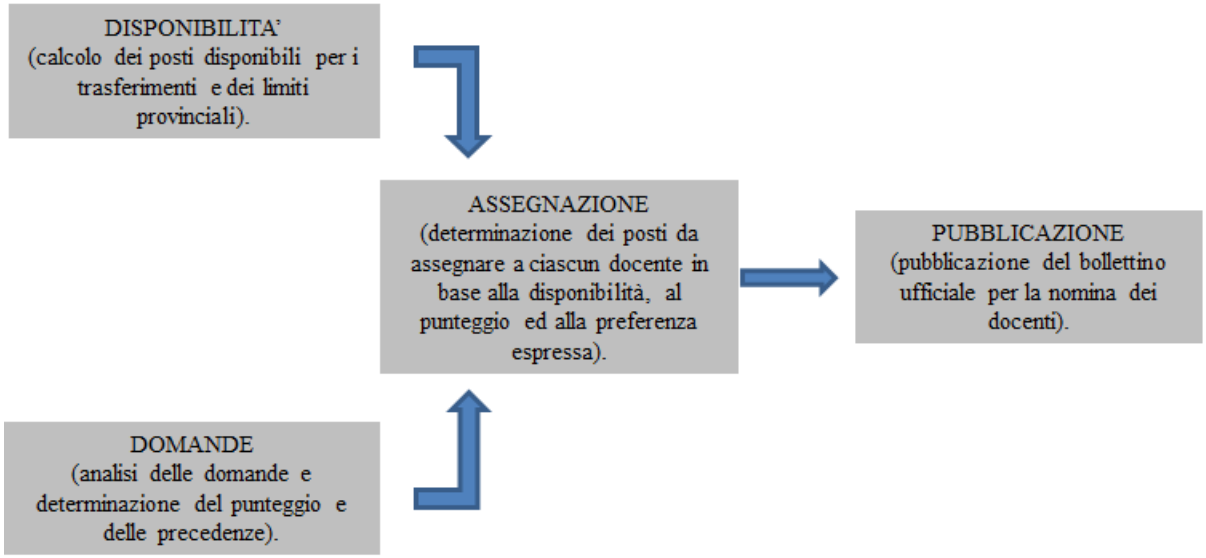
PERIZIA TECNICA PRELIMINARE SULL'ANALISI DELL'ALGORITMO CHE GESTISCE IL SOFTWARE DELLA MOBILITÀ DOCENTI PER L'A.S. 2016/2017.

Su richiesta dell'associazione "Gilda degli Insegnanti" e dell'Avv. Michele Bonetti si è preso in esame l'algoritmo che gestisce il software della mobilità dei docenti per l'a.s. 2016/2017 al fine di esprimere un parere tecnico preliminare.
A seguito di una prima analisi della struttura e delle funzioni svolte da detto algoritmo, è stato possibile effettuare le seguenti considerazioni tecniche.

INTRODUZIONE

Al fine di inquadrare la questione di cui ci si occupa e considerando la complessità della vicenda, appare imprescindibile riportare brevemente la descrizione delle operazioni di mobilità per l'anno scolastico 2016/2017.
Come noto le operazioni di mobilità si distinguono in due macrofasi. La prima relativa agli spostamenti degli insegnanti nominati fino al 2014/2015 all'interno della provincia di titolarità (fase A), e la seconda relativa agli spostamenti degli insegnanti tra province diverse, e suddivisa a sua volta in tre ulteriori fasi (fase B, C, e D).
La funzione dell'algoritmo è quella di gestire tali operazioni di mobilità degli insegnanti sulla base dei criteri stabiliti nel CCNI del 8 aprile 2016 in attuazione di quanto previsto nella legge n. 107 del 2015.
Nella specie, l'algoritmo avrebbe dovuto eseguire l'analisi dei dati di input (disponibilità e domande), determinare l'assegnazione di ambiti e di scuole a ogni insegnante e, infine, diffondere le risultanze sulla base del seguente schema logico:



Il sistema in questione, fornito dalla società HPE s.r.l., mandataria del RTI tra HPE srl e Finmeccanica Spa, si compone di cinque file differenti, quattro inerenti alla prima fase (detta fase A) relativa ai quattro gradi di istruzione (I GRADO, II GRADO, INFANZIA, PRIMARIA) ed un quinto (e unico) file relativo alle successive fasi (dette B, C e D).

