Enunț. Se dezvoltă o aplicație care prelucrează o listă cu numere întregi. Datele sunt preluate de la intrarea standard sau dintr-un fișier text. Se cere: (a) să se determine secvența de lungime maximă, cu proprietatea că toate numerele sunt prime; (b) la alegere: (i) să se afișeze elementele listei; (ii) să se afișeze elementele secvenței determinate, ordonată crescător sau descrescător; (iii) să se exporte conținutul listei într-un fișier text indicat.

Se consideră că au fost dezvoltate următoarele module:

- **A**: modul/programul principal;
- **B**: modulul **setupData(list)** –inițializează lista cu elemente (intrarea standard/ fișier text);
- **C**: modulul **readData(list)** – construiește lista cu elemente de la intrarea standard;
- **D**: modulul **importData(list, file)** – construiește lista cu elementele din fișierul **file**;
- **E**: modulul **computeMaxSeq(list, s, f)** – determină o secvență de lungime maximă care îndeplinește proprietatea precizată, care începe pe poziția **s** și se termină pe poziția **f**;
- **F**: modulul **propSeq(list, ps, pf)** – determină o secvență care îndeplinește proprietatea precizată, care începe pe poziția **ps** și se termină pe poziția **pf**;
- **G**: modulul **isPrime(nr)** – verifică dacă un număr **nr** este prim;
- **H**: modulul **sortData(list, s, f, criterion)** – ordonează lista de la poziția **s** până la poziția **f,** folosind criteriul de ordonare **criterion**;
- **I**: modulul **ascSortData(list, s, f)** – ordonează crescător lista de la poziția **s** la poziția **f**;
- **J**: modulul **descSortData(list, s, f)** – ordonează descrescător lista de la poziția **s** până la **f**;
- **K**: modulul **writeData(list)** – permite alegerea opțiunii de prezentare a rezultatelor;
- **L**: modulul **print(list)** – afișează elementele listei la ieșirea standard;
- **M**: modulul **print(list s, f)** – afișează elementele listei, de la poziția **s** la poziția **f**;
- **N**: modulul **exportData(list, file)** – exportă datele din listă în fișierul **file**.

Se consideră următoarele contexte de dezvoltare:

C01. Arhitectura aplicaţiei este clară şi doar modulul principal este dezvoltat şi testat în izolare. Modulele E, F şi G au prioritate la dezvoltare şi integrare. Modulele D, L şi N au cea mai mica prioritate la dezvoltate şi integrare.

C02. Toate modulele sunt dezvoltate simultan și sunt testate în izolare, având aceeași prioritate la integrare. Beneficiarul nu a impus o prioritate de dezvoltare, dar dorește ca să aibă acces la o primă versiune a aplicației, cu toate functionalitățile incluse.

C03. Se doreşte obţinerea unui timp de dezvoltare redus şi se reutilizează modulele C, D, G, M şi N, acestea fiind dezvoltate anterior. Arhitectura aplicaţiei nu este cunoscută la începutul integrării, fiind clară doar după ce toate modulele au fost integrate.

C04. Arhitectura aplicaţiei este clară şi doar modulul principal este dezvoltat şi testat în izolare. Modulele C, D, G, M şi N sunt dezvoltate anterior şi se reutilizează, obţinându-se astfel un timp de dezvoltare redus.

C05. Arhitectura aplicaţiei este clară şi doar modulul principal este dezvoltat şi testat în izolare. Beneficiarul doreşte ca după fiecare sprint de dezvoltare aplicaţia să aibă fiecare dintre funcţionalităţile principale integrate. Fiecare sprint îmbunătăţeşte/rafinează funcţionalităţile integrate anterior.

Se cere:

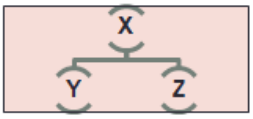
1. Să se construiască diagrama de dependență dintre module;
2. Să se simuleze testarea de integrare pe baza fiecărui context de dezvoltare definit, folosind una din strategiile:
   i. non-incrementală: **big-bang;**
   ii. incrementală: **top-down (depth first, breath first), bottom-up;**
   iii. mixtă: **sandwich**.

