d. "dominio funzionale" o "applicativo" del software; ogni applicazione software fornisce di norma molte funzioni: è ovvio che una situazione ottimale sarebbe quella in cui queste funzioni potessero essere anche singolarmente riusabili (ovvero fossero tra loro poco accoppiate, con poche dipendenze funzionali, ancorché correlate in una architettura logica). Un software più facilmente riusabile è quello che offre la possibilità di essere riusato ance solo in parte. Un passo avanti nella schematizzazione dei progetti di riuso sarebbe quindi il poter disporre di una tassonomia delle funzionalità che un sistema software

può fornire (ad esempio funzioni di accesso ai dati, di identificazione degli utenti, motori di ricerca, funzioni di georeferenziazione etc..), in modo tale che ad ogni sistema software sia associabile una architettura logico funzionale che evidenzi le componenti di servizio "standard" che offre (va da sé con il presupposto che le singoli componenti siano poi anche singolarmente riusabili);

2. quale è il "bacino di utenza" che può far uso delle funzioni offerte dal software (almeno nel senso di quali erano gli utenti "tipici" attesi per questo software al momento del suo sviluppo, al fine di valutare, in caso di riuso in un diverso contesto, un grado di "prossimità" che può essere un indicatore di quanto sarà agevole o meno riadattare il software per poterlo riusare).
   Una importante conseguenza di aumentare la grana del riuso, nel senso di effettuarlo come sopra detto a livello di singoli componenti funzionali che offrono specifici servizi, anziché a livello di interi sistemi software, è che l'importanza dell'analisi del bacino di utenza originario, ai fini del riuso, perde in tal caso parte della sua importanza. Se si costruiscono sistemi software assemblando in larga misura componenti funzionali non destinati specificatamente a un dato bacino di utenza, le componenti funzionali veramente legate a un dato bacino di utenza sarebbero residuali, abbattendo notevolmente i costi del riuso;
3. quali "caratteristiche tecniche" ha il software – linguaggio di sviluppo, tecnologie con le quali le componenti della sua architettura funzionano e comunicano tra loro, tecnologie per l'accesso ai dati e per la loro organizzazione etc.. Questi aspetti sono anch'essi importanti in caso di riuso per alcuni motivi:
   - tecnologie poco note e diffuse sono più difficilmente gestibili dal soggetto riusante, che potrebbe avere difficoltà nel reperire le risorse professionali necessarie;
   - tecnologie obsolete potrebbero non essere più mantenute nel tempo e creare in futuro problemi di aggiornamento e compatibilità con altri prodotti;
   - spesso gli sviluppatori tendono a creare delle dipendenze tra le funzioni applicative che realizzano e alcune tecnologie, che devono essere rispettate anche in caso di riuso, rendendolo meno efficiente e più costoso.

   Il software ottimale per il riuso è quello aggiornamento tecnologicamente, basato su tecnologie diffuse e "aperto" (che non significa ovviamente necessariamente *open source*) e che non ha legami obbligatori di dipendenza con determinate tecnologie; spesso, anche i software *custom* hanno un certo numero di dipendenze tecnologiche con prodotti "proprietari" di tipo COTS (e il cui utilizzo ha quindi un costo che il soggetto riusante dovrà considerare): ciò avviene in quanto per gli sviluppatori è più facile sviluppare partendo da librerie di semilavorati o comunque da elementi già disponibili forniti dai costruttori di software a pacchetto. Per inciso, vale la pena di ricordare che la dipendenza in questo caso non è solo da una data tecnologia, ma anche da un dato produttore che, essendo proprietario del prodotto, lo può modificare a proprio piacere. D'altra parte, i semilavorati "industriali" proprietari possono essere, a causa della loro diffusione, più affidabili dei componenti totalmente custom, ancorché "aperti".

La scelta del software da riusare dovrà quindi considerare, oltre che le caratteristiche tecniche in sé, anche il livello e il numero delle dipendenze tecnologiche e il grado di affidabilità dei componenti utilizzati;

4. queste ultime considerazioni introducono l'altra importante domanda che occorre porsi quando si esamina un software: "quanto costa". In termini generali, si può osservare che il prezzo di un software è legato alle modalità di sua diffusione: software COTS destinati a un ampio mercato hanno, grazie alle economie di scala dovute alla loro larga diffusione, un costo unitario (per ogni "copia" venduta del prodotto) relativamente basso (almeno rispetto ai costi di produzione), mentre per i software *custom* il prezzo di vendita al cliente e il costo di produzione non si discostano tra loro di molto (la differenza è nel *mark up* dello sviluppatore, che mediamente resta nei margini del 20%). Nel caso di riuso di software *custom*, il costo dello sviluppo, sostenuto dal soggetto che lo ha primariamente commissionato, non viene ribaltato sul soggetto riusante. Secondo la normativa italiana, se il soggetto cedente e quello riusante sono delle P.A. la cessione del software non è a titolo oneroso. Il soggetto riusante deve però valutare gli ulteriori costi che dovrà sostenere per riusare il software nel suo ambiente. Sinteticamente questi costi riguardano: adattamento, personalizzazione, parametrizzazione del software da riusare, formazione degli utenti all'uso, acquisto (se necessario) di prodotti COTS (o di licenze d'uso) verso i quali il software da riusare ha delle dipendenze non eliminabili, manutenzione ed evoluzione dopo l'installazione (questi ultimi costi non sono di norma a carico del soggetto cedente); l'insieme di questi costi può essere anche non banale e possono essere un elemento di valutazione per stabilire un *trade off* tra riuso di un software *custom* e l'acquisto di un prodotto COTS;
5. l'ultima domanda che occorre porsi in sede di valutazione di un software ai fini del suo riuso riguarda le "modalità di sua produzione". Si tende spesso a sottovalutare l'importanza di questo elemento, eppure i più diffusi standard del settore dell'ingegneria del software (ISO 9001 e 90003, ISO/IEC 15504, CMM, IEEE etc), affermano concordemente che un prodotto software è tanto più affidabile e di qualità (caratteristiche desiderabili ovviamente se lo si deve riusare) quanto più il processo di sua produzione è stato condotto in maniera controllata e standardizzata. Di norma, solo un processo di sviluppo condotto secondo le regole definite negli standard citati assicura che la documentazione progettuale sia stata realizzata in modo appropriato, che i test richiesti siano stati condotti in modo documentato, che il software e la sua architettura siano stati realizzati in modo standard (e quindi più facilmente comprensibile per chi dovrà adattarle ai fini del riuso), etc… La documentazione realizzata durante il processo di produzione non è di solito disponibile per i clienti dei prodotti COTS (anche se lo standard ISO/IEC 12119, che definisce la qualità di tali prodotti, afferma che parte di tale documentazione ad es. quella riguardante i test effettuati, dovrebbe esserlo), ma deve assolutamente essere resa disponibile ai clienti dagli sviluppatori dei prodotti *custom*.

Analizzando i vari prodotti software in relazione a coppie, ma anche a terne, delle coordinate sopra esaminate si potrebbe decidere circa l'opportunità di un loro riuso.
Analizzando ora queste relazioni dal punto di vista dello sviluppo di software riusabile, è evidente che alle coordinate fin qui individuate va aggiunta quella del "costo di produzione". Infatti, tutte le altre variabili incidono su tale costo e il committente del software "riusabile" ne deve tenere conto. Quanto più numerose sono le funzionalità offerte e i potenziali bacini di utenza, quanto più è sofisticato e aggiornato tecnologicamente il software – nonché portabile in diversi contesti tecnologici – e quanto più è controllato il processo di produzione e completa la documentazione realizzata, tanto più potrebbe essere costoso produrre il software. Questo costo, però, tende a decrescere progressivamente nel tempo se si realizzano cataloghi di componenti riusabili con i quali costruire per assemblaggio le nuove applicazioni. Tanto più numerosi saranno i componenti disponibili (e tanto più essi saranno facilmente adattabili a diverse architetture tecnologiche e logico funzionali), tanto più scenderanno i costi di produzione del software. Né va dimenticato che, se è vero che il costo di un processo di sviluppo controllato è per sé più alto di quello di un processo di lavoro meno formalizzato, a questo va aggiunto anche il costo della "non qualità", ovvero quello necessario a riparare i difetti residui nei software usciti processi produttivi non controllati - statisticamente molto più alti che nei processi formalizzati, fino a 10 volte.
Nella tabella che segue sono riepilogate le variabili da analizzare in caso di riuso di un software.

Elemento da valutare nel software candidato al riuso	Criterio di valutazione
Funzionalità offerte	Livello di prossimità delle funzioni offerte rispetto alle esigenze del soggetto riusante
Bacini di utenza	Livello di prossimità rispetto al contesto del soggetto riusante
Caratteristiche tecniche	Livello di aggiornamento tecnologico del software, livello di diffusione delle tecnologie su cui è basato, numero e complessità delle dipendenze tecnologiche che ha e che non possono essere agevolmente eliminate
Costo del riuso	Costo di adattamento, personalizzazione, parametrizzazione, costo delle dipendenze tecnologiche (acquisto di prodotti COTS o di licenze d'uso di prodotti COTS), costo di manutenzione ed evoluzione del software nel tempo (va osservato che nel caso di prodotti COTS questi ultimi costi hanno un costo per il singolo cliente piuttosto basso, grazie alle economie di scala conseguite, mentre sono alti nel caso di prodotti custom)
Modalità di produzione	Qualità della documentazione progettuale disponibile, completezza della documentazione dei test effettuati (e ricusabilità dei casi di test), affidabilità (eventualmente certificata da terze parti) del processo produttivo seguito dallo sviluppatore

8.3.4 PROGETTAZIONE

Nella fase di progettazione del software i requisiti che sono stati specificati dall'analista e validati dal cliente (*cosa* deve fare il software) vanno tradotti nel *come* il software lavorerà.
Il disegno (progettazione) del software è, di norma, un processo iterativo, che procede per raffinamenti successivi, partendo dal livello "alto", di astrazione dei requisiti per arrivare a definire aspetti del software via via sempre meno astratti, fino a produrre un documento di specifica tecnica che possa essere di guida alla realizzazione del codice sorgente e degli altri elementi previsti dal processo produttivo.
Il documento di progettazione tecnica deve descrivere come realizzare tutti i requisiti specificati nel documento di specifica dei requisiti approvato dal cliente. Questa coerenza è fondamentale ai fini di poter effettuare il collaudo del prodotto consegnato, alla fine dei lavori (il collaudo va fatto in relazione ai requisiti approvati).
Contenuti tipici del documento di specifica tecnica sono i seguenti.

1. L'architettura tecnologica e applicativa del sistema software.
2. La scomposizione dell'architettura del sistema software in sottosistemi funzionali.
3. Le classi o le funzioni che compongono ciascun sottosistema.
4. I metodi e gli attributi delle classi.
5. Gli algoritmi che risolvono gli aspetti procedurali posti a carico delle classi e delle funzioni. (Il comportamento dinamico del sistema software, come fluiscono i dati nel sistema e quali condizioni/controlli abilitano cambiamenti di stato nei componenti del sistema).
6. Le strutture dati e i relativi attributi dei dati che verranno elaborati dal software.
7. Le interfacce del sistema software verso gli utenti, verso altri software e tra i moduli componenti.
8. Il modello di dispiegamento dei componenti software sui componenti hardware.

Anche nella fase di progettazione tecnica possono essere prese decisioni in merito al riuso di componenti software già esistenti. Il livello di dettaglio a cui è giunta in questa fase la scomposizione dell'architettura del sistema applicativo permette infatti di individuare ulteriori potenzialità di riuso, specie per moduli che svolgono funzioni "di servizio" e generalizzabili.
L'architettura del software è un concetto in parte mutuato da altre discipline ingegneristiche e va intesa come la *organizzazione* del sistema software, in termini di suoi componenti, relazioni tra di essi, responsabilità loro assegnate, loro comportamento (in sostanza, è la "topologia" del sistema).
Per definire l'architettura di un sistema software, si possono utilizzare diversi approcci e "stili". Uno stile è un modello idiomatico per la rappresentazione dell'architettura, che comprende un vocabolario di componenti e di tipi di connettori, e un insieme di regole per la loro composizione.

Architettura tecnologica. L'architettura tecnologica deve evidenziare lo stack di prodotti (e componenti) che compongono il sistema software da realizzare, associando ad ogni elemento dello stack i rispettivi ruoli (si può vedere per questa operazione la descrizione degli stack tipici di un sistema software riportata in questo stesso documento). Ogni componente deve essere classificato come da "sviluppare ex novo", "riusato", o COTS. Di ogni componente devono essere evidenziate le caratteristiche tecniche, le relazioni con gli altri componenti e gli eventuali vincoli che hanno portato alla sua selezione (in particolare se si intendono utilizzare prodotti COTS).
Ai fini del riuso, particolare attenzione dovrà essere posta all'uso di tecnologie e metodologie che garantiscano l'isolamento logico tra il livello Abilitante e quello Funzionale dello stack.
Architettura applicativa (logico-funzionale). L'architettura applicativa definisce le funzioni offerte dai vari componenti del sistema software. Anche in questo caso, occorre descrivere i ruoli dei componenti, le loro interrelazioni, i vincoli, le interfacce, ricorrendo a diagrammi e notazioni standard (e di larga diffusione). Sono da prevedere in particolare diagrammi delle classi. Nella descrizione dell'architettura dovranno essere evidenziati gli eventuali punti di parametrizzazione e personalizzazione che consentano l'adattamento dell'applicazione a diverse situazioni.
È necessario che le soluzioni applicative e tecnologiche progettate trovino puntuale corrispondenza nei requisiti raccolti e specificati nella prima fase dello sviluppo del sistema software. A tal fine, vanno definite nel documento di progettazione delle apposite matrici di relazione tra requisiti e soluzioni progettuali e diagrammi che descrivono il comportamento statico e dinamico del software. Inoltre, nel documento di progettazione va analizzata la fattibilità della realizzazione, della gestione operativa e della manutenzione di quanto verrà sviluppato.
Ai fini del riuso, tra gli elementi di attenzione nella progettazione vanno segnalati i seguenti:

- scomporre il sistema da realizzare in moduli, in possesso degli adeguati livelli di coesione (alta) e accoppiamento (basso);
- usare l'astrazione per capire e analizzare i problemi da risolvere (modellare i problemi e le soluzioni, in modo da generalizzare le soluzioni);
- progettare software che possa essere facilmente modificato nel tempo (progettare per il "cambiamento");
- quando è possibile, dare una soluzione generalizzata al problema specifico (costa di più, ma ha vantaggi in caso di riuso).

Le specifiche OMG definiscono un componente (un modulo) come "una parte modulare e sostituibile di un sistema software, che comprende l'implementazione ed espone una serie di interfacce".
I componenti sono in sostanza i mattoni dell'architettura di un sistema software, ed eseguono le funzioni richieste al software, collaborando tra di loro e interagendo con altre entità

esterne al sistema (utenti, altri sistemi software). In termini di progettazione Object Oriented, un componente è un insieme coeso di classi (al limite è anche una sola classe).
Un componente è un elemento funzionale di un sistema software (un modulo), dedicato a svolgere una data funzione nell'architettura logica del sistema, che incorpora la logica elaborativa, le strutture dati che gli servono per svolgere le elaborazioni e le interfacce che servono a comunicare con altre entità e passargli dei dati. Ha un input e un output ben definiti.
Un componente è in grado di effettuare controlli, coordinando le chiamate ad altri componenti che gl servono per completare l'elaborazione che gli viene chiesta. Ha un ruolo nella gerarchia del controllo di un sistema software. I componenti, per poter essere facilmente riusabili, devono essere coesi e scarsamente accoppiati, rispettando i principi dell'information hiding e della indipendenza funzionale.
Come detto, la modellazione del software richiede uno sforzo di astrazione e rappresentazione delle soluzioni. In particolare, occorre, quando si progettano architetture di componenti:

- utilizzare modelli e stili architetturali (scegliendo quelli "giusti" per il contesto) per rappresentare la soluzione, sia dal punto di vista statico che dinamico,
- ricorrere alla "astrazione" per progettare concettualmente le soluzioni.

Stili e modelli devono essere standard e di larga diffusione, per quanto possibile, in modo da facilitare il riuso. L'indipendenza funzionale dei componenti ne facilita a sua volta il riuso.
Dal punto di vista della riusabilità della documentazione progettuale, questa può essere aumentata dalla sua completezza. A titolo non esaustivo, si raccomanda di rappresentare nel documento di specifica progettuale i seguenti elementi:

- l'architettura logico funzionale del sistema software, rappresentata attraverso diagrammi dei componenti UML, packages;
- il modello delle responsabilità funzionali, che rappresenta la distribuzione dei ruoli tra i vari componenti del sistema (diagramma delle classi UML);
- il modello dei processi eseguiti dal software, che rappresenta la sequenza di passi con i quali il software elabora le informazioni, al fine di produrre i risultati attesi (utilizzare diagrammi DFD, IDEF, diagrammi di interazione UML, diagrammi delle attività);
- il modello comportamentale, che rappresenta i possibili stati che possono assumere le componenti del software e gli eventi che provocano i cambiamenti in uno stato (utilizzare diagrammi di stato, ASF, Reti di Petri);
- il modello dell'architettura fisica, che rappresenta la distribuzione delle funzioni applicative sui sistemi hardware (la distribuzione dei compiti tra hardware e software, rappresentata attraverso diagrammi dei componenti e di dispiegamento).

Gli elementi di cui sopra dovrebbero, laddove significativo, essere tracciati rispetto ai casi d'uso permessi agli utenti dal software e definiti nel documento di specifica dei requisiti. È

poi indispensabile che tale documentazione sia aggiornata e coerente con la versione e configurazione del software consegnato per il riuso. Ogni modifica apportata nel tempo al documento di progettazione tecnica dovrebbe essere tracciata rispetto a chi la ha effettuata, quando e perché.

8.3.5 Realizzazione del software

L'aspetto su cui porre maggiore attenzione in questa fase, ai fini del riuso, è l'adozione e il rispetto di standard di codifica e la documentazione esaustiva e comprensibile dei manuali di utilizzo, manutenzione e gestione del software.
Nel caso in cui si utilizzino soluzioni basate, almeno in parte, su pacchetti applicativi proprietari, è necessario allegare o referenziare la documentazione tecnica di riferimento fornita dal produttore del pacchetto. Inoltre, se sono state effettuate personalizzazioni di prodotti COTS, è necessario documentarle in modo dettagliato.