Da un'analisi preliminare dei file si evince quanto segue.

Il codice sorgente in questione, relativo alla sola fase A, è sviluppato nel linguaggio di programmazione denominato COBOL (**CO**mmon **B**usiness-**O**riented **L**anguage). Si tratta di un linguaggio ideato agli inizi degli anni '60 ed utilizzato principalmente per lo sviluppo di software applicativi di tipo commerciale come, ad esempio, software bancari e di contabilità, caratterizzati dall'utilizzo di sistemi mainframe interni.

Ogni file relativo ad uno dei gradi di istruzione presenta mediamente circa 55 righe di codice per ogni pagina, nel formato PDF in cui è stato fornito l'algoritmo. Considerando il numero delle pagine per ognuno dei file forniti, si può stimare il totale di righe di codice che compongono l'intero programma in un numero pari a circa 29600 righe.

La scelta di realizzare l'algoritmo di gestione del software della mobilità in linguaggio COBOL appare singolare in quanto oramai utilizzato solo da società

il cui sistema di gestione dati ha avuto origine in tale forma e, per l'onerosità del passaggio ad altro sistema di scrittura nonché per i rischi di perdita di efficacia del sistema e corruzione dei dati durante la migrazione, scelgono di continuare ad utilizzare un linguaggio vetusto e poco performante.

Non è, però, il caso della stesura dell'algoritmo per la gestione della procedura di mobilità in questione, il quale, si presume sia stato stilato ex novo e si sarebbe dovuto comporre di operazioni semplici e lineari.

Quanto alle fasi B, C e D, invece, si evidenzia che il relativo codice è stato sviluppato in maniera differente, in linguaggio "C", e composto da 61 pagine totali all'interno delle quali si riscontrano particolari singolarità di cui si dirà di seguito. Si tratta di uno dei principali linguaggi di programmazione definito come "linguaggio ad alto livello", utilizzato per lo sviluppo di software di sistema per molte piattaforme hardware moderne. A differenza del COBOL, programmi realizzati in "C" hanno una dimensione ridotta e sono più efficienti grazie al linguaggio strutturato del codice stesso, anche se ingloba un metodo scadente per l'identificazione di errori.

A prima vista si nota che la stesura degli algoritmi relativi alla fase A e alle fasi B, C e D è stata realizzata non solo in linguaggi diversi, ma presumibilmente da soggetti diversi che presentano una differente cifra stilistica, nonché semantica letterale nella programmazione di software. Quest'ultima risulta essere una pratica spesso utilizzata da parte delle case produttrici di software, ma comunque delicata laddove non viene definito un unico standard di riferimento per la struttura del codice da sviluppare; standard al quale tutti gli sviluppatori devono riportarsi. Soprattutto se l'obiettivo è quello di sviluppare più sistemi che devono tra loro comunicare e condividere dati e risultati (come ad esempio la determinazione dei posti vacanti e disponibili in ciascun ambito rimasti a seguito delle operazioni di cui alla fase A i cui "dati di output" devono diventare "dati input" nelle successive fasi). Nel caso di specie tale standard non viene osservato all'interno dell'intero listato esaminato.

Allo stato non è possibile fare una compiuta analisi circa il formato dei dati di input e di output delle varie fasi in quanto tali informazioni non sono state fornite dalla società e dal ministero, come è avvenuto anche per altri elementi all'interno del codice, di cui si dirà nel prosieguo.

ANALISI FASE A

a. Quanto ai file relativi alla detta fase A, che come già riferito si distinguono in quattro diversi blocchi redatti interamente in COBOL, si osserva quanto segue. Un programma sviluppato in COBOL presenta, tipicamente, quattro divisioni esplicate qui di seguito:

1. IDENTIFICATION DIVISION
2. ENVIRONMENT DIVISION
3. DATA DIVISION
4. PROCEDURE DIVISION

La divisione IDENTIFICATION DIVISION identifica il programma, ossia contiene informazioni generali, come il nome del programma stesso, la data di edizione, la data di compilazione, il nome dell'elaboratore per il quale è stato scritto e altre annotazioni.