   Pentru fiecare strategie se va completa un tabel cu informațiile:
   - **Nr. crt.** – al operaţiei de testare şi nivelul de testare (unit, integration);
   - **Module** – enumerarea modulelor incluse în operaţia de testare curentă;
   - **Driver** – enumerarea modulelor driver folosite (dacă este cazul);
   - **Stub** – enumerarea modulelor stub (dummy, mock, fake) folosite (dacă este cazul);
   - **Bugs** – tipurile de bug-uri care pot fi identificate de operaţia de testare curentă.
3. Studiaţi avantajele şi dezavantajele aplicării fiecărei strategii de integrare, folosind diferite criterii de comparare. Se va completa tabelul de mai jos.

	Strategie de integrare				
Criteriu de comparare	**C1**	**C2**	**C3**	**C4**	**C5**
Integrare					
Arhitectură					
Aplicaţie					
#Drivers					
#Stubs					
Paralelizare (dezvoltare, testare)					

Observații:

- La testarea modulelor X, Y și Z se vor folosi următoarele tipuri de module:
  - **driver** pentru modulele X, Y și Z – $\mathbf{D_X, D_Y, D_Z}$;
  - **stub** (dummy, mock, fake) pentru modulele Y și Z – $\mathbf{S_Y, S_Z}$.

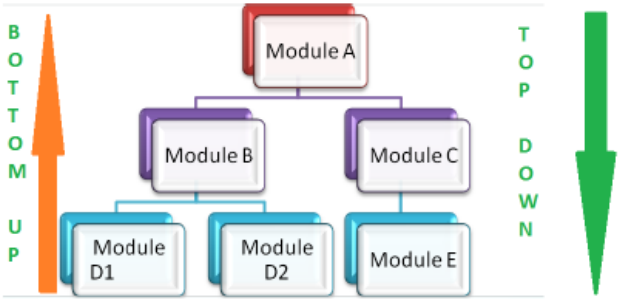


- Pentru modulele X, Y și Z se vor considera că vor fi identificate tipurile de bug-uri:
  a. doar în modulul X, Y sau Z – **Bug(X), Bug(Y), Bug(Z)**;
  b. ***misuse of interface*** – **$\mathbf{MIS_X}$ (Y, Z): la apelul lui Y și/sau Z din X:**
     - *One module makes an error in using the interface of a called module. This is likely to occur in a procedure–call interface. Interface misuse can take the form of wrong parameter type, wrong parameter order, or wrong number of parameters passed* **[Naik, pag.161].**
  c. ***misunderstanding of interface*** – **$\mathbf{MUN_X}$ (Y, Z): la apelul lui Y și/sau Z din X:**
     - *Misunderstanding of Interface: A calling module may misunderstand the interface specification of a called module. The called module may assume that some parameters passed to it satisfy a certain condition, whereas the caller does not ensure that the condition holds. For example, assume that a called module is expected to return the index of an element in an array of integers. The called module may choose to implement binary search with an assumption that the calling module gives it a sorted array. If the caller fails to sort the array before invoking the second module, we will have an instance of interface misunderstanding* **[Naik, pag.161].**

Testare de integrare. Clasificare

1

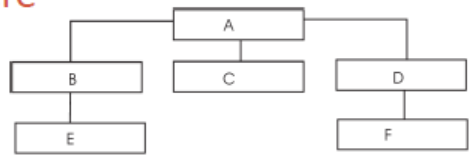
- abordări de integrare/ clasificare:
  - non-incrementală
    - **big-bang;**
  - incrementale
    - **top-down;**
    - **bottom-up;**
  - mixtă
    - **sandwich.**