8.3.6 Installazione e rilascio in esercizio

L'installazione del software deve essere supportata da apposite procedure, fornite dal produttore e realizzate tenendo conto dell'ambiente operativo e applicativo in cui va inserito.
Se l'ambiente lo richiede, devono essere previste e descritte le raccomandazioni relative ai componenti dedicati alla sicurezza (lettori di badge, smart card, chiavi hardware, interfacce standard o aggiuntive) e ai componenti di rete (schede di rete o altro).
Nel caso di procedure di installazione complesse, queste devono essere descritte in un apposito documento come sequenza di tutti i passi operativi da effettuare, con particolare attenzione alla descrizione dei layout di eventuali pannelli che guidano la procedura, da impostare con la descrizione dei parametri da inserire.
Prima di procedere all'installazione, inoltre, devono essere indicate:

a) le eventuali verifiche preliminari ed ex post da effettuare;
b) i livelli di autorizzazione necessari;
c) le procedure di caricamento iniziale della base informativa.

Al termine dell'installazione e della corretta conclusione della procedura, dovrà essere possibile una verifica degli eventuali effetti non evidenziatisi in modo immediato. Ad esempio, dovranno essere verificate le eventuali directory create su disco, le modifiche apportate ai file di sistema o ad altri file.
Se l'installazione comporta delle personalizzazioni sulle configurazioni standard dei prodotti e dei componenti che costituiscono l'applicazione, per ogni prodotto vanno indicati esclusivamente i parametri modificati.
Per quanto riguarda i componenti riusati, le procedure di installazione e la relativa documentazione devono essere disaccoppiate dall'intero processo o per lo meno rese facilmente enucleabili, al fine di poter installare separatamente i componenti in questione.

La fase del rilascio in esercizio deve essere supportata da un documento, redatto dal produttore del software, che descrive in maniera dettagliata ed esaustiva le competenze sistemistiche e applicative richieste al personale che svolgerà le attività previste in questa fase.
Devono essere descritti tutti i vincoli relativi all'ambiente operativo in cui lavora il software e alla compatibilità con altri prodotti (versioni, configurazioni etc..).

8.4 IL TEST NEL CICLO DI VITA DEL SOFTWARE

Il *testing* è parte integrante dei processi di produzione e manutenzione del software, e ha un ruolo fondamentale nel massimizzare la probabilità che il software realizzato risponda ai requisiti specificati dal cliente, senza incorrere in malfunzionamenti durante la vita operativa, dopo la messa in esercizio. Effettuare test fin dalle prime fasi del ciclo di produzione, e farli in modo sistematico, può ridurre considerevolmente anche i costi di produzione, permettendo di scoprire per tempo, e di correggere a costi ancora bassi, eventuali difetti. Tutto ciò ha forte influenza sulla facilità di riuso di un software: le attività di qualificazione del software da riusare hanno infatti un costi rilevante nel processo di riuso.
L'efficienza ed efficacia del processo di testing dipende da vari elementi: la chiarezza degli obiettivi delle valutazioni, la competenza di chi valuta, la disponibilità di strumenti adeguati, l'utilizzo di un processo di testing strutturato, basato su tecniche e metriche appropriate.
Anche il processo di testing ha però un suo costo, che deve essere ottimizzato e razionalizzato. A tal fine, le attività di testing vanno organizzate e condotte secondo modalità il più possibile standard, predisponendo test che siano anch'essi riusabili, in modo da contribuire ad abbattere i costi del riuso.

8.4.1 ATTIVITÀ DEL PROCESSO DI TESTING

Il processo di testing è descritto in diversi standard, tra i quali ISO/IEC 14598 *Information Technology, Software product evaluation*, (recepito anche in Italia come standard UNI) e lo BS 7924-2 *Standard for software component testing*.
Lo sviluppo di software riusabile richiede un'attenzione particolare per le seguenti attività connesse con il processo di testing:

1. definizione delle specifiche di test;
2. definizione del grado di copertura del test rispetto ai requisiti da valutare;
3. predisposizione del piano di test;
4. predisposizione dell'ambiente di test, ivi compresi gli strumenti per l'esecuzione automatica dei test;
5. documentazione dei casi di test,
6. tracciamento dei test e gestione delle misure rilevate.

Qui di seguito vengono brevemente descritti alcuni aspetti delle attività di cui sopra rilevanti ai fini della riusabilità.

8.4.2 Le specifiche di test

Le specifiche di test descrivono gli elementi che dovranno essere utilizzati per verificare che il software realizzato risponda ai requisiti espressi dal committente, nelle varie fasi del ciclo di lavorazione e nelle diverse situazioni d'uso dell'applicazione. Le specifiche di test dovrebbero descrivere la fase lavorativa in cui si esegue il test, gli obiettivi del test, identificare i componenti da sottoporre al test, i dati di ingresso, le azioni da effettuare (con indicazione di eventuali strumenti da utilizzare), i risultati attesi dall'esecuzione del test.
Le specifiche di test dovrebbero essere associate ai requisiti nel documento di specifica dei requisiti. Dovrebbe essere predefinito un dato livello di copertura dei requisiti con i test da effettuare. Dovrebbero essere definite le condizioni di uscita dal test (per risultato positivo o non positivo).
Dal punto di vista architettonico, si devono considerare le seguenti tipologie di casi di test:

- test di unità (su un singolo componente);
- test di integrazione (su una aggregazione parziale di componenti);
- test di sistema (sulla aggregazione di tutti i componenti del sistema).

Vanno quindi previsti test statici (da effettuare sul codice sorgente e sui documenti associati, in questo ultimo caso anche ricorrendo alla semplice lettura dei documenti) e test dinamici, che ispezionano il comportamento del software in esecuzione (utilizzando opportuni strumenti e tecniche).
Nello standard ISO/IEC 9126 (parti 2, 3 e 4) sono riportate numerose metriche che permettono di verificare in un software la qualità interna, esterna e percepita dall'utente.
Per garantire la tracciabilità dei test, è necessario predisporre una base dati del test che riporterà i casi di test, i risultati dei test (per ogni sessione di test effettuata – si noti che il numero di ripetizioni di un test dovrebbe essere definito a priori), le informazioni relative alla data del test, alla strumentazione utilizzata, all'ambiente operativo (e altre condizioni) in cui è stato effettuato, a chi è stato incaricato di effettuare il test. Informazioni sulle informazioni da prevedere di tracciare in un processo di testin sono nello standard ISO/IEC 14598.
Nelle specifiche di test devono essere previste anche le attività necessarie a predisporre un ambiente di test adeguato e funzionante (ad esempio, configurazione, parametrizzazione di prodotti e di componenti).
Le specifiche di test andrebbero sempre aggiornate per renderle coerenti consistenti con la versione corrente del software da ispezionare.
Attenersi a queste regole nella definizione delle specifiche di test, e fornire al soggetto riusante queste specifiche, aiuta considerevolmente le attività di testing da effettuare su un software in riuso presso un soggetto diverso da quello che lo ha originariamente prodotto.

8.4.3 I casi di test e la riusabilità dei casi di test

Ai fini di questo documento si assume che *il caso di test* rappresenta *una singola e specifica condizione di utilizzo del software che produce uno specifico risultato atteso, predefinito e conosciuto*. Può riguardare una funzionalità elementare o un insieme di funzionalità, a

seconda della tipologia di test che si sta implementando (test di funzione, di dominio, di scenario, ecc.) e deve specificare le condizioni di stato del software, dei dati e dell'ambiente prima dell'esecuzione, i dati di input e output, ovvero il risultato atteso.
Il risultato atteso descrive il prodotto dell'applicazione software a fronte dei dati in ingresso forniti e delle operazione ed elaborazioni svolte. Questa informazione include la messaggistica, anche di errore, generata dall'applicazione, le eccezioni, i valori di ritorno per l'utente, le condizioni finali sullo stato dell'applicazione, dei dati e dell'ambiente.
La definizione dei casi di test deve coprire tutti i requisiti funzionali e non funzionali identificati nella fase di analisi.
Per la loro essenzialità, i casi di test dovrebbero essere progettati e sviluppati in modo da garantire la loro ripetibilità e riusabilità nelle successive occasioni dove è richiesta la validazione del prodotto software da riusare.
I casi di test, per soddisfare questo requisito, devono essere progettati con caratteristiche di autoconsistenza, ossia di indipendenza da altri test e con un livello di dettaglio delle informazioni che garantisca la riesecuzione degli stessi da parte di personale diverso da chi ha progettato il test iniziale.
I principi di base da considerare per progettazione di casi di test riusabili, utili per ottenere maggiore solidità e facilità di manutenzione dei test, sono:

- progettare casi di test tra loro indipendenti e uto consistenti;
- progettare casi di test con un punto di partenza ben definito;
- progettare casi di test che non presentano sovrapposizioni tra loro.

L'indipendenza dei casi di test garantisce che non vi sia connessione sequenziale fra il successo di un test e quello dei successivi; nei test automatizzati, inoltre, limita al massimo l'intervento umano nelle fasi esecutive.

8.4.4 LIVELLO DI COPERTURA DEI TEST

Come sopra detto, il livello di copertura dei test rispetto ai requisiti dovrebbe essere definito già in sede di specifica di requisiti. Per trovare il corretto livello di copertura, occorre valutare il trade off tra esigenza di effettuare il maggior numero possibile di test e il loro costo (nonché i possibili ritardi che provocano alla conclusione del progetto). D'altra parte, numerose statistiche in ingegneria del software dimostrano che fare software senza difetti è impossibile. Il test è quindi una approssimazione ottimistica, che cerca sostanzialmente di ridurre il rischio di difetti residui nel software consegnato al cliente. Questi difetti, se rilevati in esercizio, o addirittura al momento del riuso in altri contesti, costituiscono una fonte certa di danno con elevato costo per la loro rimozione.
L'introduzione di una strategia di test basata sul controllo del rischio probabile, quindi, è una condizione essenziale per garantire la massima efficienza del test, ottimizzando il grado di riduzione del rischio e nel contempo una adeguata gestione delle risorse e dei tempi.
La strategia di test del tipo citato (Risk Based Testing) associa il concetto di rischio a qualsiasi tipologia di test e incide direttamente sui casi da predisporre, integrandosi con il con-

cetto di gestione del rischio già definita a livello di progetto e permettendo di intervenire all'interno del progetto stesso.

8.4.5 Il piano di test

Un riferimento esaustivo per la predisposizione dei piani di test è nello standard ISO/IEC 14598 e nello standard IEEE 829. In sintesi, il contenuto base di un piano di test deve essere:

a) i componenti del sistema software da sottoporre a verifica,
b) gli obiettivi del test per ogni componente, le caratteristiche indagate e il tracciamento dei test rispetto ai requisiti definiti dal Committente,
c) il processo di testing adottato (le attività e le sotto attività previste),
d) le tipologie di test cui sarà sottoposta ogni componente del sistema, con i criteri di ingresso e uscita da ogni test (i casi di test),
e) livello di copertura, report da produrre, metriche da utilizzare, numero di cicli di test previsti, il livello di rischio (classe di rischio) associato a ogni test,
f) le risorse professionali e strumentali che verranno impiegate per l'effettuazione di ogni test (ruoli e responsabilità),
g) i requisiti hardware e software per l'ambiente di test,
h) la procedura di registrazione dei risultati dei test e dei difetti rilevati,
i) eventuali vincoli che condizionano il testing,
j) la pianificazione temporale dei test (con indicazione anche del tempo stimato per effettuare ogni singolo test).

Il Piano di test deve essere reso disponibile al cliente nella sua prima versione al momento dell'approvazione delle specifiche del progetto. Il cliente deve approvare il Piano.
Il Piano di test evolve poi durante il progredire del progetto di sviluppo del software.

- Nelle fasi alte va definito cosa testare (identificazione dei casi di test in relazione ai requisiti),
- Nella fase di disegno del software si dettagliano i casi di test, definendo le specifiche dei test (test eseguibili e documentati opportunamente),
- Viene poi sviluppato l'eventuale codice di test,

Quanto realizzato viene via via approvato, revisionato, aggiornato e integrato da parte del cliente.

8.4.6 Documentazione dei casi di test

La documentazione di test deve essere corredata di tutte le informazioni propedeutiche a rendere ripetibile e automatizzabile il singolo test (ad es. base dati iniziale e finale, pre-condizioni a margine, ecc.); l'automazione dei test semplifica la valutazione del componente da

parte di chi volesse riutilizzarlo, minimizzandone tempi e costi. Per i moduli riusabili, se possibile, andrebbero prodotti anche gli unit test automatizzati.
La documentazione delle fasi del test deve utilizzare una struttura logica che può prevedere le seguenti attività:

1. *fase di setup*, che descrive i requisiti per avviare il test (ad esempio, il caricamento di dati sul database) e lo stato iniziale dell'applicazione ed è necessaria per rendere uto consistente e rieseguibile il test o per segnalare la sua relazione con altri test o funzionalità;
2. *fase di operazioni utente / macchina*, che descrive la sequenza delle azioni da svolgere, i dati da utilizzare e i risultati attesi da verificare durante le attività svolte;
3. *fase di riesecuzione* per condizioni diverse, nel caso sia necessario riciclare la stessa sequenza di operazioni per verificare la stessa o altre condizioni di test, con un insieme di dati diverso;
4. *fase di verifica del test*, per indicare quali altre verifiche sono previste per accertare l'esito del test oltre a quelle svolte direttamente durante la prevista sequenza di operazioni; a titolo di esempio si possono citare le verifiche di congruità sul database dei dati inseriti.

Oltre alla struttura logica, ogni caso di test deve essere progettato in base a contenuti 'oggettivi' e dettagliati, per garantirne la comprensione ed esecuzione da parte di altri utenti, anche con una conoscenza minima dell'applicazione sottoposta a test (aumenta la riusabilità del test).
Sono sconsigliati i casi di test del tipo "usa e getta", per non perdere l'investimento iniziale e per non dover reinventare nuove fasi di test riprodotte per necessità contingenti sulle versioni successive del software o sulla versione personalizzata per il suo riuso, che non sono mai rigorose come gli originali.

8.4.7 TRACCIABILITÀ DEI TEST

Il test, come detto, assolve anche compiti di validazione e di controllo sulla corretta implementazione di una specifica o di un requisito utente.
La scomposizione dei requisiti permette di definire una relazione diretta con i test che deve essere sempre evidente, in tutte le fasi delle verifiche durante il ciclo di lavorazione del software. Essa permette innanzitutto di pianificare risorse e tempi per la progettazione dei test, consente di definire e calcolare indicatori di qualità del software basati sul livello di copertura dei test e garantisce il controllo dei test effettuati.
Le eventuali anomalie identificate possono così essere immediatamente collegate ai requisiti, accelerando le indagini e le analisi per la loro rimozione e consentendo, inoltre, di stabilire anche i test necessari per verificare e validare gli interventi di manutenzione o di personalizzazione (come nel caso del riuso).
L'utilizzo di standard documentali e di strumenti di test, infine, consente di controllare sia la progettazione, sia i risultati del test, sulla base della copertura dei requisiti, in ogni momento delle attività di sviluppo.

8.4.8 Automazione dei test funzionali

La tecnologia attuale consente di optare per strumenti di test automatizzati, in grado di realizzare *'script di test'* per rieseguire i test funzionali senza il supporto umano.
In termini di efficienza ed economicità, i test automatici risultano più adatti a individuare errori durante i test di regressione, mentre i test manuali sono più idonei a trovare gli errori alle prime esecuzioni.
L'insieme di test iniziale dovrebbe essere il più produttivo nella ricerca dei nuovi errori, mentre i test di regressione permettono di trovare più errori legati all'evoluzione e manutenzione del software, ma spesso anche errori mai identificati, già presenti nella prima versione del software.
L'automazione del test impone un investimento iniziale maggiore rispetto al test manuale, se commisurato alla singola prima esecuzione, ma una volta implementato offrirà notevoli economie di scala per le successive esecuzioni, anche considerando i costi di manutenzione collegati.
Il valore aggiunto derivante dall'automazione dei test è proporzionale all'accuratezza con cui la fese è stata progettata. Infatti, la qualità della fase di test è indipendente dalla qualità dell'automazione che interviene quasi esclusivamente sugli attributi di economicità della fase stessa e non sulla sua efficacia ed esemplarità. Il beneficio finale, tuttavia, ne sarà sicuramente amplificato, in positivo, ma anche in negativo, perché un risultato scadente fornito dai test manuali può produrre, se automatizzato, soltanto un risultato peggiore e più costoso.
Il diagramma seguente mostra il variare degli attributi di qualità del test con l'introduzione dell'automazione.

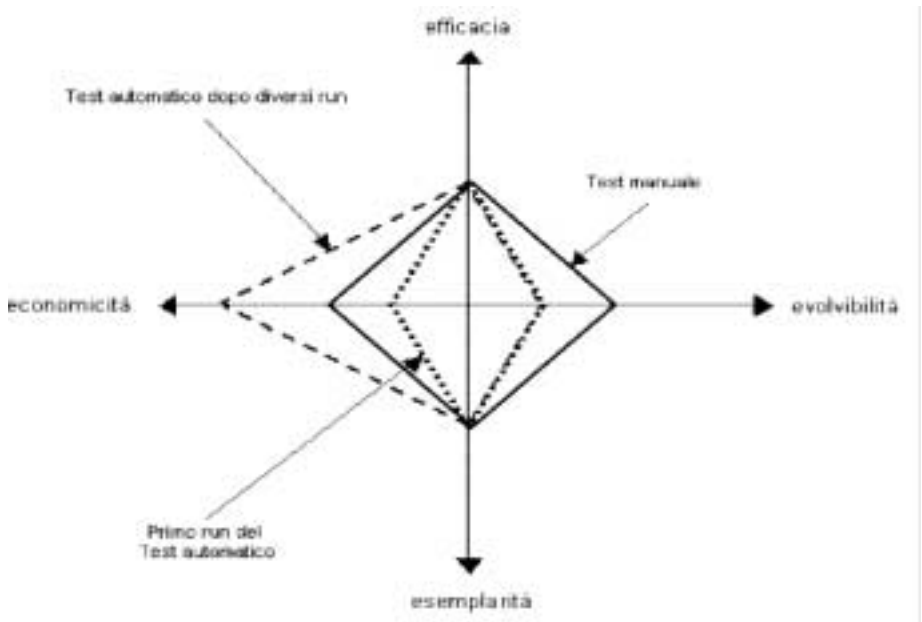


Figura 4 – Attributi di qualità del test manuale ed automatico

Più alti sono i valori sugli assi, più grande è l'area delimitata dai punti di incontro e quindi il qualità del caso di test. Un caso di test eseguito manualmente è rappresentato dalla linea continua. Lo stesso caso di test, quando viene automatizzato per la prima volta, assume un minore valore di evolvibilità e di economicità.
Dopo un certo numero di riesecuzioni, il livello di economicità si incrementa sensibilmente rispetto al test eseguito manualmente e rende il test automatico economicamente più vantaggioso.
Anche le modalità di sviluppo del test automatico hanno un'importanza elevata, considerati i successivi costi di manutenzione. Utilizzare script di test automatico realizzati con la sola tecnica di registrazione del test manuale, senza interventi di parametrizzazione o segmentazione delle componenti, producono script molto sensibili ai cambiamenti, anche minimi, e quindi espongono il sistema di test a costi elevati di manutenzione.