La divisione ENVIRONMENT DIVISION specifica le apparecchiature usate e i file che servono al programma.

La divisione DATA DIVISION contiene la descrizione dei file e dei dati creati o utilizzati, assieme a tutte le altre variabili e costanti che servono al programma.

La divisione PROCEDURE DIVISION specifica il procedimento elaborativo da applicare ai dati, ossia la parte logico-aritmetica.

Da un'analisi preliminare del codice si evince che ogni file è composto da più di una macrostruttura come quella appena descritta, facendo intuire la presenza di diverse funzioni ("PROGRAM-ID") utilizzate dal programma all'interno di uno stesso grado di istruzione.

b. Ciò che appare evidente da una rapida analisi è la struttura alquanto articolata del programma stesso, che di primo acchitto intravedere la presenza di istruzioni ridondanti e non efficienti per la realizzazione di semplici operazioni di gestione dei dati (e di calcolo logico-aritmetico). La suddetta caratteristica viene tipicamente analizzata tramite appositi "tool" che studiano l'indice di ridondanza di un codice sorgente. È difatti noto, nell'ambiente informatico, come il COBOL presenti una sintassi molto lunga, dispersiva e poco flessibile. Basti osservare l'elevata quantità di variabili di appoggio dichiarate nella sezione DATA DIVISION.

Una ulteriore complessità si osserva nella scelta dei nomi per la definizione delle variabili e degli elementi usati all'inizio di ogni funzione (DATA DIVISION) nella computazione logico-aritmetica dei dati, sicuramente non orientati ad un facile utilizzo ed ad una facile comprensione anche da parte dello sviluppatore stesso. Di seguito si riporta un esempio in cui si evidenzia la presenza di cosiddetti "paragrafi" (porzioni di codice con il compito di eseguire particolari operazioni) denominati in maniera farraginosa e non orientata ad un utilizzo agevole del codice.

```
001316*
001317 ML-180.
001318     IF W-IND-PRV LESS SMRESTA-NUM-REC AND W-ABEND EQUAL ZERO
001319       GO TO ML-050
001320 ML-190.
001321     MOVE W-CLC-CORRENTE TO W-CLC-PRECEDENTE.
001322*
```

Difatti, nell'ambito dello sviluppo software, tutti i programmatori tendono a rendere il codice sviluppato di facile intuizione, utilizzando costrutti logici più semplici possibile e nomi per variabili il più possibile di rapida comprensione ed autoesplicativi, in modo da avere un pieno controllo dell'intero codice, anche e soprattutto in caso di modifiche successive dello stesso. In questo caso, all'interno del codice fornito, si osserva l'uso di nomi articolati o abbreviati, ed ai quali non

viene associato alcun tipo di commento o indicazione, tutto ciò sintomo di una possibile elaborazione successiva delle variabili, che, in alcuni casi viene anche utilizzata per rendere il codice di difficile cognizione da parte di soggetti terzi. Segue il frammento di codice già riportato sopra, in cui vengono evidenziati i nomi degli elementi in parola.

```
001316*
001317 ML-180.
001318     IF W-IND-PRV LESS SMRESTA-NUM-REC AND W-ABEND EQUAL ZERO
001319       GO TO ML-050.
001320 ML-190.
001321     MOVE W-CLC-CORRENTE TO W-CLC-PRECEDENTE.
001322*
```

c. Quanto osservato finora porta quindi a dedurre che il codice possa non essere stato sviluppato direttamente nella forma in cui appare, ma che invece possa essere il prodotto di una traduzione o di generazione automatica da un linguaggio di programmazione di più alto livello logico e di più rapida intuizione per l'utente e per lo sviluppatore. Difatti si suppone, in questo caso, l'utilizzo di particolari ambienti per lo sviluppo di software, di più facile utilizzo e tipicamente adoperati in quest'ambito, come ad esempio specifici framework.