Integrarea Big-bang. Descriere

2

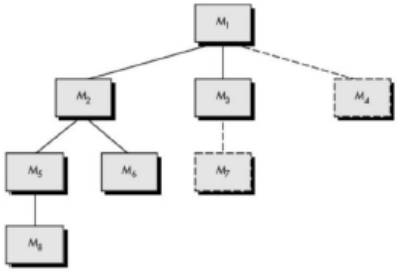
- Procesul de integrare Big-bang:
  1. testare unitară pentru fiecare modul, folosind:
     - un modul driver;
     - câteva module stub;
  2. se combină simultan modulele pentru construirea funcţionalităţii programului;



Integrare incrementală Top-down. Algoritm

3

- după testarea unitară a modulului principal folosind **stub-uri** pentru modulele imediat subordonate, procesul de **integrare Top-down** constă în:
  1. în funcţie de tipul de integrare ales (**depth-first/ breadth-first**), se înlocuieşte un stub cu un modul real;
  2. se testează cu modulul subordonat concret care a fost integrat;
  3. pentru integrarea unui nou modul se repetă paşii 1 şi 2.



- **driver** = modulul principal; nu se folosesc alte drivere;
- **stub** = înlocuieşte un modul direct subordonat;
- **E.g.,**
  - **depth-first:** M1, M2, M5, M8, M6, M3, M7, M4;
  - **breadth-first:** M1, M2, M3, M4, M5, M6, M7, M8.

Integrare incrementală Bottom-up. Algoritm

4

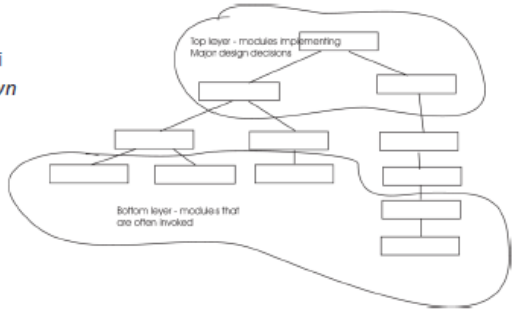
- Procesul de integrare Bottom-up:
  - pentru toate modulele terminale sau clusteri de module se descriu drivere şi se testează;
  - *module terminale:*
    - se înlocuieşte driver-ul cu modulul de pe nivelul imediat superior şi se testează, continuând integrarea modulelor spre partea superioară a structurii programului;
  - *clusteri de module:*
    - se înlătură driver-ul, iar clusterul de module se combină cu alte module continuând integrarea spre partea superioară a structurii programului.

- E.g.
  - componentele se combină în clusterii 1 , 2 şi 3;
  - clusterii se testează folosind driver-ele D1, D2 şi D3;
  - se înlătură D1 şi D2, iar clusterul 1 şi clusterul 2 sunt integraţi în Ma;
  - similar, D3 este înlăturat iar clusterul 3 este integrat cu modulul Mb.
  - Ma şi Mb se vor integra în Mc.

Integrare Sandwich. Descriere

5

- **integrare sandwich** (*engl.* **sandwich integration**):
  - permite construirea şi testarea programului combinând abordările de integrare *top-down* şi *bottom-up*;
- Procesul de integrare Sandwich:
  1. **modulele terminale** (bottom-layer):
     - integrare *bottom-up*;
  2. **modulul principal** (top-layer):
     - integrare *top-down*;
  3. **celelalte module** (middle-layer):
     - integrare *big-bang*.



Testare de integrare. Reguli generale de aplicare

6

- se studiază arhitectura aplicaţiei pentru **identificarea modulelor critice**; aceste module se testează cu **prioritate**;
- testerul este familiarizat cu modulele care se integrează (arhitectura modulelor);
- se aplică o **strategie de integrare** care să permită atingerea **obiectivelor** testării;
- fiecare modul este testat înainte de a execuţia testelor de integrare, i.e., unit testing;
- se utilizează **documentaţia care descrie interacţiunile dintre fiecare două module** (interfeţele acestora şi modul de colaborare).