8.5 LA GESTIONE DELLA CONFIGURAZIONE DEL SOFTWARE PER IL RIUSO

La gestione della configurazione software deve mantenere la tracciabilità del progetto di sviluppo software, e quindi la riproducibilità in tempi brevi di un qualsiasi suo stato durante tutto il ciclo di vita. Nel caso specifico, la gestione della configurazione del software riusabile deve permettere all'Amministrazione, proprietaria del software, di estrarre dal catalogo, a richiesta, la versione "riusabile" del software, per l'implementazione di progetti di riuso e il deployment sull'ambiente *target*.
In generale, una "versione riusabile" o "baseline" del software, può essere ottenuta aggregando insieme i vari *elementi di configurazione*.
Gli elementi (Configuration Item – CI), tipicamente hardware e software, possono essere di vario genere) e sono un'aggregazione identificata come unità di configurazione (CSCI). Un CSCI può includerne altri a più basso livello (decomposizione gerarchica). Le "baseline" devono essere consolidate sulla base di criteri di riesami *formali* (review, configuration audit, ecc…).
La gestione della configurazione, in tema di riuso, ha un'importanza fondamentale. Infatti, in un progetto di riuso si utilizzano in misura significativa i cosiddetti NDI (Non Developmental Item), ovvero gli item già disponibili che non devono essere realizzati "ex novo". Nel caso in cui un NDI non è sviluppato, ma è semplicemente "scelto" (allocazione di requisiti, analisi costi/benefici e validazione) ed "usato" (integrazione nel prodotto finale), il progettista non ne conosce in dettaglio la struttura interna e risulta quindi essenziale il ruolo della Gestione di Configurazione. In particolare essa consente l'identificazione di ogni elemento costituente l'NDI, sia costruttivo (oggetto funzionale), sia descrittivo (manuali e documentazione di supporto), integrata eventualmente da documentazione interna.
Nel presente documento, non sono dettagliate le attività di gestione della configurazione da svolgere, né i tools da utilizzare per automatizzare, per quanto è possibile, tali attività. Infatti, una grande varietà di best practices e di strumenti per la gestione della configurazione è

ad oggi disponibile, per cui sembra arbitrario fare in questa sede una scelta al di fuori di uno specifico contesto operativo. Inoltre, è da ritenere che indicazioni categoriche sulla materia sarebbero troppo vincolanti per il "gestore" del Catalogo e forse onerose per le Amministrazioni.

Un sistema efficace per la gestione della configurazione, che potrà essere implementato solo combinando gli sforzi tecnico-organizzativi, delle strutture coinvolte, dovrà includere anche tutto l'ambiente utilizzato nel ciclo di vita, (file di configurazione, compilatori, assemblatori, ecc…), nonché tutta la documentazione software opportunamente associata ai diversi CSCI.

In relazione a quanto detto, il sistema di gestione della configurazione utilizzato nell'ambito di un progetto di riuso dovrà garantire le seguenti attività:

1) Identificazione della configurazione

Deve essere possibile identificare i CSCI che costituiscono la *baseline* riusabile, come questi sono stati prodotti e da quale organizzazione, nonché i requisiti applicativi e tecnologici a cui rispondono. Deve essere inoltre possibile individuare e acquisire tutta la documentazione associata.

2) Identificazione del supporto fisico

Deve essere possibile individuare il supporto fisico dove i *Configuration Items* risiedono e le modalità di accesso a tele supporto, nonché le regole di security applicabili. Devono essere garantite le procedure operative (Back-up, ecc…) atte a garantire la conservazione integrale dei suddetti supporti fisici a fronte di eventi accidentali o di attacchi dolosi.

3) Controllo di configurazione

Devono essere definite le tipologie degli utenti abilitati a utilizzare le funzionalità messe a disposizione dall'ambiente di gestione della configurazione e implementate correttamente le procedure per l'utilizzo delle funzionalità stesse. In particolare, devono essere definiti i ruoli delle varie Amministrazioni (cedente e utilizzatrici) e le responsabilità assegnate a ciascuna di esse (ad esempio, per quanto riguarda le attività di manutenzione evolutiva/correttiva) in relazione al relativo impatto sulla *baseline* del software. Deve essere chiaramente definita una figura di responsabile della configurazione, relativamente a una o più baseline software.

Inoltre, ogni elemento gestito nella configurazione del software deve seguire, durante il ciclo di sviluppo, un flusso di controllo corretto (normalmente, tramite un sistema a stati finiti) che dalla sua prima consegna in configurazione lo porti al rilascio conclusivo nella baseline di prodotto definitiva.

4) Mantenimento dello stato di configurazione

Deve essere garantito il mantenimento dello stato di configurazione, ovvero la consistenza della *baseline* riusabile, a seguito di interventi di creazione di nuovi CSCI e/o di modifica dei CSCI esistenti. In particolare, nel caso di sviluppo basato sul riuso, deve essere garantito che i nuovi oggetti siano anch'essi gestiti in modo coerente con il sistema di gestione della configurazione.

5) Gestione delle release e delle consegne

Le release e le consegne di nuovo software devono rispettare le regole del sistema di gestione della configurazione (sia da un punto di vista dell'identificazione sia da quello del

controllo di configurazione) e non devono corrompere le baseline esistenti (mantenimento dello stato di configurazione). La consegna di un elemento nuovo o modificato deve prevedere, per la sua messa in configurazione, uno specifico protocollo formale di approvazione da parte del responsabile della configurazione.

8.6 SINTESI DELLE ATTIVITÀ DEL CICLO DI PRODUZIONE CON MAGGIOR IMPATTO SULLA RIUSABILITÀ DEL SOFTWARE

Si riporta, nel seguito, una tabella che sintetizza le attività connesse al riuso che vengono svolte durante il processo produttivo. Per ogni attività viene indicata la fase di appartenenza, se l'attività produce software riusabile (PUT) o se lo acquisisce (GET).

Attività	Input	Output
Analisi dei requisiti Predisporre il riuso futuro del progetto corrente (PUT): classificare i requisiti tecnici e funzionali secondo classi predefinite	• Documentazione contrattuale	• Decisione sulla riusabilità intrinseca • Piano del riuso • Requisiti classificati secondo una tassonomia standard e riusabile
Analisi dei requisiti Eseguire l'analisi del riuso (GET): identificare a macro livello sistemi simili a quello che si intende realizzare in termini di dominio applicativo, progettazione o architetture runtime. Questa attività deve essere eseguita da esperti del dominio applicativo	• Piano del riuso • Archivio delle componenti riusabili • Matrice delle tracciabilità di progetti analoghi	• Strategia di riuso delle risorse (identificazione di requisiti riusabili) • Identificazione dei maggiori artefatti riusabili fra loro associati nella matrice di tracciabilità • Identificazione tool case abilitanti il riuso • Piano di riuso aggiornato
Analisi dei requisiti Creare primo entry nella matrice di tracciabilità (PUT)	• Matrice di tracciabilità vuota	• Matrice di tracciabilità con requisiti tecnico funzionali
Analisi dei requisiti Aggiornare il piano della qualità del progetto	• Strategia del riuso (Piano del riuso)	• Piano della qualità che include punti di verifica della strategia di riuso nei processi di ingegneria del software.

Attività	Input	Output
Analisi delle alternative make or reuse/buy Sviluppo dei requisiti software GET/PUT: identificare requisiti software che abbiano un potenziale riuso nei sistemi futuri analizzando quelli appartenenti ad applicazioni simili. Si prenderanno decisioni basate sul rapporto costo/beneficio valutando l'impatto anche sui tempi e costi e sul paradigma delle funzione esterna/interna al prodotto	• Requisiti tecnico funzionali • Piano del riuso	• Requisiti Software • Piano del riuso aggiornato
Analisi delle alternative make or reuse/buy Aggiornare matrice di tracciabilità dei requisiti (PUT)	• Matrice di tracciabilità con requisiti tecnico funzionali	• Matrice di tracciabilità con requisiti tecnico funzionali e software
Analisi delle alternative make or reuse/buy Identificare design pattern standard (GET/PUT) Si selezionano le alternative in base requisiti di riusabilità espressi. Eseguire analisi del rapporto costo/beneficio	• Standard architetturali di riferimento	• Standard architetturali per il progetto
Test e collaudo Identificare test case riusabili (GET) operando su matrici di tracciabilità esistenti, dalle quali si derivano i casi di test adeguati	• Matrice di tracciabilità	• Matrice di tracciabilità aggiornata
Test e collaudo Definire la strategia dei test (PUT)	• Indicazione delle componenti soggette a riuso	• Definizione del livello di automazione per i test • Artefatti di test riusabili
Test e collaudo Aggiornare matrice di tracciabilità (PUT)	• Matrice di tracciabilità con requisiti tecnico funzionali	• Matrice di tracciabilità con requisiti tecnico funzionali e software/casi di test

8.7 QUADRO DI RIEPILOGO DELLE ATTIVITÀ DEL CICLO DI PRODUZIONE DEL SOFTWARE RIUSABILE

Il contesto dello sviluppo di sistemi applicativi disegnato da queste linee guida è rappresentato nella figura successiva.

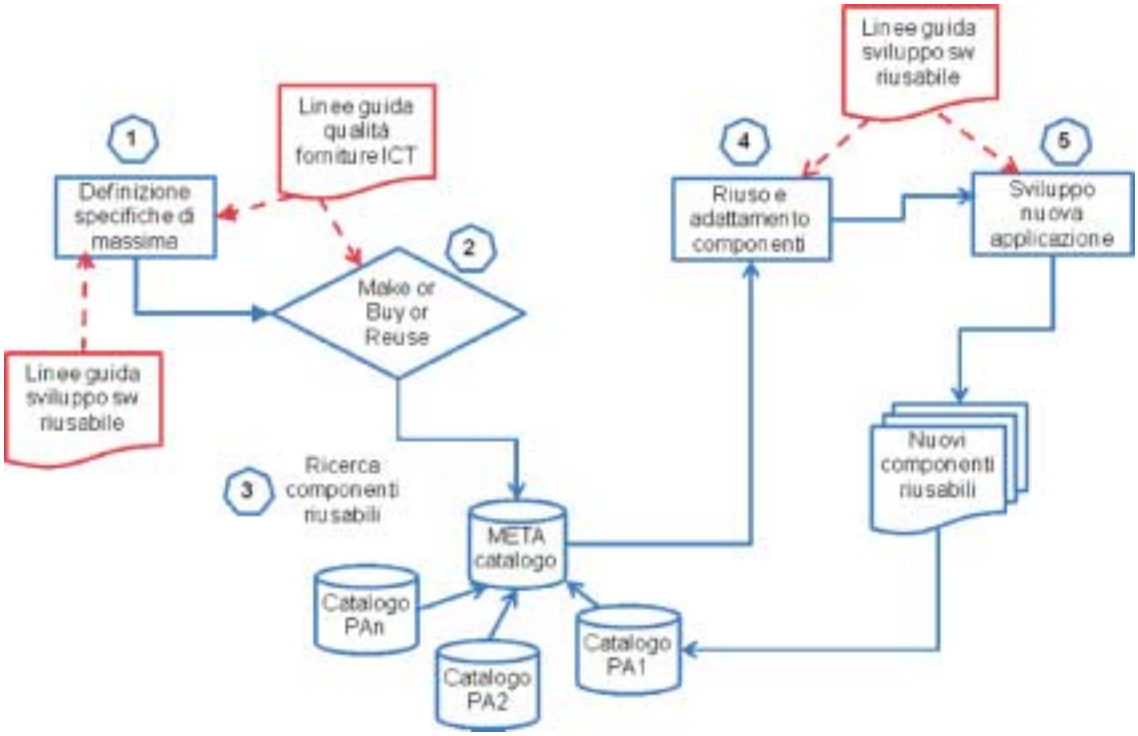


Figura 5 – Il contesto di sviluppo software custom secondo le linee guida

9. Catalogare componenti software riusabili

9.1 Il ruolo del catalogo

L'importanza, per ogni organizzazione, di poter disporre di un catalogo/repository del software che possiede, è universalmente riconosciuta dalla comunità dell'ingegneria del software, e in particolare da quella che si è occupata di riuso. Anche prima dell'ampia diffusione delle pratiche di sviluppo del software basate sugli oggetti e, più recentemente, sui servizi web, lo sviluppo di nuovo software partiva di norma dall'analisi del patrimonio software già nelle disponibilità dell'organizzazione produttrice. A maggior ragione, l'attuale indirizzo di e-government che privilegia le politiche di riuso nella Pubblica Amministrazione ha evidenziato la necessità di costruire, popolare e gestire cataloghi del software di proprietà della P.A., attribuendo agli oggetti software contenuti nei cataloghi l'attributo "riusabile", laddove possibile.
Queste linee guida, d'altra parte, prevedono esplicitamente che il processo di sviluppo di nuovo software riusabile abbia come riferimento un catalogo, dal quale prelevare componenti per costruire la nuova applicazione (se disponibili e adatti a tal fine) e nel quale far confluire, in modo controllato, al termine dei lavori, il software realizzato nel rispetto dei requisiti di qualità del riuso, definiti e discussi nei capitoli che seguono. Il catalogo deve quindi essere considerato non solo una "vetrina" del software di proprietà dell'Amministrazione, quanto uno strumento operativo concreto in grado di supportare i processi di sviluppo che si basano sul riuso come pratica costruttiva del software.
Altre funzioni che vanno assegnate al catalogo sono quelle di permettere la quantificazione (anche economica) del patrimonio software dell'Amministrazione, aumentare la diffusione delle informazioni tra le diverse amministrazioni riguardo ciò che è già disponibile per il riuso, permettere la costante valutazione comparativa della qualità degli artefatti che contiene.
Le fasi del processo di sviluppo software che interagiscono con il catalogo devono essere regolamentate e controllate, in particolare:

- l'interrogazione del catalogo per valutare l'opzione del riuso di componenti che contiene, ad inizio lavori,
- il prelievo dal catalogo di componenti da riusare,
- l'inserimento nel catalogo di componenti riusabili sviluppati,
- la manutenzione e modifica di componenti già presenti nel catalogo,
- la cancellazione di componenti presenti nel catalogo.

È opportuno prevedere che il catalogo possa contenere diverse versioni del medesimo componente, con caratteristiche che ne permettono la migliore adattabilità a vari contesti, noti a priori.

9.2 STRUTTURA DEL CATALOGO

Da un punto di vista concettuale, il catalogo è un'unica entità logica, con una interfaccia di accesso unificata; gli aspetti legati alla distribuzione dei dati e meta-dati che lo popolano sono mascherati da opportuni strati software. A titolo esemplificativo, si può ipotizzare che a livello architetturale il catalogo offra una interfaccia standard di accesso tramite un web server e, parallelamente, offra i suoi servizi anche tramite un insieme di web services, lasciando liberi i fruitori di progettare strategie di interazione adatte ai propri bisogni.
Per la classificazione ed il reperimento dei componenti riusabili (indicati come artefatti nelle specifiche OMG) ci si deve basare su uno schema concettuale in grado di descrivere compiutamente tutti gli artefatti, i loro attributi e le loro relazioni, quali, ad esempio:

a) componenti software (singoli moduli e/o intere applicazioni);
b) documenti di analisi e specifica dei requisiti;
c) documenti di progettazione funzionale e tecnica;
d) documentazione d'uso, manutenzione e gestione;
e) procedure di test e casi di test con l'indicazione delle specifiche dalle quali sono stati derivati e degli "oggetti" software su cui devono essere applicati;
f) procedure di installazione e configurazione;
g) documentazione relativa al comportamento del software durante l'uso da parte dell'utente dopo il rilascio.

Per facilitare il popolamento del catalogo, e quindi anche le operazioni di ricerca e di prelievo dal catalogo, si propone di adottare il recente standard OMG RAS (Reusable Asset Specification). Lo standard evidenzia, tra le altre cose, gli aspetti essenziali di un catalogo di componenti software riusabili, quali:

a) la struttura di un file XML (XML manifest) per la classificazione e l'accesso agli artefatti;
b) le quattro aree principali del catalogo: meccanismi di classificazione con i vari contesti, soluzioni presenti nel catalogo, procedure necessarie per l'utilizzo degli artefatti e collegamento a cataloghi similari;
c) i tipi di artefatti considerati, classificati tramite una struttura gerarchica a due livelli (primary type and secondary type);
d) i punti di intervento per adattare l'artefatto al contesto in cui viene riusato (variability point).

La struttura generale della proposta OMG va specializzata nel contesto specifico della Pubblica Amministrazione, costruendo una ontologia di dominio che permetta meccanismi di

classificazione e ricerca più efficaci. Coerentemente con la proposta OMG, ogni elemento riusabile deve essere corredato di:

- documentazione specifica per la fase di esplorazione e identificazione,
- documentazione specifica per il suo corretto riuso ed evoluzione,
- specifici moduli software dedicati ai test automatizzati (a solo titolo di esempio, Junit e TestNG per gli oggetti prodotti utilizzando la tecnologia Java, NUnit per la tecnologia .NET),
  - un documento di contesto che descriva in maniera esaustiva:
  - quali sono stati i requisiti funzionali e architettonici che hanno dato luogo alla realizzazione dell'elemento riusabile,
  - il contesto architettonico ove l'elemento possa essere considerato come chiave per una futura selezione di riuso,
  - la sua eventuale origine nel caso si tratti dell'evoluzione di un elemento già presente nel catalogo.