Nell'ambiente informatico, per "framework" si intende un'applicazione che permette il più celere sviluppo di software. I componenti di cui fa uso sono, infatti, creati per essere sfruttati dalle altre applicazioni per adempiere ad una serie di operazioni che il programmatore non deve così preoccuparsi di implementare. In generale un framework può includere software di supporto, librerie e, a seconda dei casi, anche un linguaggio di scripting o un IDE (Integrated Development Environment) con funzioni di edit, compile e debug, un ambiente di sviluppo, o altri strumenti utili alla realizzazione finale del codice sorgente di un programma. In tal caso, è molto probabile che l'azienda abbia sviluppato l'algoritmo utilizzando un sistema proprietario già esistente e collaudato per altre operazioni e sul quale è stata aggiunta semplicemente la parte computazionale

vera e propria, generando così gran parte del codice COBOL relativo all'architettura del software dell'algoritmo in maniera automatica, aumentando la probabilità di avere al suo interno una buona parte di codice ridondante, ma con il minimo sforzo da parte della ditta appaltatrice ed a discapito della leggibilità del codice stesso.

d. A quanto esposto si aggiunge anche l'assenza di porzioni di codice, in più punti, nel listato fornito, come si evince dallo stralcio che segue.

```
001450    IF W-TAB-COR1-IND > 150 MOVE 1 TO W-ABEND
001451      DISPLAY 'SFONDATA TABELLA SMFORZA W-TAB-COR1'
001452      GO TO A0-100.
001453*
001454    IF W-INIZ NOT EQUAL 1 GO TO A0-020.
001455*
001458 A0-040.
001459*
001460    MOVE 0 TO W-INIZ.
001461    READ SMFORZB INTO SMMFORB AT END MOVE 1 TO W-INIZ.
001462*
001463    IF W-INIZ EQUAL 1
001464      DISPLAY
```

Si sottolinea che è possibile riscontrare questa operazione in COBOL, la quale non è vietata e non risulta essere un'anomalia se, ad esempio, collegata ad un "jump" nel programma, ma che assolutamente deve essere evidenziata, rendendo noto il salto eseguito. In questo caso nulla viene detto in merito alla numerazione non continua presente nel codice in svariati punti. Senza specificazioni in merito potrebbe trattarsi addirittura di un'operazione di oscuramento attuata dagli sviluppatori per motivi ignoti.

e. Si osserva inoltre che i file forniti non permettono il trasferimento delle righe di codice in formato testuale su un qualsiasi altro editor di testo. L'impossibilità di copiare il codice quindi non permette di eseguire un'analisi diretta del programma stesso, o di porzioni di esso, per un riscontro oggettivo attraverso software e compilatori specifici utilizzati per il debug di programmi in questa forma o anche semplicemente per eseguire l'algoritmo stesso.

f. È doveroso, poi, sottolineare una ulteriore possibile criticità del software nella fase di "data entry", ossia nella parte di immissione dei dati in ingresso su cui il codice esegue la parte computazionale. Difatti nelle divisioni "DATA DIVISION" di ogni grado di istruzione vengono "chiamati" i file dei dati su cui operare e riguardo i quali non viene fornita alcuna informazione nella documentazione rilasciata dagli sviluppatori. Nello specifico, è importante conoscere la struttura e la formattazione di tali dati presenti nei file caricati e "chiamati" all'inizio di ogni funzione ed in ogni parte del programma per poterne attestare il corretto uso all'interno del codice stesso. Possibili anomalie sui dati in ingresso possono ovviamente produrre risultati alterati o non attesi; stessa cosa dicasi anche dei dati in uscita.

Di seguito si riporta un estratto in cui si osserva la "dichiarazione" dei file usati dal programma.

```
000324 FD SMALIA BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000325 01  SMALIA1          PIC X(180).
000326     SKIP2
000327     SKIP2
000328 FD SMALIT BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000329 01  SMALIT1          PIC X(180).
000330     SKIP2
000331 FD SMFDOA BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000332 01  SMFDOA1          PIC X(6).
000333     SKIP2
000334 FD SMFORZA BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000335 01  SMFORZA1         PIC X(40).
000336     SKIP2
000337 FD SMFORZB BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000338 01  SMFORZB1         PIC X(40).
000339     SKIP2
000340 FD SMFORZC BLOCK CONTAINS 0 RECORDS LABEL RECORD IS STANDARD.
000341 01  SMFORZC1         PIC X(40).
```

Alla luce di quanto riferito risulta determinante conoscere la struttura di questi dati e file utilizzati. Ovviamente si parla di conoscere esattamente la struttura e la composizione dei dati per un'analisi corretta delle operazioni eseguite dal codice, e non di contenuto formale dei file, del tutto ininfluenti ai fini della verifica.

Riguardo quest'ultimo aspetto, nessuna informazione viene ceduta dalla società appaltatrice o dal ministero.

g. Inoltre, tipicamente nelle sezioni di ENVIRONMENT DIVISION si definisce il tipo di macchina sulla quale il programma viene eseguito. Nel codice relativo alla fase A di ogni grado di istruzione si nota che il programma è stato eseguito su un'architettura tipo IBM 370[1], ossia un mainframe IBM il cui sistema è stato progettato agli inizi degli anni '70. Segue lo screen del codice relativo.

```
000273*****************************************************************
000274*      *** ENVIRONMENT DIVISION ***                 *
000275*****************************************************************
000276    SKIP2
000277 CONFIGURATION SECTION.
000278    SKIP2
000280    SKIP1
000281*OBJECT-COMPUTER. IBM-370.
000282    SKIP2
000283 INPUT-OUTPUT SECTION.
000284    SKIP1
000285 FILE-CONTROL.
000286    SKIP2
000287    SELECT SMDISP ASSIGN TO EXTERNAL SMDISA
```

Ciò a conferma di quanto esposto al punto c.

ANALISI FASE B, C e D

a. Le fasi B, C e D della mobilità sono racchiuse in un unico file di 61 pagine in cui è presente il codice sorgente realizzato in linguaggio "C". L'intero programma si compone di diverse porzioni di codice disgiunte, ognuna con un particolare scopo e presumibilmente realizzati e compilati su più file. Difatti questa pratica risulta essere abbastanza comune nella programmazione in linguaggio C, in cui tutti i file compilati (porzioni del codice) vengono inclusi e richiamati nel corpo del programma. Si osservano quindi numerose funzioni c.d.

[1] https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3145.html

VOID realizzate dal programmatore per eseguire specifiche operazioni, più volte richiamate nel listato. Ma ovviamente ciò che appare evidente è la differenza nella quantità di righe di codice utilizzate per le fasi B, C e D rispetto a quelle usate per la fase A, fermo restando la differenza dei due linguaggi, ma che comunque hanno lo scopo di eseguire lo stesso tipo di operazioni computazionali.

b. Il codice, in tutte le sue parti, fa spesso uso di librerie richiamate all'inizio di ogni sezione. Nell'ambito della progettazione software, per librerie si intende un insieme di file che hanno al loro interno delle particolari funzioni utilizzate dai programmatori nella fase di stesura del codice e necessarie per l'elaborazione. Tipicamente nello sviluppo di software di questo tipo si includono delle librerie standard che contengono funzioni comuni, spesso già presenti di default nell'ambiente di sviluppo, oppure si possono includere librerie (e quindi funzioni) modificate o create ex novo dal programmatore stesso. Nel caso in esame, spesso si utilizzano particolari librerie ("header") denominate come "db.h" e "utl.h", necessariamente definite ed esplicate in tutte le loro parti all'interno del codice e che presentano funzioni operanti su particolari file di dati, quali database.

È di estrema importanza, al fine di una corretta analisi del codice, possedere tutte le informazioni riguardo la struttura ed il formato dei database e dei file su cui le funzioni ed il programma operano. Risulta quindi necessario conoscere ad esempio tabelle, campi e vincoli di integrità referenziale dei database usati. Tipicamente, nel caso di un'analisi formale e dettagliata del codice, queste informazioni devono essere e vengono fornite contestualmente a tutta la documentazione ed al codice sorgente stesso, ma mancanti nel caso di specie.

c. Inoltre, anche per il caso delle fasi B, C e D si osserva la stessa inefficienza nell'uso dei costrutti logici, delle istruzioni condizionali e nell'uso dei nomi di difficile intuizione per variabili e funzioni, assolutamente controproducente già nella fase di sviluppo stessa del codice.