Quest'ultima considerazione implica che tra le funzionalità di base presenti nel catalogo esista un meccanismo per il controllo e gestione della configurazione.

9.3 LA GESTIONE DEL CATALOGO

La gestione del catalogo prevede le seguenti attività:

1) immissione di un artefatto riusabile, corredato da:
   - relativa documentazione;
   - collegamenti con altri artefatti;
   - classificazione rispetto all'ontologia di dominio definita per la P.A.
2) aggiornamento di un artefatto con la creazione di una nuova versione;
3) eliminazione dalla parte pubblica di un oggetto riusabile;
4) manutenzione dei manufatti contenuti nel catalogo;
5) modifica dei dati di classificazione ricerca;
6) modifica delle funzionalità offerte dalle interfacce di accesso.

Il processo di gestione del catalogo deve essere regolato e controllato con attenzione. Infatti, per sua natura, un catalogo di software riusabile serve un insieme ampio di soggetti, che a volte potrebbero operare in concorrenza per alimentare il catalogo (sia per inserire nuovi componenti che per modificare quelli esistenti). È possibile anche prevedere che i cataloghi contengano più versioni dei medesimi componenti, dedicate a essere utilizzate in ambienti specifici.
Perciò, è opportuno che la gestione del catalogo sia affidata ad un soggetto univoco, ad esempio al centro di competenza per il riuso attivo presso l'amministrazione.

Si può anche ipotizzare che il catalogo contenga due insiemi di artefatti: certificati dal centro di competenza e non certificati. Gli artefatti non certificati rappresentano le proposte recenti di aggiunta/modifica all'esistente e, come tali, vanno considerati ed utilizzati con opportune cautele. Gli artefatti certificati, invece, rappresentano soluzioni riusabili consolidate, il cui utilizzo può essere fatto con maggiore confidenza. Periodicamente, l'organismo di certificazione esamina gli oggetti non certificati e, ove ne sussistano le condizioni, li promuove al ruolo di artefatti certificati classificandoli rispetto all'ontologia del dominio di riferimento.

Questa politica di certificazione favorisce l'autonomia dei singoli contributori e, allo stesso tempo, garantisce la raccolta di un insieme di oggetti con elevato livello di qualità, evitando l'inserimento di artefatti scarsamente riutilizzabili, (basso livello di qualità, inefficace classificazione, scarsa documentazione, ecc.). Lo stesso organismo di certificazione è responsabile, poi, della manutenzione dei metadati, della interfaccia di accesso e del coordinamento delle attività della "officina del riuso".

Il corretto inserimento dei componenti riusabili nel catalogo è quindi un passo cruciale del processo di produzione del software riusabile, in quanto una errata o carente documentazione renderà difficile reperire gli oggetti di interesse. In mancanza di uno standard condiviso per la definizione e il popolamento del catalogo è opportuno prevedere almeno i seguenti meccanismi di catalogazione e ricerca:

- uso di un vocabolario controllato basato su caratteristiche elencate e su una classificazione di valori degli attributi;
- classificazione e ricerca a testo libero;
- classificazione e ricerca basata su una semplice ontologia dei domini applicativi.

10. Documentazione per gestire il riuso

10.1 Il Piano del riuso

La gestione del riuso all'interno di un progetto deve essere pianificata e monitorata in un apposito *piano del riuso* che imposta e descrive le decisioni, le azioni e i risultati delle attività finalizzate al riuso. Il piano deve integrare le proprie specifiche attività con quelle del piano di progetto che definisce il ciclo di produzione del software. Il piano di riuso fa inoltre riferimento al piano di qualità del progetto.
In linea generale si ipotizzano attività finalizzate al riuso all'inizio e alla fine di ogni fase, con revisione di quanto realizzato e produzione di documentazione *ad hoc*. Nella fase di messa a disposizione del software nel catalogo si terrà conto di quanto descritto nel piano del riuso.
Il piano del riuso definisce quindi tutti gli elementi necessari per il riuso del software, tra i quali:

1. scopo del riuso (ambito di applicazione e obiettivi specifici per il riuso);
2. elenco dei requisiti specifici ai fini del riuso;
3. elenco e tipologia di componenti e definizione delle librerie del software riusabile;
4. elenco delle attività specifiche e loro interazione con il piano di progetto;
5. definizione dei ruoli e delle responsabilità delle risorse del progetto.

Nel piano, quindi, sono descritti i componenti dell'applicazione da inserire nel catalogo e quelli che da esso saranno simmetricamente prelevati per essere riusati nelle attività di sviluppo.

10.2 Il libretto del riuso

Oltre alla documentazione del processo di sviluppo, il fornitore del prodotto software deve redigere il "libretto" del riuso, che fornisce in una scheda tecnica di sintesi gli elementi utili a verificare l'adeguatezza dell'applicazione o di qualche suo componente, in un'ottica di riuso.
Il libretto è una sorta di manuale operativo, aggiornato nel tempo, che funge da guida per le azioni di riuso per documentare e garantire il mantenimento livelli di qualità del software riusabile. e per valutarne l'utilizzo in altri contesti.

È predisposto dal primo sviluppatore ed aggiornato dai soggetti coinvolti in successivi progetti di riuso fra cui, ovviamente, è compresa anche l'officina del riuso.
Per i fornitori, la redazione e l'aggiornamento del libretto del riuso sono azioni includere espressamente nel capitolato.
La scheda tecnica di sintesi dell'applicazione riassume i seguenti aspetti:

a) modalità operative, sia nel suo insieme che dei singoli componenti,
b) dati interessati, nel loro insieme e nei singoli componenti,
c) vincoli tecnologici, organizzativi e normativi,
d) indicazioni sulle dimensioni e sui costi stimati per l'utilizzo.

Le informazioni nella scheda devono agevolare la classificazione dell'applicazione per il suo inserimento nel "Catalogo delle applicazioni riusabili". La scheda tecnica, inoltre, deve poter essere generata automaticamente, o con il minimo intervento manuale, in fase di documentazione del progetto per non incidere significativamente sui costi complessivi.
A seguire si declina una possibile struttura della scheda tecnica di sintesi.
Descrizione sintetica dell'applicazione. Questa sezione fornisce un quadro sintetico delle principali caratteristiche dell'applicazione, che saranno approfondite nelle successive sezioni.
Dati sintetici dell'applicazione. La sezione deve riassumere lo scenario di utilizzo dell'applicazione, evidenziandone i seguenti aspetti:

1. obiettivi;
2. elementi di contesto;
3. domini applicativi per i quali esistono significative opportunità di riuso;
4. parametri di riferimento;
5. funzionalità offerte, descritte in termini di servizi erogati all'interno di un dominio applicativo;
6. esistenza di eventuali vincoli e prerequisiti relativi a
   - tecnologia,
   - funzionalità,
   - caratteristiche d'uso,
   - interfacce disponibili.

Dati analitici della componente di servizio. Il capitolo deve contenere la descrizione dei singoli servizi forniti dall'applicazione, dettagliando le modalità di erogazione. In particolare occorre riportare:

1. nome e descrizione dei servizi erogati;
2. attori che interagiscono con essi (destinatari, erogatori);
3. descrizione del flusso dei servizi (modalità per reperire le informazioni erogate);
4. dati che il destinatario deve fornire per usufruire di uno specifico servizio;
5. livelli di erogazione dei servizi (ad esempio, livello 1: informazioni disponibili solo on-line; livello 2: informazioni disponibili mediante download di documenti);

6. canali con cui i servizi sono erogati (portale web, SMS, digitale terrestre).

Dati analitici della componente tecnologica. Il capitolo deve definire le specifiche tecniche dell'infrastruttura tecnologica necessaria per l'erogazione dei servizi. A tal fine devono essere illustrati in maniera sintetica i principali aspetti architettonici di ogni componente evidenziandone:

1. organizzazione logico/funzionale su più livelli (di presentazione, applicativa, di accesso ai dati);
2. tecnologia utilizzata per ciascun livello;
3. eventuali vincoli e prerequisiti tecnologici per l'utilizzo dei servizi;
4. documentazione e manualistica messa a disposizione.

10.3 Il Piano della Qualità del software

Il Piano precisa le particolari modalità operative, le risorse e le sequenze delle attività finalizzate a massimizzare la qualità di un prodotto o servizio secondo diversi punti di vista. Lo standard ISO 10005:2005, *Quality management systems - Guidelines for quality plans,* fornisce una guida alla stesura dei Piani della Qualità, in accordo con le specifiche dello standard ISO 9001. Il Piano della Qualità definito in questo standard indica precisi obiettivi di qualità attesi dal committente (i target posti alle caratteristiche del software) che sono riportati esplicitamente, di norma, nel Capitolato tecnico della procedura di acquisizione della fornitura, insieme alle metriche che misurano il livello di qualità effettivamente posseduto dalle caratteristiche del software e ai relativi indicatori che ne derivano.
Il processo suggerito per l'individuazione degli obiettivi di qualità e degli indicatori deve:

1. identificare le aspettative qualitative del cliente riguardo al prodotto da consegnare o al servizio da erogare;
2. definire gli obiettivi di qualità del prodotto o del servizio, espressi mediante una o più caratteristiche di qualità, per soddisfare tali aspettative;
3. segmentare le caratteristiche di qualità in sottocaratteristiche del prodotto o del servizio;
4. definire gli aspetti o i requisiti da valutare del prodotto o del servizio per ogni sottocaratteristica;
5. individuare gli indicatori e le soglie da utilizzare per valutare il soddisfacimento degli obiettivi di qualità, coerenti con gli aspetti qualitativi da valutare.

Il contenuto tipico del Piano della Qualità di un progetto di sviluppo software si articola nelle cinque componenti di base sintetizzate di seguito.

1. *Definizione degli obiettivi di qualità* corredati, ove possibile, da indicazioni quantitative, essenziali per la verifica del raggiungimento degli stessi. Gli obiettivi di qualità devono essere collegati ai criteri di accettazione indicati nel contratto.

2. *Lista delle attività di revisione*. Occorre descrivere in dettaglio i manufatti del processo produttivo che devono essere oggetto di revisione, il tipo di revisione da effettuare (ispezione, revisione formale), la data in cui la revisione deve essere effettuata, le persone coinvolte, i criteri di superamento dell'attività di revisione e le operazioni da compiere in caso di fallimento.
3. *Piano di test*. Vanno indicate le tipologie e la sequenza di test a cui verrà sottoposto il software, con le date ed il personale coinvolto.
4. *Test di accettazione* per il software sviluppato esternamente o riusato. Vanno indicate le tipologie e la sequenza di test a cui verrà sottoposto il software sviluppato esternamente o riusato.
5. *Gestione della configurazione*. Occorre indicare le procedure utilizzate per il controllo della documentazione e gli strumenti di supporto che saranno utilizzati

11. Organizzazione per favorire il riuso

La conoscenza è l'elemento abilitante per il miglioramento dell'efficienza operativa dei patrimoni applicativi della Pubblica Amministrazione. Nello specifico, tale conoscenza deve riguardare:

1. le dimensioni, la qualità, le caratteristiche tecniche e funzionali del proprio patrimonio software (inclusa la mappatura tra software e procedimenti amministrativi che servono);
2. la semantica dei dati e delle applicazioni software gestiti;
3. la riusabilità del patrimonio software.

Allo stato, poche amministrazioni pubbliche dispongono di tali informazioni e anche in quelle che hanno già realizzato il loro catalogo del patrimonio applicativo le modalità "semantiche" e "sintattiche" della descrizione di tali patrimoni è disomogenea, e non consente quasi mai la interoperabilità di tali cataloghi.
Presso il CNIPA è attivo un catalogo nel quale sono descritte (a livello di intera applicazione) i software riusabili della Pubblica Amministrazione, ma tale catalogo non contiene il software né i suoi contenuti sono descritti con descrittori formali (tipo xml) che consentano di effettuare ricerche strutturate su quanto contiene e che ne esplicitano la semantica.
Il problema, come già detto, è anche organizzativo: le funzioni per censire, classificare e gestire in modo strutturato il proprio portafoglio applicativo devono essere interne alle Direzioni responsabili dei sistemi informativi nelle quali va istituita una struttura operativa dedicata, che sia centro di competenza sul patrimonio applicativo e sul suo riuso e che abbia la responsabilità e le deleghe necessarie per operare
In termini di funzioni per il riuso, a questa struttura dedicata competerà la valutazione dei componenti da riusare (interni o anche esterni alla specifica amministrazione) e di quali sviluppare affinché siano riusabili (all'interno dell'organizzazione e/o in altri ambiti conosciuti). Solo a valle di tale attività sarà possibile, in un progetto, valutare l'opzione di sviluppare ex novo o acquisire prodotti di mercato, in base alle analisi specifiche di costi/benefici..
Un altro compito dell'unità organizzativa dovrebbe essere la certificazione delle componenti riusabili, la loro "pubblicazione" interna all'organizzazione di appartenenza, la loro "proposizione" in catalogo alle altre amministrazioni.
Vanno poi considerate anche tutte le componenti software che possono avere una valenza "inter-organizzativa" o appartenere a un progetto trasversale a più amministrazioni.

A questo livello superiore, occorre definire una struttura legittimata a coordinare le attività e a favorire le sinergie tra le organizzazioni aderenti. La struttura dovrebbe poter operare come centro di coordinamento per una rete (federata) dei diversi centri di competenza per il riuso, e dovrebbe disporre di strumenti conoscitivi trasversali e condivisi, quali i cataloghi federati o gli indici dei cataloghi per la ricerca distribuita. Ad essa, infine, spetterebbe una funzione formativa e di stimolo, per indurre quel cambiamento culturale necessario per introdurre la cultura del riuso nelle pubbliche amministrazioni.
Le funzioni caratteristiche dei singoli centri di competenza per il riuso sono riassumibili nel seguente elenco:

- gestione del catalogo dei componenti software riusabili nella propria amministrazione e loro correlazione con i procedimenti amministrativi per i quali sono stati sviluppati;
- certificazione e "pubblicazione" dei componenti riusabili;
- valutazione e scelta di quali componenti riusare nei vari progetti (provenienti dal proprio catalogo o da cataloghi esterni all'organizzazione);
- indicazione di quali componenti sviluppare con caratteristiche tali da poter essere riutilizzati (sia all'interno della stessa amministrazione che presso altre amministrazioni);
- indicazione sull'opzione tra sviluppo ex novo (con riuso) e acquisizione di pacchetti di mercato e/o software open source già esistenti:
- misurazione continua dei benefici raggiunti;
- formazione e diffusione della cultura del riuso.

L'attività del Centro di competenza per il riuso inizia già nella fase dello studio di fattibilità di un nuovo progetto di sviluppo software e consiste nella destrutturazoione logica e tecnologica del processo di business di interesse nelle sue diverse componenti software elementari. Per ciascun componente il Centro potrà selezionare e proporre le opzioni per la strategia di riuso più appropriata.
Questa progettazione architetturale e la selezione dei componenti applicativi, relativamente alla progettazione e riuso degli oggetti, presuppone che il Centro di competenza disponga di un complesso di competenze funzionali e tecniche tali da permettergli di intervenire in via preliminare rispetto alla progettazione esecutiva, in coerenza con le scelte tattiche e strategiche della Direzione dei sistemi informativi e dell'alta direzione.
Questa attività progettuale di alto livello a supporto dello sviluppo consentirà di utilizzare al meglio le risorse, informatiche, umane ed economiche, disponibili, sfruttando quanto già a disposizione (riusabile), e focalizzando le attività di progettazione e sviluppo per il riuso ai soli componenti che hanno una prospettiva in tal senso, e non all'intero progetto in maniera completa ed acritica.

12. Impatto del riuso sui costi di un progetto di sviluppo software

Per valutare l'impatto del riuso sul costo di uno sviluppo di software si possono utilizzare l'esperienza (stima per analogia), metodi analitici o algoritmici.
La stima per analogia è l'unica possibile nelle fasi preliminari del progetto, quando ancora non si dispone della scomposizione funzionale dell'architettura logica del sistema software. Si possono ipotizzare similitudini tra il progetto in esame e altri progetti già conclusi di cui sono noti i parametri dimensionali e fissare una percentuale (a forfait) di dimensione del software da sviluppare che potrà essere abbattuta grazie al riuso di componenti già esistenti.
Va osservato che tale percentuale deve tenere conto che una quota di analisi dei requisiti e di attività di integrazione e test devono comunque essere svolte. Tali quote sono in parte proporzionali alla quantità di componenti che verranno riusati, ma uno sforzo minimo di progettazione va previsto indipendentemente da tale quantità.
Si possono poi ipotizzare percentuali diverse di oggetti software "riusati" a seconda delle fasi del ciclo di sviluppo (e.g. una % di riuso di schemi progettuali, una & – diversa – di riuso di codice, una di casi di test etc..).
La percentuale di riuso così fissata a preventivo può essere utilizzata, ad esempio, per abbattere l'importo stimato della base d'asta o per abbattere a forfait – a consuntivo – la dimensione funzionale del software sviluppato, qualora non si intenda procedere con un calcolo analitico (calcolo che utilizza ad esempio metriche funzionali come i punti funzione).
Per poter utilizzare un approccio analitico per calcolare il risparmio derivante in un progetto di sviluppo di nuovo software dal riuso di moduli pre esistenti, occorre invece disporre della scomposizione funzionale dell'architettura logica del sistema da realizzare. Questa architettura è di norma un output della specifica dei requisiti.
Disponendo di tale scomposizione funzionale si può calcolare (conoscendola) la dimensione dei moduli che si ritiene possano essere riusati e non sviluppati ex novo, e da tale dimensione si può risalire al costo di sviluppo che può essere risparmiato, utilizzando costi unitari standard per l'unità di misura dimensionale utilizzata per il calcolo (punti funzione, LOC o altro, ad esempio classi o use case points) ovvero la produttività degli sviluppatori e tariffe di mercato per il loro lavoro.
Anche in questo caso, dal risparmio così calcolato deve essere detratto un importo che corrisponde a una quota parte di analisi dei requisiti, progettazione, sviluppo e test comunque necessari (per integrare i moduli da riusare nell'architettura del sistema). L'entità di tale

quota parte dipende dal numero di moduli da integrare, dal numero di funzioni che svolgono, dalle interazioni che hanno con gli altri moduli etc… e non è quantificabile a priori.
Un metodo algoritmico che permette di considerare tutti questi elementi nel calcolo del risparmio effettivo consentito da un progetto di sviluppo che riusa in parte moduli pre-esistenti è fornito nel Constructive Cost Model (CoCoMo II) ideato da B.Boehm.
Questo metodo deriva il costo dello sviluppo del software dall'impegno necessario a realizzarlo, calcolato con una formula la cui forma è definita su base statistica. Nella formula la variabile indipendente (il "cost driver") è la dimensione del software da sviluppare espressa in punti funzione o KLOC.
La formula è la seguente:

$Size_{reuse} = Loc_{orig} (AA + AAF + SU \times UNMF) / 100$

dove:

Loc_{orig} = LOC del software da riusare
AA = livello di analisi da effettuare per il riuso (varia tra 0-8)
AAF (fattore di adattamento) = 0.4DM + 0.3 CM + 0.3 IM
DM = % di modifiche necessarie al disegno del software da riusare
CM = % di modifiche al codice sorgente da riusare
IM = % di integrazioni da effettuare sul software da riusare
SU = livello comprensione del software (varia tra 0-50)
UNMF = livello di familiarità sviluppatori col software

Le regole per valorizzare AA, SU e UNMF sono fornite dal metodo CoCoMo.
In Appendice 3 è riportata una trattazione dei criteri da utilizzare per misurare in un progetto di sviluppo software il riuso funzionale tramite punti funzione.

Appendici

13. Appendice 1 – Il Reuse Maturity Model

Il SEI (Software Engineering Istitute) ha sviluppato il Capability Maturity Model per guidare le organizzazioni nel miglioramento del processo di sviluppo del software. Analogamente, è stato derivato dal modello base il "Reuse Maturity Model" (RMM).
L'organizzazione delle procedure di realizzazione del software è caratterizzata nel RMM da differenti gradi di standardizzazione e definizione. Relativamente alla capacità di riusare software preesistente, i processi adottati possono essere qualificati secondo cinque livelli.

1. Il livello 1 (riuso ad hoc) è tipico delle organizzazioni in cui il riuso si basa su tecniche informali, dipende dalla buona volontà del singolo progettista e viene deciso caso per caso in modo destrutturato; questo approccio è, normalmente, costoso e poco produttivo;
2. Il livello 2 (riuso formale del software) caratterizza le organizzazioni in cui processo di riuso del software è ripetibile, segue una approccio formale standardizzato, è sistematico, pianificato e produttivo. Normalmente, è introdotto all'inizio del ciclo di vita del software ed è un processo formale e ben documentato, basato su un processo ripetibile, con l'obiettivo di riusare tutti i sottoprodotti del ciclo di vita del software, dai requisiti all'implementazione.
3. Il livello 3 (implementazione formale del riuso) è caratterizzato dall'esistenza di un processo di riuso in cui esiste un catalogo popolato con beni certificati e validati. L'architettura generica del dominio di un progetto e dei sotto-domini è consolidata e sono implementate anche delle metriche di riuso. L'uso di metriche adeguate consente di misurare i benefici del processo di riuso del software in termini di qualità della produzione, migliore profittabilità e maggiore soddisfazione del cliente. I componenti comuni sono identificati e (re)ingegnerizzati nel rispetto dei requisiti. Il grado di riuso in nuovi progetti è determinato dall'ammontare, dalla qualità, dalle esigenze di manutenzione e dalla esistenza e disponibilità di beni che soddisfino i nuovi requisiti nello sviluppo e nella manutenzione dei sistemi software.
4. Il livello 4 (gestione del ROI) misura in termini economici il ritorno sull'investimento nelle tecniche per il riuso del software. Si usano le stesse metriche stabilite al precedente livello tre per determinare il successo dell'uso pratico dell'inventario di beni riusabili, ma per questo livello di approccio la priorità risiede nella determinazione e nel controllo del ritorno dell'investimento. Le scelte sono guidate dal costo dei

sotto-domini, dalla formalizzazione dei domini applicativi, nonché dal tasso di cambiamento dei requisiti. Questi elementi dovrebbero consentire la valutazione dei costi e l'opportunità non solo del riuso, ma anche della modifica di componenti riusabili.

5. Il livello 5 (ottimizzazione) caratterizza le organizzazioni in cui la progettazione del sistema prevede un processo operativo per il miglioramento continuo del software riutilizzabile.

Lo schema del modello è riassunto nella tabella che segue.

	1 Iniziale	2 Monitorato	3 Coordinato	4 Pianificato	5 Ottimizzato
Motivazione/ cultura	Riuso scoraggiato	Riuso incoraggiato	Riuso incentivato	Riuso nella cultura aziendale	Fondamentale nella produzione
Pianificazione	Nessuno	Nessuna in particolare	Opportunità considerata	Principale nelle scelte	Valutata nelle strategie aziendali
Ampiezza	Individuale	Gruppo di lavoro	Dipartimento / dirigente	Direzione	Aziendale
Responsabilità del riuso	Iniziativa del singolo	Iniziativa del gruppo	Personale dedicato	Gruppi dedicati	Direzione dedicata
Processo da seguire per il riuso	Caotico; non è chiaro come reperire il modulo da riusare	Si verifica la possibilità in fase di progetto	Progettazione basata sul riuso	Sviluppo di famiglie di prodotti	Tutto il software è generalizzato
Beni riusabili	Nessuna struttura per la raccolta	Esiste un catalogo di blocchi di codice ripartiti per linguaggio e piattaforma	Esiste un catalogo per famiglia di prodotti	Il catalogo include funzioni di ricerca specializzate	Si ricercano o sviluppano prodotti per alimentare il catalogo
Classificazione delle attività	Informale, individuale	Ogni gruppo classifica ed organizza secondo schemi indipendenti	Si diffonde uno schema univoco di catalogo	Si conducono ricerche per determinare le categorie da introdurre nel catalogo	Esiste una classificazione formale, completa e consistente
Tecnologia a supporto	Strumenti propri, se disponibili	Strumenti non specializzati per il riuso	Template per classificazione e sintesi dei contenuti	Libreria e tool specifici	Supporto automatizzato ed integrato nei tool di sviluppo
Metriche	Nessuna sul riuso, nessun confronto sui costi	LOC riusate in modelli di costo	Registrazione manuale delle occorrenze di riuso	Misurazione manuale dei risparmi derivanti dal riuso	Misurazione automatica del risparmio in base al software riusato
Considerazioni economico-legali	Nessuna, salvo la presentazione dei rischi	Centri di costo e ripartizione dei benefici	Viene premiato il miglioramento dei componenti in occasione di problemi	Si valuta il ricavo (figurativo) di una cessione in licenza dei componenti riusabili	Software come bene primario

14. Appendice 2 – La valutazione del software

Principi generali della valutazione del software

In termini generali, le valutazioni sul prodotto software sono da prevedere lungo tutto il suo ciclo di vita, dalla fase di analisi delle esigenze del cliente e progettazione della soluzione, fino alla dismissione del prodotto dall'esercizio, dopo che ha concluso il suo ciclo operativo ed economico, secondo un modello iterativo ed incrementale basato su cicli di tipo Plan-Do-Check-Act (PDCA).[17]

Si possono valutare semilavorati o prodotti software finiti, così come si valuta non solo il codice sorgente ma anche i documenti di progettazione logico funzionale e/o tecnica, schemi di basi dati, etc...

Le valutazioni sul software durante il ciclo di vita possono avere due finalità:

a) la *previsione* delle caratteristiche che avrà l'entità in esame in un momento successivo a quello della valutazione (ad es. si valuta il livello di complessità strutturale del flusso di controllo di un software in via di realizzazione per prevedere la complessità ed entità degli interventi di manutenzione che dovranno essere effettuati sul software dopo la sua consegna al cliente);

b) la *stima* delle caratteristiche che l'entità possiede al momento della valutazione (ad es. si valuta il livello di copertura funzionale di un documento di specifica, rispetto ai requisiti del cliente, oppure si valutano le prestazioni di un software durante il collaudo).

La valutazione, per essere significativa, deve essere basata su riscontri oggettivi ed essere condotta con metodi rigorosi e ripetibili. Gli elementi che caratterizzano una valutazione di questo tipo sono, soprattutto:

a) le metriche utilizzate, che assicurano che la valutazione si basa su riscontri quantitativi ed oggettivi,

b) le tecniche utilizzate e le specifiche di test, che assicurano della affidabilità, completezza e ripetibilità delle misurazioni effettuate,

c) il processo di valutazione utilizzato, che evidenzia la metodicità delle attività svolte per rilevare le misure,

[17] Per la definizione delle attività che compongono il tipico ciclo di vita del software si veda lo standard UNI CEI ISO 12207:2003, *Tecnologia dell'informazione – Processi del ciclo d vita del software.*

d) il piano del test, che certifica la trasparenza e le modalità organizzative delle attività di valutazione.

Ogni valutazione richiede, poi, la definizione di due elementi che in genere, sono specifici di ogni progetto:

- *i valori di soglia* rispetto ai quali confrontare le misure rilevate,
- *la scala di giudizio,* che permette di controllare e pesare gli scostamenti delle misure rilevate rispetto ai valori di soglia (meccanismo di *rating*).

Nel seguito del capitolo saranno sinteticamente discussi gli elementi sopra individuati. Preliminarmente vengono fornite alcune definizioni necessarie alla comprensione dei termini utilizzati.

ALCUNE DEFINIZIONI

Le definizioni che seguono sono tratte dagli standard ISO 9126, 14598 e 15939.[18]

Misurazione

L'uso di una metrica per assegnare un valore (un numero od una categoria) a un attributo, su una scala predefinita. In sostanza, la misurazione è il processo con il quale dei numeri sono assegnati agli attributi di una entità.

Misura

Valore assegnato ad una variabile a seguito di una misurazione.

Misura base

Una misura funzionalmente indipendente da altre.

Misura derivata

Una trasformazione di una misura base effettuata utilizzando una funzione matematica (funzione di 2 o più valori base).

Indicatore

Misura che può essere usata per stimare o prevedere un'altra misura.

Metodo di misurazione

Sequenza logica di operazioni effettuate per eseguire una misurazione.

Metodo di valutazione

Una procedura che descrive le azioni da compiere al fine di utilizzare delle misure per esprimere un giudizio.

Rating

Metodo per mappare un valore misurato su una scala di valutazione (ad es. "tempo di risposta" < 1 sec = "buono").

[18] ISO 15939:2002 *Software Engineering – Software measurement process.* I termini utilizzati dallo standard si basano sullo *International vocabulary of basic and general terms in metrology* (ISO DGuide 99999:2004).

Scala di misura

Un insieme ordinato di valori, continui o discreti, od un insieme di categorie, rispetto ai quali è mappato un attributo. In sostanza, è un insieme di relazioni empiriche mappato su un insieme di relazioni numeriche.

Metrica

Una unità di misura, una scala di misura ed il metodo utilizzato per la misura.

LE METRICHE

Ogni misurazione (e quindi ogni valutazione che ne segue) si basa sull'utilizzo di metriche. Lo standard ISO/IEC 9126 definisce infatti la misurazione come "*l'uso di una metrica per assegnare un valore (un numero od una categoria) su una scala predefinita*". In sostanza, la misurazione è il processo con il quale agli attributi di una entità sono assegnati dei numeri.

Nel caso del software, l'utilizzo di metriche permette di decidere in modo oggettivo se un elemento software possiede determinate caratteristiche – nel nostro caso i requisiti di riusabilità.

Le metriche disponibili in letteratura per effettuare misure sul software sono molte. L'efficacia delle valutazioni, tuttavia, non dipende di norma dal numero di metriche utilizzate, ma dalla loro significatività nel contesto di utilizzo. Le metriche vanno quindi scelte di volta in volta secondo il tipo di software in esame, gli obiettivi di misura che si hanno, le caratteristiche dell'ambiente di misurazione, i costi che si possono sostenere, i tempi a disposizione.

ISO/IEC 9126 fornisce tre insiemi di metriche per la misura, rispettivamente, della qualità interna, esterna ed in uso. L'elenco di metriche proposto in questa norma non è esaustivo e non tutte le metriche possono essere utilizzate in ogni contesto. La scelta va fatta secondo i criteri sopra accennati. Inoltre, è bene che un requisito sia valutato ricorrendo, ove possibile, a metriche diverse, che offrono differenti punti di vista sul medesimo fenomeno. L'associazione di più metriche può più compiutamente evidenziare le caratteristiche peculiari di un oggetto software. La scelta delle metriche da utilizzare deriva spesso da esperienze precedenti, e questo suggerisce di creare e mantenere un data base delle metriche che si utilizzano nei vari progetti.

Qui di seguito si forniscono alcuni principi base per il corretto utilizzo delle metriche nel caso di valutazioni sul software.

Le metriche possono essere:

- *Dirette*, se misurano una caratteristica del software rilevando direttamente un suo comportamento o un suo attributo, senza considerare l'influenza di fattori esterni, come l'ambiente (hardware e software) nel quale il software "gira" o il comportamento e le caratteristiche degli utenti etc... Esempio di queste misure sono quelle rilevabili leggendo il codice sorgente e quelle rilevabili da una lettura dei documenti di analisi e/o specifica.

- *Indirette*, se non operano direttamente sul software, ma sulla sua relazione funzionale con altre entità che si suppone gli siano correlate. Ad esempio, il grado di usabilità di un software è direttamente correlato alla misura della facilità con la quale un utente ne apprende l'uso.

Le metriche "di base" si possono suddividere anche in:

- metriche che misurano le "dimensioni" del software (punti funzione, LOC etc..),
- metriche che misurano il "tempo" nel quale l'oggetto software svolge le operazioni richieste, oppure il tempo necessario a correggere gli errori etc,
- metriche che misurano le "occorrenze" di una entità, come il numero di moduli che costituiscono il software, il numero di malfunzioni che si manifestano dopo il rilascio al cliente, il numero di difetti che provocano le malfunzioni etc.

Ogni metrica si può comporre combinando metriche di base:

tempo / dimensione – misura l'efficienza e l'impegno

n° occorrenze / dimensione – misura la densità di un fenomeno

dimensione / dimensione – misura l'incidenza di un fenomeno

dimensione / tempo – misura l'andamento nel tempo di un fenomeno

Gli *indicatori* sono altre misure specializzate, che "predicono" il comportamento del prodotto, in determinate condizioni, o stimano l'andamento di un fenomeno a partire dalla misura di un altro, con il quale è in correlazione.
La funzione e l'adozione di una metrica in un contesto di valutazione, e le modalità di suo utilizzo, devono essere descritte chiaramente ed in modo esaustivo in un capitolato tecnico.
Nel caso della valutazione della riusabilità del software, le metriche devono permettere di valutare nel software in esame il livello di possesso dei requisiti individuati in queste linee guida.
Ogni metrica utilizzata sarà quindi descritta dalle seguenti informazioni:

1. identificativo della metrica,
2. caratteristica/sottocaratteristica (attributo) che misura,
3. fonte di riferimento per la sua definizione,
4. unità di misura,
5. formula (o procedimento) utilizzata per il calcolo della misura, con descrizione degli elementi della formula,
6. interpretazione delle misure rilevabili
7. dati elementari da rilevare per elaborare la metrica,
8. fonte/i dei dati elementari da rilevare,
9. periodo di riferimento per la rilevazione dei dati elementari,
10. frequenza di rilevazione dei dati,
11. regole di campionamento,

12. regole di arrotondamento,
13. scala su cui sono proiettati i risultati (nominale, ordinale, a intervalli, a rapporti, assoluta),
14. tipologie di oggetti riusabili a cui si applica la metrica (ad esempio, documento di specifica, manuale d'uso, codice sorgente), e attività del ciclo di vita del software in cui è utilizzabile,
15. valori di soglia di riferimento.

I valori di soglia attesi per le metriche sono, di norma, specifici del contesto in cui vengono definiti, e devono essere scelti in base a considerazioni di ordine tecnico ed economico.
Di seguito vengono maggiormente dettagliati alcuni degli elementi descrittori di una metrica sopra individuati.
Fonte dati. I dati di base necessari per l'elaborazione di una metrica possono provenire da diverse fonti, ad esempio:

- requisiti utente,
- analisi del codice sorgente,
- risultati di prove e collaudi,
- analisi sull'utenza (ad esempio, rilevazioni di customer satisfaction o prove di "qualità in uso" di un prodotto), segnalazioni di problemi provenienti dall'utenza,
- rapporti sui problemi emersi, compresi quelli sugli strumenti di sviluppo e gestione,
- richieste di modifiche,
- verifiche ispettive e rapporti di assessment,
- riesami e controlli,
- dati operativi e funzionali del prodotto,
- condizioni interne o esterne al prodotto.

Periodo di riferimento. Le misure devono essere rilevate per un determinato intervallo temporale (non necessariamente continuo), che assicuri la significatività dell'informazione elaborata.
Frequenza di misurazione. Per ogni misura deve essere indicata la frequenza di rilevazione dei dati che la valorizzano. Periodo di riferimento e frequenza di rilevazione non sono parametri legati tra loro. La cadenza e la frequenza delle misure, infatti, è indipendente dalla lunghezza del periodo di riferimento per l'accumulo dei dati. A titolo di esempio, si considerino i seguenti tre casi, che prevedono due misure in un anno solare, a gennaio e giugno (frequenza semestrale), ma con un periodo di riferimento diverso.

1° caso

ogni misura è riferita ai dati raccolti nel bimestre immediatamente precedente (periodo di riferimento bimestrale); in questo caso, poiché l'anno non è coperto interamente, si hanno due momenti dell'anno in cui si possono prendere in considerazione i valori rilevati, sulla base di un campionamento su quattro mesi, pari 1/3 del tempo annuale di erogazione del servizio;

2° caso

ogni misura è riferita al semestre immediatamente precedente (periodo di riferimento semestrale); in questo caso l'intero anno è coperto e si hanno due momenti dell'anno in cui si possono prendere in considerazione i valori rilevati;

3° caso

ogni misura è riferita all'anno immediatamente precedente (periodo di riferimento annuale); in questo caso parte dei dati appartiene a più misure (è il caso tipico del calcolo di una media mobile per l'analisi di tendenza).