A ciò si aggiunge la presenza di sporadici commenti lungo il listato che lasciano intendere una dubbia certezza da parte del programmatore nelle operazioni da

compiere, come si evince chiaramente dall'esempio che si riporta di seguito, dove con il termine "AA" si indica la classe di concorso "scuola primaria", così definita poi in una sezione separata del codice.

```
else if (strncmp("DH",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
else if (strncmp("EH",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
else if (strncmp("EN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
else if (strncmp("HN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
else if (strncmp("IN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
/* Questi 2 dovrebbero valere solo per AA */
else if (strncmp("CN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}
else if (strncmp("DN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'H'; dst.dstid[30] = 'H';}

else if (strncmp("AN",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'A'; dst.dstid[30] = 'N';}
else if (strncmp("IL",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'A'; dst.dstid[30] = 'N';}
else if (strncmp("ZJ",db->campi[DST_COD_TPP],2)==0) { dst.dstid[29] = 'A'; dst.dstid[30] = 'N';}

  if (!isupper(cp[0])) return 0;
  if (!isupper(cp[1])) return 0;
  for (k=2; k<16; k++)
    if (!isdigit(cp[k])) return 0;
  return 1;
}

/* Restituisce l'ordined della scuola AA=Infanzia EE=Primaria MM=I Grado SS=II Grado */
ordine_t creadom(db_t *db, char*fname)
{

  FILE *domfile;
  FILE *infile;
  FILE *ndxfile;
  FILE *pntfile;
```

A titolo di esempio, si osserva sempre nello stesso frammento di codice sopra riportato, l'uso inappropriato ed inefficiente del costrutto "elsif". Nella programmazione software è buona norma evitare di annidare più istruzioni condizionali "elsif" in cascata, come nel modo riportato, poiché può portare ad un appesantimento del codice stesso, in fase di compilazione, denotando quindi una chiara forma di inefficienza nella progettazione software.

d. In ultimo, anche nel caso in esame, riguardo le fasi B, C e D, è presente un blocco nel file che non consente di trasferire il listato per intero o in porzioni di esso direttamente su un compilatore opportuno al fine di testare tutte le funzioni

esistenti tramite appositi tool di debug, atti alla verifica della sua corretta funzionalità.

CONSIDERAZIONI FINALI

In conclusione, seppur trattasi di un'analisi preliminare in cui non è stato eseguito un vero e proprio studio completo del codice, tale da rilevare l'effettiva congruenza dell'algoritmo si osserva che lo stesso appare complessivamente uno strumento poco confacente ai fini operativi da raggiungere, e questo per diversi ordini di ragioni.

- In primis, nell'ambito della programmazione di software, è buona norma impostare la struttura del codice che si sviluppa in maniera quanto più possibile semplice. Difatti, tipicamente, un buono stile di programmazione prevede la stesura del codice sorgente osservando un trade-off tra la leggibilità e manutenibilità del codice, e l'efficienza computazionale in termini di tempo di esecuzione e consumo di risorse. È quindi caldamente consigliabile eliminare o evitare il più possibile codice ridondante e codice morto. Nel caso di specie salta subito all'occhio che non sono stati osservati i più basilari criteri di programmazione che notoriamente si applicano. Difatti anche alla luce della semplicità dell'operazione richiesta, non si comprende quali siano le ragioni che hanno indotto il programmatore a creare un sistema ampolloso, ridondante e non orientato alla manutenibilità, specie come nel caso della fase A dell'algoritmo. Ciò anche in considerazione del fatto che è statisticamente provato che un software che deve eseguire operazioni elementari dal punto di vista logico, se consta in un gran numero di righe di codice, ha più probabilità di presentare errori e malfunzionamenti al suo interno. L'aver articolato in tale maniera un algoritmo che doveva svolgere funzioni relativamente semplici è anche sinonimo di un lavoro confuso e frammentario più volte maneggiato nel tempo anche da parte di programmatori diversi che hanno osservato standard di descrizione differenti.