Dati elementari da rilevare. Sono i dati elementari necessari per il calcolo dell'indicatore. Per le misure di qualità percepita la valutazione è spesso numerica, pur riflettendo una percezione "soggettiva" che, per lo stesso fatto o evento tecnico, può variare da persona a persona. In tutti questi casi il dato elementare non deve essere filtrato, censurato o mascherato.
Regole di campionamento. I dati su cui è effettuata la misura possono essere tutti i dati esistenti oppure un loro sottoinsieme. Quando non è possibile o economico applicare la raccolta dati in modo sistematico si adottano regole di campionamento per creare una base dati di dimensione ridotta, ma statisticamente significativa. L'uso del campionamento introduce un'incertezza (rischio statistico) sulla misura effettuata, ben bilanciato dal vantaggio di effettuarla su un campione ridotto, rispetto all'intera popolazione. La scelta del campionamento (ove venga usata questa tecnica) è indicata assieme alle norme di riferimento e i livelli da esse indicati. Per esempio, se si utilizzano le norme UNI[19] si sottolinea come l'uso del campionamento sia un processo e non un evento sporadico, per cui il passaggio da un livello a un altro deve avvenire secondo le regole indicate dalle norme.
Formula. Devono essere indicate la formula o le formule di calcolo adottate per elaborare, integrare, riassumere, la molteplicità dei dati elementari per renderli espressivi dell'oggetto della misurazione e quindi dell'indicatore di qualità.
È opportuno che la formula sia semplice per favorirne l'immediata comprensione, o – nel caso di formule complesse – che si riesprima o si scomponga la formula di partenza in una forma possibilmente meno complicata. Questo significa, ove possibile, evitare le funzioni logaritmo e esponente, potenza e radici, ricercando principalmente l'uso delle quattro operazioni aritmetiche, di proporzioni e percentuali, di medie aritmetiche o ponderate.
Formula o procedimento. Devono essere indicate la formula o le formule di calcolo adottate e i passi da svolgere per elaborare, integrare, riassumere la molteplicità dei dati elementari per renderli espressivi dell'oggetto della misurazione. È conveniente che tali formule siano semplici per favorirne l'immediata comprensione. Questo di solito significa evitare le funzioni logaritmo, esponenziale, elevazione a potenza e radice, limitandosi all'uso delle quattro operazioni aritmetiche, percentuali, medie aritmetiche o ponderate.
Regole di arrotondamento. Devono essere indicate le regole di arrotondamento (quante cifre decimali significative impiegare, come arrotondare a un certa cifra) dei dati elementari

[19] In particolare, la norma ISO 2859-0:2001 *Procedimenti di campionamento nel collaudo per attributi - Introduzione al sistema di campionamento per attributi della UNI ISO 2859.*

e delle misure risultanti dalle formule di calcolo, ai fini dell'utilizzo dei dati per successive sintesi.
Valore di soglia. Ad ogni indicatore di qualità è associato un valore di soglia (obiettivo) che rappresenta il limite di controllo. Esso può essere:

- valore atteso maggiore di X;
- valore atteso minore di Y;
- valore atteso compreso tra i due valori X (maggiore di X) e Y (minore di Y).

Inoltre possono essere presenti più limiti per la stessa misura, ad esempio:

- primo limite, se superato si genera un allarme di gravità bassa;
- secondo limite, genera un allarme più intenso e avvia le procedure di intervento previste a questo livello di scostamento;
- altri limiti successivi, con conseguenze proporzionate allo scostamento rispetto al valore di soglia disatteso.

Sono quindi indicati gli obiettivi per l'indicatore di qualità, i relativi valori soglia e le eventuali eccezioni.
Si noti che in un Capitolato possono essere anche definite delle *Eccezioni* ovvero quelle circostanze particolari che limitano, vincolano, sospendono o ritardano l'applicazione del sistema di rilevamento dell'indicatore.

LE TECNICHE DI VALUTAZIONE

Le tecniche di valutazione del software devono permettere di effettuare verifiche su tutte le sue componenti significative (per il contesto), quali:

- il codice sorgente,
- la documentazione d'uso, operativa e di manutenzione,
- i dati interessati al processo elaborativo,
- gli elementi di processo che influenzano la qualità del prodotto,
- gli effetti rilevati sull'uso del prodotto,
- la qualità in uso.

La scelta della tecnica dipende dallo stadio di lavorazione del prodotto e dagli obiettivi della valutazione (valutazione della qualità interna, esterna o in uso e da quale caratteristica si vuole misurare).
La tipica sequenza di attività di testing prevede in sequenza:

1. un iniziale test di unità,
2. un test di integrazione,
3. un test di sistema,
4. un test di accettazione.

Successivamente, il software diventa operativo presso il cliente e le attività di test sono relative ad operazioni di manutenzione, verificando che le modifiche effettuate dagli interventi di manutenzione abbiano raggiunto il loro obiettivo e non abbiano introdotto degli errori prima non esistenti (test di regressione).
Nel seguito si analizzano le attività di test appena descritte, evidenziando gli aspetti significativi per la valutazione della riusabilità del prodotto, con particolare riferimento ai test di tipo funzionale.

Test di unità

Il test di unità si basa su attività ispettive (ispezione del codice, *walkthrough*), sull'analisi della struttura del codice sorgente (test a scatola bianca), sull'analisi dei dati in ingresso e uscita (test a scatola nera) e sull'analisi delle funzionalità prevista (copertura delle funzionalità).
Durante le attività ispettive è possibile rilevare le seguenti metriche utili per valutare le caratteristiche di qualità indicate:

a) Proporzione dei componenti autoconsistenti: modularità;
b) Numero di variabili globali: modularità;
c) Coesione delle classi: modularità, modificabilità, analizzabilità;
d) Semplicità delle classi: modificabilità, analizzabilità;
e) Densità dei commenti e standardizzazione della codifica: modificabilità, conformità a standard di codifica, analizzabilità.

Durante il test glass box è possibile utilizzare le seguenti metriche:

a) Complessità ciclomatica: analizzabilità, testabilità;
b) Percentuale dei test automatizzabili: testabilità;
c) Proporzione dei casi di test osservabili senza modifiche: testabilità.

Durante il test a scatola nera (black box) – che copre i requisiti funzionali – è possibile utilizzare le seguenti metriche:

a) Percentuale di funzioni elementari (o requisiti funzionali) con casi di test correttamente eseguiti: testabilità;
b) Proporzione dei casi di test osservabili senza modifiche: testabilità;
c) Percentuale dei test automatizzabili: testabilità.

Test di integrazione

Durante il test di integrazione ci si concentra sulla struttura delle chiamate tra i moduli, producendo opportuni *driver* e *stub* sino a giungere ad una prima versione funzionante del sistema. In questa fase è possibile utilizzare le seguenti metriche:

a) Proporzione dei formati di dati per lo scambio di informazione correttamente implementati: interoperabilità;

b) Proporzione dei protocolli di comunicazione per lo scambio di informazione correttamente implementati: interoperabilità;

c) Accoppiamento tra le classi: modularità, manutenibilità, modificabilità, analizzabilità;

d) Proporzione delle dipendenze simulate con stub: testabilità.

Test di accettazione

Il test di accettazione viene svolto dall'utente finale nell'ambiente in cui l'applicazione funzionerà a regime e viene effettuato per validare l'aderenza ai requisiti (funzionali, non funzionali e di qualità). In questa fase è possibile utilizzare le seguenti metriche:

a) Proporzione di funzionalità indipendenti dall'ambiente hardware e/o software di base: adattabilità;

b) Proporzione di funzionalità indipendenti dalla organizzazione: adattabilità;

c) Numero di parametri di configurazione: configurabilità;

d) Numero di ambienti operativi nel quale il sistema può essere installato: installabilità;

e) Proporzione di passi di installazione automatizzati: installabilità;

f) Proporzione di passi di installazione che possono essere facilmente ripetuti: installabilità;

g) Proporzione dei prodotti con i quali l'applicazione coesiste: coesistenza;

h) Numero di errori riscontrati sulle applicazioni coesistenti: coesistenza;

i) Tempo medio di apprendimento: usabilità;

j) Proporzione dei task eseguiti correttamente utilizzando il manuale: usabilità;

k) Proporzione dei task per i quali l'utente ha rintracciato le documentazione corretta: usabilità.

Manutenzione e test di regressione

In questa fase si effettuano i test sui moduli modificati e sull'intero sistema per verificare che non siano stati introdotti nuovi malfunzionamenti; è quindi possibile utilizzare le seguenti metriche:

a) Sforzo medio in ore uomo per modificare una linea di codice: modificabilità;

b) Proporzione dei casi di test applicati in modo automatico durante il test di regressione: testabilità.

IL PROCESSO DI VALUTAZIONE DEL SOFTWARE

La valutazione del software non è una fase specifica del ciclo di vita del software, ma è un processo trasversale a tutte le altre attività del ciclo di vita, che deve essere portato avanti per tutta la vita operativa del software, in parallelo con le attività "primarie" del processo di produzione.

Data la sua rilevanza nel determinare la qualità del prodotto in uscita dal ciclo di lavorazione, il processo di valutazione è stato oggetto di standardizzazione da parte di vari Enti (ISO, IEEE, Nato etc..).
Verranno qui presi come riferimenti principali per la definizione del processo di valutazione della riusabilità del software gli standard ISO/IEC 14598 *Information Technology, Software Product Evaluation* e ISO 15939:2002 *Software Engineering – Software measurement process*.
Lo standard ISO/IEC 14598 è composto da 6 parti, recepite in Italia da UNI, che nel loro insieme forniscono una guida al corretto svolgimento dei processi di valutazione del software ed una serie di strumenti di supporto alla valutazione, come lo scheletro dei piani di valutazione e degli altri documenti da produrre. Il processo di valutazione definito nello standard può essere utilizzato per verifiche sulle caratteristiche ISO/IEC 9126 o per valutare la presenza nel software di qualsiasi altra caratteristica. Inoltre, è applicabile sia ai prodotti commercial-off-the-shelf (COTS), sia a quelli sviluppati su commessa in base a specifici requisiti utente (custom software), sia alle personalizzazioni di prodotti esistenti.
Il processo di valutazione definito da questi standard prevede una serie di attività che possono essere logicamente raggruppate in 4 fasi, definizione dei requisiti della valutazione, specifica della valutazione, impostazione del piano di attività di testing ed effettuazione dei test e delle valutazioni previsti dal piano.
Più in dettaglio, le fasi sopra individuate sono articolate come segue:

1) definizione dei requisiti di contesto della valutazione; è la fase in cui il processo di valutazione che sarà seguito durante lo sviluppo del prodotto viene inquadrato e descritto nei suoi aspetti generali; la definizione va effettuata tipicamente prima di iniziare le attività di sviluppo del software (ad esempio in fase di definizione dei documenti di gara per l'affidamento del servizio di sviluppo software o di produzione dell'offerta tecnica in risposta ad un bando di gara) e produce un documento che fissa il contesto di riferimento per la successiva definizione delle specifiche di test e del piano di test; nel documento, denominato "requisiti di contesto della valutazione", sono descritti, almeno:
   1.1) lo scopo per cui saranno effettuate valutazioni durante il processo di produzione del software,[20]
   1.2) gli oggetti da valutare, inclusi:
      - livelli di integrità richiesti al prodotto (in funzione delle conseguenze di malfunzioni), che influenzano il rigore della valutazione,
      - descrizione delle interfacce del prodotto e dei requisiti delle interfacce (ad es. il tipo di dati che passano attraverso l'interfaccia, il formato dei dati, i meccanismi di accesso all'interfaccia, le modalità di gestione degli errori etc…),
      - requisiti di integrazione del prodotto in altri contesti e sistemi, se richiesto;

[20] Tra i possibili scopi vi sono l'accettazione e/o il collaudo di un prodotto, il confronto di un prodotto con altri concorrenti (benchmark), la valutazione sulla convenienza a dimettere un prodotto dall'esercizio, la valutazione sulla opportunità di apportare modifiche ad un prodotto già esistente etc..

1.3) gli utenti del prodotto e i loro obiettivi, i compiti che svolgono e che vanno informatizzati, l'ambiente di utilizzo del prodotto,
1.4) il modello qualitativo di riferimento (nel caso della riusabilità le sotto-caratteristiche di riusabilità individuate in queste linee guida),
1.5) i requisiti di qualità che il prodotto deve rispettare, espressi in termini di livello di possesso delle sotto-caratteristiche,
1.6) i vincoli di contesto nel quale si svolge la valutazione, tra cui il budget disponibile, il tempo, la disponibilità di risorse;

2. definizione delle specifiche di valutazione; in questa fase viene definito in modo "operativo" come verrà accertato che il prodotto risponde ai suoi requisiti; le specifiche di valutazione vanno definite insieme a quelle di progettazione del prodotto (dovrebbero essere parte integrante del documento di specifica dei requisiti), e devono garantire la valutazione, con almeno un test, di tutti i requisiti del prodotto, per avere un adeguato livello di copertura; questa fase del processo di test produce il documento di "specifica di test" in cui, per ogni requisito del prodotto, vanno definiti, almeno:
   2.1) i casi di test da eseguire (le metriche da utilizzare, le prove da eseguire),
   2.2) le tecniche per rilevare le misure necessarie ad elaborare le metriche,
   2.3) le fonti da cui rilevare i dati elementari necessari alle misure,[21]
   2.4) la consistenza del campione osservato,
   2.5) le elaborazioni che verranno effettuate sui dati rilevati (aggregazioni, operazioni, arrotondamenti etc…),
   2.6) i criteri di *rating* per mappare le misure rilevate su una scala di giudizio,
   2.7) l'ambiente operativo (hardware, software di base etc..) necessario a svolgere i test previsti dalle tecniche selezionate,
   2.8) la modalità di gestione delle eccezioni e dei problemi;
   Si noti che le specifiche di test vanno definite dal fornitore ma approvate dal cliente, unitamente alle specifiche del software.

3) definizione del piano di testing;[22] il piano deve essere predisposto dal fornitore appena lo stato di avanzamento delle attività di sviluppo lo consente e deve descrivere, almeno, per ogni specifica (requisito) funzionale e non funzionale da valutare:
   - *schedule* dei test previsti dal documento di specifica dei test,
   - risorse allocate sulle varie attività,
   - rischio rappresentato dal singolo test,

[21] Si noti che le fonti possono essere, oltre che il prodotto software in esame, anche documenti correlati al prodotto o gli stessi utenti del prodotto.

[22] Si intende qui il piano di "accettazione" e/o collaudo del prodotto. Tuttavia, quanto qui definito può valere anche per il piano dei test condotti internamente dal fornitore e propedeutici al test di accettazione/collaudo. Dare al cliente visibilità su questi test, attraverso la presentazione di un apposito piano di test e di specifiche di test, dà maggiore garanzia al cliente sulla affidabilità del fornitore e diminuisce i rischi di arrivare al momento del collaudo con un prodotto non in grado di soddisfare i requisiti.

- responsabilità assegnate nell'effettuare le verifiche,
- procedure di registrazione dei risultati dei test effettuati;

4. effettuazione dei btest previsti dal Piano; in questa fase del processo di valutazione vengono effettuate le prove previste dal documento di specifica del test, secondo quanto definito nel Piano di valutazione, confrontando quindi i risultati con i valori obiettivo attesi; più in dettaglio i passi citati consistono in:
   - la *misurazione,* che rileva le misure selezionate nella fase di pianificazione della valutazione;
   - il rati*ng,* che mappa i valori rilevati su una scala di valutazione, utilizzando i criteri di trasposizione definiti;
   - l'*assessment,* che confronta gli obiettivi che si vogliono raggiungere con lo stato attuale delle misure, determinando l'eventuale *gap* esistente.

15. Appendice 3 – Misure della quantità di riuso in uno sviluppo di software custom

Misura del riuso funzionale tramite punti funzione

Modello architetturale di riferimento

Si richiamano qui di seguito gli schemi architetturali illustrati nei primi capitoli di questo stesso documento, per una discussione della applicabilità della misurazione funzionale tramite punti funzione alla stima dell'impatto che ha sul costo di un progetto il riuso di componenti software pre-esistenti.

Per quantificare la dimensione funzionale di specifiche porzioni software nell'architettura di un sistema software, al fine di fornire metriche relative al riuso di solo determinate componenti rispetto a un intero sistema, occorre utilizzare un metodo di misurazione che disponga di definizioni, procedure e regole per la decomposizione funzionale del software secondo l'architettura di riferimento adottata.

Tra i metodi più diffusi e dichiarati conformi alla serie di standard internazionali ISO/IEC 14143 sulla misurazione della dimensione funzionale del software[23] vi sono in particolare il metodo IFPUG Function Point e il metodo COSMIC (Full) Function Point.

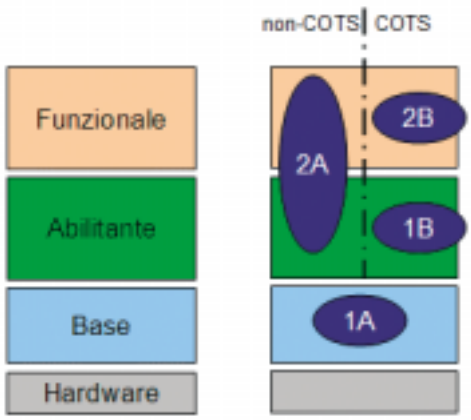


Figura 7. Architettura a livelli (sinistra) ed esempi di componenti distribuiti tra gli strati (destra).

[23] ISO/IEC 14143-1:1998 "Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts" e ISO-IEC 14143-6:2006 "Information Technology – Software measurement – Functional size measurement – Part 6: Guide for use of ISO/IEC 14143 series and related International Standards". I metodi di misurazione accettati come conformi allo standard sono: ISO/IEC 19761:2003 "COSMIC-FFP – A functional size measurement method" (rel. 2.2); ISO/IEC 20926:2003 "IFPUG 4.1 Unadjusted functional size measurement method"; ISO/IEC 20968:2002 "Mk II Function Point Analysis"; ISO/IEC 24570:2005 "NESMA functional size measurement method v. 2.1".

In termini di supporto alla misura del riuso in un ambiente che sviluppa per componenti, il metodo COSMIC, in particolare, fornisce specifiche definizioni e procedure di misurazione relative a componenti software distribuite in architettura multi-strato e/o multi-componente[24]. In tale approccio, il software può essere partizionato in una o più porzioni tali che:

- ogni porzione opera a un differente livello di astrazione funzionale nell'ambiente di esercizio, e
- vi è una relazione gerarchica tra i livelli di astrazione sulla base degli scambi funzionali (input/output di dati) che avvengono tra di loro.

Ognuno di questi livelli è individuato come uno strato ("layer") distinto. Ogni strato incapsula le funzionalità utili allo strato che usa i suoi servizi nella relazione gerarchica e usa le funzionalità fornite dallo strato 'inferiore' in questa relazione. Inoltre, il software di uno dato strato può ulteriormente essere progettato (e quindi dimensionato) come composto da più 'porzioni' distinte. Le comunicazioni tra porzioni distinte di software appartenenti al medesimo strato sono note come comunicazioni 'tra pari livello' (o 'da pari a pari' – 'peer-to-peer').
Dato uno strato qualsiasi, la porzione di software oggetto di misurazione è distinta tramite un confine dal software di qualsiasi altro componente o strato con il quale essa scambia dati. Il confine consente di fare una chiara distinzione tra gli elementi che sono parte della componente software individuata e gli elementi che sono parte dell'ambiente "esterno" (altre componenti, altri strati, altri sistemi – in generale: utenti) in cui tale componente è inserita o può essere riusata. Per esempio, nel caso di un middleware che fornisce servizi di memorizzazione dei dati disaccoppiando il livello di astrazione logico da quello fisico, l'"utente" potrebbe essere l'applicazione che gestisce i servizi anagrafici verso i cittadini (e non direttamente il cittadino, che sarebbe invece utente di quest'ultima applicazione, a un diverso livello di astrazione funzionale, ossia di un diverso strato software).
Utilizzando l'approccio sopra delineato, la misurazione funzionale può essere applicata per determinare la dimensione in punti funzione dei soli componenti riusabili o riutilizzati rispetto alla dimensione complessiva del sistema software che li utilizza o li utilizzerà, consentendo di derivare una metrica di riuso o riusabilità del software. La valutazione del riuso può essere svolta a diversi livelli di granularità, in funzione del momento della misurazione e delle esigenze di accuratezza della stessa, per es. a livello di singolo processo funzionale elementare, di aggregato di più processi funzionale o di intera componente.

[24] Una discussione della misurazione funzionale di software middleware, senza impianto architetturale specifico, è stata proposta in forma di white paper preliminare dall'IFPUG nel 1999, ma non è ad oggi stata ratificata in alcun documento ufficiale. Nel caso del metodo IFPUG, quindi, il partizionamento del software in porzioni ai fini di individuazione e misurazione del riuso o della riusabilità può richiedere linee guida particolari per l'individuazione di ambito della misura e di confini dei sistemi, per esempio per la misura di Web Services o più in generale di SOA (Service Oriented Architectures).

PROCEDURA DI MISURAZIONE ORIENTATA AL RIUSO

Occorre in primo luogo stabilire e documentare chiaramente l'ambito della misurazione (ovvero, quali funzionalità della/e componenti software sono individuate come riusabili e quindi incluse nella misura), la decomposizione funzionale che meglio descrive l'architettura del sistema esaminato e chi sono gli utenti della/e componenti individuate. Si noti, ai fini dell'individuazione di funzioni in strati inferiori a quello applicativo direttamente visibile all'utente finale del software, che ogni componente e strato software, oltre che misurato a sé stante, può costituire anche l'utente funzionale di un altro strato o componente software interfacciata.
Più in dettaglio, in funzione dello scopo della misurazione orientata al riuso, occorre:

- identificare gli strati (e relativi confini);
- identificare le componenti alla pari (e relativi confini) per ogni strato;
- identificare e misurare le funzioni elementari di ciascuna componente che risulti oggetto di possibile riuso e che rientri quindi nell'ambito della misura;
- calcolare le dimensioni delle componenti residue del sistema esaminato (opzionale, ai fini della valutazione di riuso percentuale rispetto al sistema complessivo).

Le definizioni e regole di misurazione delle funzioni elementari (processi e dati) dipendono dallo specifico metodo di misurazione prescelto, e sua versione, che dovrà essere chiaramente documentato. Tali definizioni e regole sono riportate nella manualistica ufficiale di ciascun metodo e non sono esposte in dettaglio in questa sede. Si ricorda sinteticamente che, nel collocare i confini tra due porzioni software interagenti tra loro, a ogni processo elementare di input/output in una porzione del software dovrebbe tipicamente corrispondere un analogo processo di output/input nell'altra porzione.
In accordo con i principi guida della definizione di architetture orientate ai servizi (SOA), che costituiscono come già evidenziato in precedenza un valido esempio di strutturazione del software orientata al riuso, criteri utili per individuare le componenti software separatamente riusabili ai fini anche della loro misurazione possono essere:

- l'"accoppiamento debole" ("loose coupling"): ogni componente mantiene una relazione con altre componenti che ne minimizza necessariamente le interdipendenze;
- la "separazione degli aspetti" (secondo il principio dell'"information hiding"): ogni componente "nasconde" la propria logica elaborative interna alle altre componenti, ma è descritta in termini di quali dati sono necessari per l'accesso in input e quail dati fornisce in output, attraverso opportuna documentazione.

QUANTIFICAZIONE DI RIUSO O RIUSABILITÀ CON METRICHE FUNZIONALI

Per analizzare quantitativamente un software pre-esistente e individuare le opportunità di riuso verso nuovi sistemi, occorre disporre di (o compilare all'occorrenza) una lista possibilmente gerarchica (ossia, a vari livelli di aggregazione) delle funzionalità offerte dal software esistente e delle funzionalità desiderate per l'eventuale riuso nel nuovo sistema custom.

La stesura di tali "liste" con l'individuazione delle funzionalità non comporta un impegno aggiuntivo particolare laddove la misurazione sia comunque richiesta per motivi contrattuali, patrimoniali o a fini di previsione economica del nuovo software custom: essa cioè potrebbe da una parte essere già stata svolta all'epoca della misurazione del sistema di origine, e dall'altra considerata come parte integrante dell'analisi dei requisiti del nuovo software. In ogni caso tale misurazione è fattore abilitante imprescindibile per una adeguata quantificazione della riusabilità, ove realizzabile.
La quantificazione dell'effettiva riusabilità di una porzione software rispetto ai requisiti del nuovo software custom richiesto discende dal confronto di elementi corrispondenti tra le due viste. A un livello di confronto accurato, essa può derivare numericamente dai dettagli della misurazione delle funzioni appartenenti alle componenti esaminate. A seconda del metodo di misurazione adottato, possono presentarsi vari casi, con o senza l'adozione di criteri di individuazione e di pesatura addizionali rispetto alle regole standard di misurazione dei punti funzione.

Caso IFPUG

Nel caso di metrica IFPUG, un indicatore della riusabilità funzionale dei processi elementari è dato dal numero di elementi di tipo dati (DET) uguali e di gruppi di dati referenziati (FTR) analoghi tra "vecchia" e "nuova" situazione, per similitudine. Si noti che l'approccio NESMA, che risulta essere in effetti una variante del metodo IFPUG, già per la manutenzione evolutiva che comporti la modifica di una funzione pre-esistente del medesimo software, suggerisce una scala percentuale di abbattimento della dimensione efficace (ai fini della valorizzazione economica) che può ugualmente corrispondere alla valutazione di riuso funzionale qui indagata, per esempio nel caso di un singolo processo elementare:

modifica percentuale =

	[%DET ≤ 67%]	**[%DET ≤ 100%]**	**[%DET > 100%]**
[%FTR ≤ 33%]	*25%*	*50%*	*75%*
[%FTR ≤ 67%]	*50%*	*75%*	*100%*
[%FTR ≤ 100%]	*75%*	*100%*	*125%*
[%FTR > 100%]	*100%*	*125%*	*150%*

riuso (riusabilità) percentuale = complementare a 1 = *100% - modifica percentuale*

dove: %DET = DET nuovi-modificati-cancellati / numero totale DET originali (x 100)

e: %FTR = FTR nuovi-modificati-cancellati / numero totale FTR originali (x 100)

Un analogo approccio (eventualmente con scala di valori da calibrare opportunamente) può dunque essere adottato per quantificare la similitudine tra requisiti del software pre-esistente e requisiti del "nuovo" software da considerare, ossia per valutare l'effettiva riusabilità del "vecchio" software, componente per componente, in base a elementi metrici di dettaglio.

Caso COSMIC

Nel caso di metrica COSMIC, un analogo indicatore di riusabilità funzionale è dato dal confronto dei "movimenti di dati" (data movement) contati per ciascuna coppia di processi funzionali individuati rispettivamente per il software pre-esistente e per il nuovo insieme di requisiti del software desiderato.

I movimenti di dati di ogni processo funzionale possono essere di quattro tipi: Entry, Exit, Read e Write, in funzione della finalità dei dati movimentati; ogni processo funzionale può contare più movimenti di dati del medesimo tipo, relativi a gruppi di dati distinti tra loro, senza limiti alla scala di valori utile per la misurazione. Il numero di punti funzione da associare ad ogni processo funzionale individuato è funzione diretta della quantità di movimenti di dati contati per tale processo funzionale.

Si noti che l'approccio NESMA sopra descritto per la valutazione di elementi "nuovi" rispetto alla versione pre-esistente dei processi software, sopra evidenziato, è già logicamente incluso nel metodo COSMIC, in quanto la dimensione di una modifica di una funzione pre-esistente è data dai soli movimenti di dati nuovi, modificati o cancellati rispetto al pre-esistente, e non dall'intera dimensione della funzione dopo la sua modifica, come nell'approccio IFPUG.

Non si richiede quindi per il metodo COSMIC un aggiustamento ad hoc particolare rispetto alla procedura standard per esprimere la percentuale di riuso o riusabilità in funzione delle componenti metriche elementari: la valutazione percentuale di copertura dei requisiti funzionali, ossia di riusabilità, tra "vecchio" e "nuovo" software discende in maniera semplice dal confronto delle quantità di movimenti di dati "in comune" piuttosto che "differenti" tra requisiti soddisfatti dal "vecchio" software e requisiti espressi per il "nuovo" software, componente per componente.[25]

VALUTAZIONE COMPLESSIVA DEL RIUSO O RIUSABILITÀ

Un indicatore complessivo del livello di riuso effettivo o riusabilità potenziale di una intera componente tra software pre-esistente e nuovo software desiderato deriva per aggre-

[25] Un ulteriore criterio di similitudine tra funzioni è ovviamente il tipo di elaborazione interna (manipolazione dei dati) che le funzioni compiono sui (medesimi) dati prima di fornire il risultato della propria elaborazione al "proprio" utente o di collocarlo in base dati, come per esempio calcoli, convalide, etc. Nessun metodo di misurazione attuale fornisce una differenziazione numerica specifica per differenti manipolazioni (salvo un peso banalmente maggiorato senza scala numerica particolare nel caso dei processi di tipo External Output nell'approccio IFPUG), ma essa può essere oggetto di ulteriori raffinamenti o linee guida.

gazione della misura di riuso/riusabilità di ogni funzione della componente esaminata, secondo una media pesata della dimensione funzionale di ciascuna funzione. Si noti che i valori di modifica percentuale uguali o superiori a 100% nell'esempio dell'approccio NESMA di cui sopra sono indice di riusabilità scarsa o nulla, conducendo infatti a valori negativi per essa – non dovrebbe in tal caso individuarsi né essere richiesto alcun riuso significativo.

Nel caso occorra fornire una valutazione di massima della riusabilità a livello di intera componente in assenza di dettagli effettivi, per esempio per scopi di stima preliminare o analisi di fattibilità, gli indicatori percentuali suggeriti possono essere ovviamente oggetto di stima e non di misura esatta, a patto di utilizzare la medesima scala percentuale (0-25-50-75-100% o analoga), in funzione della copertura stimata dei requisiti del nuovo software da parte del software pre-esistente.

16. Appendice 4 – Struttura base del Capitolato Tecnico per il riuso

Generalità

Come evidenziato in altre sezioni di questo documento, lo sviluppo di software riusabile richiede che venga posta particolare attenzione, oltre che alle caratteristiche tecniche intrinseche del manufatto che ne devono agevolare la riusabilità, anche alla documentazione prodotta a corredo del software (progettuale, d'uso, di manutenzione e gestione), alla riusabilità dei casi di test predisposti e a quanto altro si ritiene possa facilitare un futuro riuso del componente software sviluppato.
Queste accortezze, peraltro, dovrebbero essere sempre considerate in qualsiasi progetto di sviluppo software, in quanto ciò che facilita la riusabilità di un software è, in larga misura, l'averlo fatto a regola d'arte e averlo documentato in maniera completa e comprensibile.
Un capitolato tecnico per l'acquisto di un servizio di sviluppo di software riusabile, quindi, non dovrebbe differire particolarmente da un normale capitolato tecnico per l'acquisizione di un servizio di sviluppo software, ma in esso vi sono alcuni aspetti peculiari del processo di sviluppo di software riusabile basato su componenti da precisare.
Si ricorda, comunque, che nel caso di sviluppo di software per il web, altre indicazioni vanno prese dalla legge 4/2004.

Principali contenuti del capitolato tecnico

Il capitolato tecnico è il documento che definisce, nel caso degli sviluppi software, ciò che il cliente richiede al fornitore, val a dire i requisiti funzionali e tecnici della fornitura, i requisiti base per le modalità di realizzazione delle attività del fornitore (riguardanti il processo lavorativo) e i criteri e modalità di valutazione di quanto fornito.
Nel capitolato vanno inseriti, tra i requisiti del software, anche quelli di riusabilità, così come le prescrizioni in merito alla documentazione associata al software da produrre, sui test da preparare, effettuare e documentare, sull'utilizzo di un processo di sviluppo di tipo get-put.
In definitiva, il capitolato dovrà includere tutte le prescrizioni atte a garantire la qualità del prodotto e del processo lavorativo utilizzato per realizzarlo in coerenza con quanto raccomandato in queste linee guida. La struttura del capitolato, tuttavia, dovrà ponderare i criteri

teorici nel contesto di riferimento, in funzione delle effettive esigenze del committente, del costo che si è disposti a sostenere, dei vantaggi che si prevede di ottenere (ad esempio in caso di futuri riusi dei componenti che si stanno acquisendo).
La struttura del capitolato tecnico varia a seconda delle caratteristiche della fornitura, ma vi sono alcuni elementi base che devono essere comunque presenti nel capitolato. Nel seguito sono descritti i contenuti da dare a cinque capitoli del documento, la cui stesura è di particolare rilievo per lo sviluppo *ex novo* di software riusabile:

a) termini e definizioni,
b) oggetto della fornitura e suo dimensionamento,
c) modalità di esecuzione della fornitura,
d) modalità valutazione di quanto fornito,
e) Piano della qualità della fornitura.

TERMINI E DEFINIZIONI

Particolare attenzione dovrà essere prestata alla definizione del significato che avranno, nell'ambito della documentazione tecnica e contrattuale, specifici termini utilizzati per la descrizione delle attività oggetto della fornitura e dei prodotti che dette attività realizzano, sui quali sono da definire le condizioni di proprietà e trasferibilità. Si suggerisce di inserire nel capitolato un paragrafo dedicato ad elencare tutte le definizioni applicabili e il loro significato. A questo paragrafo si potrà fare riferimento, nello schema di contratto, per la definizione dei termini riportati negli articoli, ovvero, nel caso di particolare rilevanza della specifica terminologia, potrà essere riportata integralmente la definizione del termine stesso nell'articolo[26].
Nell'ambito di un progetto di sviluppo di software riusabile si ritiene che all'amministrazione acquirente possano essere utili alcune definizioni di uso comune e di immediato utilizzo, di seguito riportate a titolo di esempio. Un esemplificazione non esaustiva, da adattare al contesto contrattuale, che ogni Amministrazione acquirente potrà modulare e verificare in termini di rispondenza a standard e regole interne

a) *Software custom*. Un sistema software sviluppato, secondo vari metodi, mezzi e modalità, singolarmente o in modo congiunto, in dipendenza dagli obiettivi, funzionali o meno, richiesti dall'amministrazione e sulla base di analisi delle specifiche e dei requisiti, con l'utilizzo di linguaggi di programmazione e ambienti di sviluppo. Il software *custom* comprende quindi lo sviluppo di nuovi sistemi applicativi, in lotti o per obiettivi parziali, che risolvono esigenze specifiche a fronte di funzionalità e requisiti richiesti dall'Amministrazione acquirente.
b) *Manutenzione evolutiva, adeguativi e migliorativa di software custom*. Si intende con questo termine l'insieme di interventi volti ad arricchire il prodotto di nuove funzionalità o di altre caratteristiche non funzionali, o comunque a modificare o integrare il

[26] Per la definizione dei termini si deve far riferimento sia a norme standard che alle classi di fornitura elencate nel *Dizionario delle Forniture ICT elementari*, a cura del CNIPA.

prodotto esistente. La manutenzione implica la scrittura di funzioni aggiuntive d'integrazione ad applicazioni esistenti o parti di funzioni (anche in sostituzione di altre già esistenti) di dimensione significativa e di cui è possibile preventivamente definire i requisiti o quantomeno identificare le esigenze. In pratica si tratta di realizzazioni di uno specifico sistema software, anche aggregabili fra loro, che comunque danno luogo a una nuova versione del prodotto iniziale.

c) *Software commerciale (COTS)*. Un'applicazione software commercializzata in modo standard da un'azienda nel campo dell'informatica. Tale applicazione può essere sviluppata ulteriormente presso l'amministrazione acquirente tramite attività di parametrizzazione e personalizzazione, secondo vari metodi, mezzi e modalità, in dipendenza dagli obiettivi, funzionali, o meno, richiesti dall'Amministrazione acquirente.

d) *Parametrizzazione di COTS*. L'uso di file o tabelle standard, accessibili tramite menù codificati o comunque editabili, in cui è possibile definire il funzionamento del sistema senza necessità di sviluppo e conoscenza di codice o linguaggi di programmazione.

e) *Personalizzazione di COTS*. Lo sviluppo di funzionalità non originariamente offerte dall'applicazione, come per esempio nuovi reports di stampa, nuove funzioni, o altro, con cui si vanno a coprire ulteriori aree funzionali richieste dall'Amministrazione acquirente. Per personalizzazione si intendono tutte le attività di sviluppo, che prevedono l'utilizzo di linguaggi di programmazione, assimilabili allo sviluppo di software *custom*.