- Il lavoro eseguito da parte di programmatori con diversi strumenti e stili di programmazione è ipotizzabile anche se si considera il fatto che l'algoritmo in questione è stato sviluppato con due linguaggi di programmazione completamente dissimili per la fase A (linguaggio COBOL) e per le fasi B, C e D (linguaggio C), a maggior ragione se le fasi in questione devono interagire e condividere dati e risultati. In tali casi è sempre auspicabile lo sviluppo di programmi che presentino la medesima struttura ed architettura nonché il medesimo formato di dati su cui si opera.
- Riguardo la fase A dell'algoritmo della mobilità esaminato, non è ben chiaro il motivo dello sviluppo del codice in linguaggio COBOL il quale, seppur ancora utilizzato nell'ambito dei software gestionali di varie aziende, risulta essere un linguaggio datato e ormai sostituito da nuovi e più performanti linguaggi di sviluppo, anche in termini di sintassi logico-aritmetica. Difatti, le aziende che utilizzano tuttora software compilati in COBOL e che hanno la necessità di analizzare e gestire una grande quantità di dati, preferiscono non migrare i propri software ad un più moderno ambiente di sviluppo per via delle ingenti risorse da investire dovute alle molte righe di codice in COBOL presenti sui loro sistemi o perché vincolati dalla propria struttura informatica. Basti pensare che software come quelli bancari, tutt'ora in uso, possono contenere milioni di righe di codice scritte in COBOL risultando quindi troppo oneroso e rischioso il passaggio ad un nuovo sistema. Essendo l'algoritmo qui preso in esame commissionato nel 2015/2016, e quindi presumibilmente totalmente sviluppato ex novo, non si vede ragione di creare un programma software di questo tipo nell'ambiente di sviluppo e nella forma in cui appare, a maggior ragione se si considera la semplicità del tipo di elaborazione richiesta. A tal proposito si pone in essere una questione di natura formale riguardo i termini e le modalità sancite dal ministero nell'assegnazione dello sviluppo dell'algoritmo della mobilità.
- Ciò che si nota visionando il codice in più parti è una difficoltà nella stesura ed una scelta delle variabili assolutamente fuori dagli standard generali. Tale

circostanza rende di difficile comprensione vari passaggi che spesso appaiono superflui o poco chiari. La scarsa chiarezza del codice si evidenzia anche nella mancanza di alcune righe di codice in più punti del listato e di cui non si comprende se trattasi di una omissione voluta o meno.

- Riguardo l'analisi preliminare del codice delle fasi B, C e D si osserva che le funzioni inserite nel listato fanno largo uso di un database di dati e file attraverso i quali presumibilmente vengono gestite le informazioni di input e di output del programma, nonché possibili risultati e valori temporanei di cui non si conosce la struttura ed il contenuto, informazioni quest'ultime necessarie per una corretta verifica del programma stesso. Ogni programma che fa uso di un database di dati deve essere corredato di relativa documentazione che attesti la struttura ed il formato del database stesso in modo da attestarne il corretto uso nel codice sviluppato. Difatti, possibili anomalie dell'algoritmo sviluppato possono derivare non solo da errori inerenti il codice sorgente stesso, ma anche da una errata predisposizione e gestione dei dati di input su cui eseguire l'elaborazione o sui relativi risultati ottenuti.

Anche in questa occasione si evince fin da subito che lo sviluppo del codice inerente alle sole fasi B, C e D è stato eseguito da più programmatori. Questa, seppur pratica usuale, evidenzia un uso di cifre stilistiche e criteri di sviluppo differenti a partire dall'architettura stessa dell'intero programma. Ne è un esempio la mancata standardizzazione delle indicazioni e dei commenti, sempre correlati e necessari in codici come quello preso in esame. Si notino ad esempio commenti nel listato prima in italiano e poi in inglese, nonché commenti, utilizzati tipicamente con lo scopo di spiegare il funzionamento delle successive linee di codice, che denotano la dubbia sicurezza del programmatore su funzioni e porzioni del codice stesso (si veda pag. 11).

Si riporta, come esempio significativo, un estratto del codice dove si evidenzia l'uso inadatto di particolari costrutti logici che determinano una elaborazione

poco fluida e del tutto inefficiente in fase di computazione, nonché risultati relativi del tutto inattesi.

- In ultimo viene interdetta la possibilità di eseguire un controllo diretto ed automatico, attraverso appositi software e compilatori che renderebbero rapida la verifica della correttezza logica e sintattica del programma, in quanto è presente un blocco dei file. Presumibilmente tale blocco è stato apposto al fine di interdire l'accertamento delle funzioni svolte dall'algoritmo che, dunque, deve essere "copiato a mano" (si ricorda che trattasi di oltre 29600 righe solo per la fase A, cui sommare le righe delle successive fasi).
- A margine della disamina eseguita finora, si pone il problema di analizzare e conoscere quale sia la forma di accordo sancita tra il MIUR e l'azienda appaltatrice per lo sviluppo dell'algoritmo in questione.