OGGETTO DELLA FORNITURA

L'oggetto della fornitura ne dettaglia i contenuti, identificando ed elencando i singoli componenti (prodotti e/o servizi). Si suggerisce di inserire nel capitolato una sezione introduttiva in cui elencare i componenti nei quali è stata suddivisa la fornitura e prevedere un paragrafo per ogni componente nel quale specificare:

a) la tipologia di classe di fornitura[27];

b) le attività richieste al fornitore e i relativi prodotti da realizzare;

c) i requisiti funzionali e non funzionali per i componenti da sviluppare, eventualmente rimandando a specifiche appendici al capitolato nelle quali sono descritte nel dettaglio le funzionalità applicative richieste e/o i processi applicabili al componente da sviluppare;

d) i parametri di dimensionamento da prendere in considerazione;

e) l'utenza di riferimento (ad esempio, utenti diretti, utenti indiretti, utenza tecnica del sistema per la gestione applicativa e la manutenzione);

[27] Si faccia riferimento ai lemmi del *Dizionario delle forniture ICT pubblicato dal CNIPA, cfr il materiale sul sito www.cnipa.gov.it*.

f) eventuali vincoli e requisiti organizzativi (ad esempio, scelte tecnologiche pregresse, interfacce con sistemi di strutture esterne).

Qui di seguito si trattano brevemente alcuni degli elementi sopra richiamati.

CLASSI DI FORNITURE

Le forniture che prevedono la consegna di software al termine delle attività svolte dal fornitore si possono differenziare in queste classi:

- "sviluppo ad hoc (o custom)", basato su requisiti espressi dal cliente per soddisfare esigenze specifiche, non soddisfatte da prodotti COTS, già disponibili e acquisibili sul mercato,
- manutenzione evolutiva (MEV), che modifica un software già esistente per adattarlo a nuove esigenze funzionali del cliente,
- manutenzione correttiva di software già esistente, per prodotti la cui garanzia sia scaduta,
- manutenzione adeguativa e migliorativa, che modifica un software già esistente, per adattarlo a nuove esigenze non funzionali del cliente (ad es. nuovo ambiente operativo, o nuove prestazioni tecniche).

OGGETTI SOFTWARE DA CONSEGNARE

Per quanto riguarda gli oggetti software che il fornitore deve consegnare al cliente a fine attività, è necessario qui indicare che la consegna deve comprendere anche:

1) il documento di specifica dei requisiti[28] (SRS) che definisce i requisiti funzionali e non funzionali del cliente, i dati che il software elabora, le interfacce che il software espone verso l'utente, i casi di test funzionali e prestazionali che verranno utilizzati – nel collaudo e nei test di accettazine – per verificare che il software consegnato risponda ai requisiti espressi. È necessario definire qui i contenuti di massima e le modalità sintattiche e semantiche con le quali deve essere prodotto questo documento dal fornitore.

L'elenco di requisiti che devono essere definiti in questo documento è nello standard ISO/IEC 12207 (paragrafo 5.3.4). Tale elenco comprende:

- le funzioni e le prestazioni richieste al software,
- le caratteristiche dell'ambiente tecnologico nel quale il software deve operare,
- le caratteristiche delle interfacce verso l'utente,
- le specifiche di sicurezza e riservatezza dei dati,
- le specifiche ergonomiche riguardanti le modalità di interazione uomo macchina,

[28] Requisito: dallo std IEEE 610.12: *una dichiarazione documentata attestante una condizione od una capacità che un sistema deve possedere per soddisfare una richiesta di un utente, riguardante la risoluzione di un problema, il raggiungimento di un obiettivo…*". Un elenco di requisiti che devono essere definiti in questo documento è nello std ISO/IEC 12207 (paragrafo 5.3.4). Un indice del documento di specifica dei requisiti è definito nello std IEEE

- la identificazione dei dati da trattare e dei loro volumi,
- criteri e modalità di installazione ed accettazione,
- specifiche per la documentazione per l'utilizzo del software da parte degli utenti (il manuale d'uso e di gestione),
- criteri e modalità di gestione e manutenzione del prodotto dopo la consegna.

Un indice del documento di specifica dei requisiti è definito nello stesso standard ISO/IEC 12207 e prevede:

- la descrizione del contesto lavorativo dove opererà il software,
- la descrizione dei processi lavorativi del cliente che vengono impattati dal software da realizzare,
- la individuazione degli utenti del software,
- la lista dei requisiti:
  - Funzionali
  - Non funzionali
  - Vincoli sui dati
- un glossario e/o dizionario dei dati

Tra le modalità sintattiche da utilizzare per descrivere i requisiti, si può prevedere il ricorso a diagrammi di casi d'uso, linguaggio naturale o semiformale, altre notazioni di uso corrente.
È fondamentale che i requisiti espressi dal cliente siano descritti nel documento SRS in modo non ambiguo ed esaustivo. In particolare, tali requisiti devono possedere questi attributi:

- devono essere identificati in modo univoco (ad es. con un codice o un acronimo), correlati alla loro fonte (utente, contesto, tecnologia) e a chi li ha validati;
- deve essere garantita la "tracciabilità" delle eventuali modifiche ad essi apportate (chi le chiede, quando e perché);
- devono essere tra loro collegati per evidenziare se la modifica in uno di essi ha conseguenze su altri;
- devono essere classificati per tipologia (ad esempio, funzionali, non funzionali, opzionali o irrinunciabili, tecnologici etc..);
- a ognuno andrebbe associato il razionale, ovvero perché è stato così definito nella forma corrente (rende più facile modificarlo in seguito, se necessario).

Nel caso più generale, un requisito è descritto nel documento di specifica SRS da questi elementi:

- Identificativo (acronimo o numero progressivo)
- Tipologia di requisito (funzionale, non funzionale, inverso)
- Descrizione (sintetica) del requisito

- Collegamento del requisito ad eventuali diagrammi esplicativi (ad es. casi d'uso e diagrammi di sequenza o collaborazione)
- Se requisito funzionale, dati in ingresso ed uscita alla funzione (identificazione dei dati, loro attributi, fonte degli input e archivio di destinazione per gli output, volumi in gioco)
- Se requisito funzionale, identificazione degli utenti
- Criteri per verificare il soddisfacimento del requisito (casi di test)
- Fonte del requisito (chi lo ha espresso e quando)
- Razionale del requisito (perché è stato richiesto)
- Storia delle modifiche apportate nel tempo al requisito

2) un documento di disegno tecnico del software, che deve illustrare l'architettura logico funzionale del sistema, i comportamenti statici e dinamici delle componenti del sistema software, le strutture dei dati, gli algoritmi che permettono le elaborazioni dei dati, le interfacce verso l'utente.
È necessario dare definire qui i contenuti di massima e le modalità sintattiche e semantiche con le quali deve essere prodotto questo documento. Tra le notazioni utilizzabili sono diffusi i diagrammi UML:

- delle classi, per rappresentare il comportamento statico del software,
- di interazione, per rappresentare come i componenti del sistema software si scambiano messaggi,
- delle attività, per rappresentare il comportamento dinamico) e i diagrammi E/R per definire il modello concettuale dei dati elaborati dal software.

3) la documentazione d'uso, ovvero il manuale utente e quelli di gestione e manutenzione del software, di cui è opportuno dare una bozza di indice o, in alternativa, chiedere al fornitore di produrla nelle prime fasi del lavoro in modo che il cliente la possa approvare o chiederne una modifica in tempi utili. È utile indicare qui che il 100% delle funzioni a disposizione dell'utente deve essere descritto nel manuale d'uso.
4) il piano dei test che verranno effettuati sul software. Una trattazione del processo di testing del software e del piano dei test è nello standard ISO/IEC 14598. Un possibile indice del piano deve prevedere:

- le finalità della valutazione (obiettivi delle verifiche che verranno effettuate, ad esempio per validare semilavorati prodotti da una fase di lavoro, verifica di accettazione, di collaudo, di non regressione, di usabilità, di sussistenza delle caratteristiche originali, di post rilascio in esercizio etc..)
- gli elementi da valutare, cioè i componenti del software da sottoporre a test
- le caratteristiche indagate e obiettivi quantitativi/qualitativi da misurare
- la pianificazione temporale *(schedule)* delle verifiche e risorse allocate sulle varie attività
- la responsabilità nell'effettuare le verifiche

- le misure da rilevare e le metriche da utilizzare
- i criteri di elaborazione delle misure rilevate
- le procedure di registrazione dei risultati
- l'ambiente hardware e software da usare per il test
- i casi di prova utilizzati, in riferimento a quanto previsto dal documento di specifica dei requisiti.

È utile indicare qui la percentuale di funzioni e di requisiti non funzionali che verranno sottoposti a verifica (fino al 100%) e quante verifiche vanno previste per ognuno dei requisiti, ovvero il criterio di fine dei test per ognuno dei requisiti dati.
5) La configurazione del sistema software fornito. È utile chiedere che tale configurazione sia gestita attraverso un prodotto ad hoc, che faciliti le modifiche e la tracciabilità delle versioni del software.

DIMENSIONAMENTO DELLA FORNITURA

Si definisce "dimensione della fornitura" la quantità di tutte le componenti che devono essere consegnate al cliente.
La quantità deve essere definita non solo per il software in senso stretto (da definire in termini di punti funzione o linee di codice), ma anche per i documenti da produrre (in termini di requisiti e/o funzioni descritti), per le attività di supporto allo sviluppo (in giorni persona), per i test da effettuare (in numero di test per punto funzione o per requisito, o metriche equivalenti).
Le quantità possono essere definite anche come massimali non superabili (ad esempio nel caso dei punti funzione e delle linee di codice).
Se il progetto si avvale del riuso di software già esistente per produrre parte del nuovo software, va dimensionata l'incidenza del riuso – tipicamente espressa come percentuale – nel determinare la quantità di software effettivamente realizzato nella fornitura e di conseguenza quanto "nuovo" sviluppo andrà pagato al fornitore.
Il contratto, tuttavia, può prevedere un corrispettivo da versare al fornitore per le attività di adattamento o integrazione di software già esistente, da riusare nell'ambito della nuova fornitura.

SERVIZI PER IL RIUSO

Tra i servizi previsti all'interno di una fornitura di software riusabile, possono essere compresi anche quelli finalizzati proprio a favorire il futuro riuso degli oggetti in altri contesti. L'applicabilità di questi servizi ai casi concreti va valutata in modo specifico. Si riporta di seguito, a titolo esemplificativo e non esaustivo, un elenco di questi servizi:

- manutenzione evolutiva, per adattare un'applicazione a nuove esigenze mediante l'aggiunta di nuove funzionalità;
- servizi di personalizzazione, ad esempio per adattare l'applicazione a un ambito diverso da quello per cui è stata inizialmente sviluppata;

- manutenzione correttiva, diagnosi e rimozione delle cause e degli effetti dei malfunzionamenti;
- servizi di bonifica e normalizzazione delle banche dati dell'applicazione;
- affiancamento per la presa in carico; è questo un servizio che impegna il fornitore a dare al cliente tutte le indicazioni e le conoscenze necessarie a prendere in carico l'applicazione, mettendo a disposizione proprio personale con qualifica e per il tempo stabiliti a contratto e ritenuti adeguati ad assicurare il trasferimento integrale delle conoscenze;
- servizi erogati in modalità ASP: il fornitore installa l'applicazione, di proprietà dell'Amministrazione, su infrastrutture ospitate in propri centri servizi, è responsabile della gestione e manutenzione dei sistemi e delle applicazioni, offre i servizi applicativi alle altre amministrazioni indicate dall'Amministrazione proprietaria.

Per questi adempimenti, nel contratto vanno previste le condizioni di erogazione e la loro durata nel tempo che il fornitore dovrà rispettare nei confronti delle amministrazioni indicate dall'acquirente.

MODALITÀ DI ESECUZIONE DELLA FORNITURA

In questa sezione del Capitolato vanno definite le modalità con le quali devono essere eseguite dal fornitore le attività e gli eventuali vincoli posti a tali attività. Potrebbe essere utile scomporre ogni componente della fornitura in sotto prodotti o sotto-attività (costruendo una WBS de progetto), in maniera tale da definire, laddove ragionevole, modalità specifiche. Vanno indicati, tra l'altro:

- gli standard di riferimento per il processo lavorativo (quali gli standard ISO 9001 e ISO 90003, il Capability Maturity Model - CMMI, lo std ISO/IEC 15504, ITIL etc…) e riferimenti metodologici per lo svolgimento delle attività,
- il ciclo di sviluppo da adottare (iterativo e incrementale, a spirale, RUP, XP etc..),
- le modalità di gestione dei requisiti (come si raccolgono e come si modificano i requisiti) e di progettazione tecnica del software,
- le modalità di ricorso al riuso di software già esistente per produrre nuovi componenti software (ricorso a un catalogo di componenti software riusabili, modalità di prelievo dal catalogo e di valutazione della utilizzabilità di quanto prelevato etc..),
- il contenuto del piano di progetto che il fornitore deve predisporre (un esempio è nello std ISO 10006) e i vincoli temporali fissati dal cliente sulla data di fine di attività e/o sotto attività con consegna di semilavorati,
- i vincoli normativi e/o regolamentari da osservare nello svolgimento delle attività (ad esempio regole interne al cliente che il fornitore deve rispettare),
- le caratteristiche delle risorse professionali utilizzate dal fornitore e criteri di eventuale *turn over* delle risorse del fornitore (previa approvazione da parte del cliente).

CRITERI DI VALUTAZIONE DI QUANTO FORNITO

In questa sezione vanno definite le modalità con le quali il cliente effettuerà le verifiche di accettazione e collaudo di quanto fornito. Devono essere qui indicate:

- le scadenze per le verifiche,
- gli oggetti da sottoporre a verifica,
- i riferimenti per la verifica (standard, documenti progettuali e di requisiti etc..),
- gli ambienti operativi che dovranno essere approntati dal fornitore per consentire le verifiche,
- i criteri di accettazione di quanto fornito,
- l'eventuale contraddittorio in caso di esito non positivo della verifica (ad es. la scadenza per l'effettuazione di una nuova verifica, conseguenze di esito ancora negativo etc..),

Per garantire all'amministrazione acquirente gli strumenti contrattuali per gestire i rapporti con il fornitore, è necessario formalizzare nel capitolato e nello schema di contratto le regole alle quali le parti dovranno attenersi, definendo almeno i seguenti elementi:

- i ruoli e le strutture funzionali dell'amministrazione coinvolte nel collaudo (ad esempio, capo progetto, commissione di collaudo);
- le date rilevanti (milestone) nell'esecuzione della fornitura (ad esempio, data di accettazione, data di attivazione in esercizio);
- le modalità di tracciamento e formalizzazione delle attività e relativi esiti.

Devono essere specificate le cadenza temporali entro le quali l'amministrazione procederà all'approvazione o alla formalizzazione di eventuali rilievi, a seguito della consegna dei prodotti. Il contratto potrà stabilire espressamente per il fornitore un tempo limite entro il quale correggere le anomalie rilevate, oppure riservare al committente la facoltà di assegnare in corso d'opera il termine limite entro il quale i vizi e le difformità dovranno essere corretti. Questa facoltà, dovrà essere esercitata entro un lasso di tempo variabile, a discrezione dell'amministrazione, dall'esito negativo del collaudo e in difetto di risoluzione dei problemi da parte del fornitore.

CONSIDERAZIONI SULLE PENALI APPLICABILI

Nello sviluppo di software riusabile, la conformità di quanto realizzato rispetto ai requisiti generali e specifici per il riuso indicati nel capitolato è talmente rilevante da consentire al committente di prevedere in contratto un meccanismo di penali monetarie proporzionato al mancato rispetto dei vincoli temporali fissati per la consegna e per i ricicli in fase di test, nonché per gli scostamenti per difetto dai valori minimi specificati per gli indicatori di qualità.

L'importo della penale applicabile può essere un valore percentuale del corrispettivo richiesto per la realizzazione dell'oggetto software oppure essere quantificato in valore assoluto.

La penale può anche essere prevista per importi che aumentano progressivamente al crescere del numero di indicatori o termini violati[29] ed è comunque un efficace deterrente che impone al fornitore il rispetto dei requisiti richiesti e offerti per la fornitura, garantendo all'Amministrazione un'equo indennizzo per le carenze rilevate nell'oggetto acquisito.
L'incidenza delle penali, comunque, deve essere commisurata al valore complessivo del contratto di fornitura e deve essere prevista anche una contenuta "tollerabilità" dell'inadempimento, soprattutto quando esso sia ascrivibile a responsabilità di entrambi i contraenti o a cause di forza maggiore.
In particolare, il mancato rispetto dei limiti fissati per gli indicatori (fuori soglia) determina azioni contrattuali conseguenti che possono consistere in:

- ripetizione da parte del fornitore dell'erogazione di una prestazione, rifacimento di un'attività, riconsegna di un prodotto (chiusura di una non conformità);
- richiesta al fornitore di effettuare azioni correttive sui processi produttivi per evitare il ripetersi di sistematiche non conformità;
- applicazione di penali monetarie progressive, con un limite massimo, nel senso sopra descritto;
- azioni aggiuntive, come richiesta danni o risoluzione anticipata del contratto, anche con il ricorso ad azioni legali.

Le penali sono una quantificazione ex ante del danno che subirà l'utilizzatore a fronte di un determinato inadempimento e sono stabilite in relazione all'analisi del disservizio presunto. In molti casi, nei lemmi del dizionario delle forniture ICT, il valore economico oggetto di penale viene indicato attraverso un intervallo di valori tipici. In alcuni casi, ove non sia possibile precisarlo neanche in misura approssimata, l'importo viene concordato fra le parti in sede di definizione del contratto.

[29] Per evitare una crescita incontrollata della penale si ritiene opportuno fissare un limite superiore oltre il quale il contratto può essere risolto di diritto.