Tipicamente, lavori della stessa tipologia di quello preso in esame nella presente relazione sono preceduti da documenti che attestino il formale contatto tra chi assegna o richiede il lavoro e l'ente o azienda incaricata di eseguire quanto richiesto. Tale carteggio è fondamentale in quanto ivi vengono specificati i punti focali del lavoro commissionato, da quelli di tipo squisitamente gestionale a quelli di natura prettamente tecnica. Di norma chi commissiona il lavoro deve esplicare in maniera il più esaustiva possibile quanto si richiede al fine di ottimizzare e rendere efficace il lavoro assegnato. In particolare vengono specificati diversi parametri (specifiche tecniche), come, ad esempio, tempi, strumenti e metodi da impiegare.

Alla luce di quanto emerso, si pone la necessità di verificare quanto specificato dal MIUR in occasione dell'assegnazione dell'algoritmo in quanto tale documentazione non è stata fornita.

- Si ritiene necessario, inoltre, sapere in che modo e se sono stati chiaramente esposti dal Ministero tutti i criteri necessari al corretto completamento della procedura di mobilità, al fine di verificare se la società sviluppatrice fosse in

possesso di tutta la necessaria documentazione sulla base della quale sviluppare il software.

La mancanza della documentazione sopra citata, difatti, rende di difficile comprensione i termini del lavoro commissionato che poteva anche prevedere lo sviluppo in una particolare forma (ambiente di sviluppo, linguaggi, strumenti ecc.) a noi, ancora ad oggi, ignota.

- Altra importante zona d'ombra si riscontra nelle modalità di predisposizione materiale dei dati di ingresso (graduatorie, dati insegnanti ecc.) sui quali è stata eseguita l'elaborazione; in particolare, data la carenza di tale documentazione, non si riesce a comprendere se tali dati siano stati ceduti alla società che ha sviluppato l'algoritmo e da essa elaborati o se, una volta predisposto l'algoritmo, questo sia stato ceduto al M.I.U.R. che ne ha gestito il funzionamento (input e output dei dati) sino alla pubblicazione delle nomine dei docenti.

È evidente che la mancanza di tali precisazioni, così come la mancanza dei file richiamati all'interno del codice, del database, dei file che il software utilizza in lettura e scrittura dei dati (non tanto nei contenuti quanto nella forma) nonché delle specifiche tecniche, configura una condotta poco trasparente, nonostante l'intervenuto ordine di ostensione dei dati e degli atti da parte del TAR, nei confronti del Ministero.

Tali omissioni inficiano in maniera irreversibile la possibilità di un completo controllo sulle concrete modalità di utilizzo dell'algoritmo e, quindi, sulle modalità che hanno determinato lo spostamento degli insegnanti sul territorio nazionale.

La consegna parziale dei dati limita difatti l'analisi tecnica richiesta sull'*agere* dell'Amministrazione che oltretutto, come già indicato, si è servita di linguaggi di programmazione non all'avanguardia. Questa scelta non appare condivisibile in quanto tali sistemi non si rilevano sufficientemente performanti (quanto meno con riferimento al linguaggio di programmazione COBOL) e non trova

giustificazione neanche in eventuali necessità inerenti al contenimento della spesa pubblica considerando che da quanto appreso, da fonti esterne, le somme erogate per la realizzazione dell'algoritmo appaiono fuori mercato.

Roma, li 4 giugno 2017.

Ing. Alessandro Salvucci
Università di Roma Tor Vergata
Ricercatore in ingegneria elettronica

Ing. Maurizio Giorgi
Università degli studi di Roma "La Sapienza"
Dottore in ingegneria delle costruzioni edili e dei sistemi ambientali

Dott. Emilio Barchiesi
Università degli studi di Roma "La Sapienza"
Dottore magistrale in Ingegneria matematica

Prof. Ing. Scafidi Matteo
Ingegnere meccanico