UNIVERSITÉ PARIS DESCARTES

SDA - Structures de Données et Algorithmes

Equipe pédagogique
Marie-José Caraty, Denis Poitrenaud, Julien Rossit,
Camille Kurtz, Jacques Alès-Bianchetti et Eloi Keita

Cours « Projet »

Analyse du projet
La gestion de messagerie

Passage des recettes la semaine du 8 janvier (en SDA3)
Remise des dossiers (version papier), le lundi 15 janvier au secrétariat
Dépôt des dossiers électroniques dans le puits, le lundi 15 janvier (au plus tard)

Sommaire

1. ANALYSE DE L'APPLICATION

Exemple de ligne de transmission

```
1 caraty gr9a12 26/11/17 14:16:37 0 1
Commentez vos source
2 caraty gr1a4 26/11/17 14:20:02 0 2
 de commenter les fi
3 caraty gr5a8 26/11/17 14:35:23 0 1
Programmez sans nomb
4 caraty gr1a4 26/11/17 14:20:02 0 1
Il est indispensable
5 caraty gr5a8 26/11/17 14:35:23 1 4
 ex. 255, ".txt").
6 caraty gr1a4 26/11/17 14:20:02 0 4
uls fichiers à consu
7 caraty gr1a4 26/11/17 14:20:02 0 3
chiers d'entête : se
8 caraty gr5a8 26/11/17 14:35:23 0 3
ntes littérales, par
9 caraty gr1a4 26/11/17 14:20:02 0 5
lter pour la réutili
.....
```

```
10 caraty all 26/11/17 15:27:57 0 1
Bon courage à tous..
11 caraty gr1a4 26/11/17 14:20:02 1 6
sation.
12 caraty all 26/11/17 15:25:41 1 1
Bon travail...
13 caraty gr5a8 26/11/17 14:35:23 0 2
res magiques (consta
14 caraty gr9a12 26/11/17 14:16:37 1 2
s.
15 caraty all 26/11/17 15:27:57 1 2
. MJC
```


1. ANALYSE DE L'APPLICATION

Simulation de la ligne de communication 1/3

```
1 caraty gr9a12 26/11/17 14:16:37 0 1
Commentez vos source
2 caraty gr1a4 26/11/17 14:20:02 0 2
 de commenter les fi
3 caraty gr5a8 26/11/17 14:35:23 0 1
Programmez sans nomb
4 caraty gr1a4 26/11/17 14:20:02 0 1
Il est indispensable
5 caraty gr5a8 26/11/17 14:35:23 1 4
 ex. 255, ".txt").
6 caraty gr1a4 26/11/17 14:20:02 0 4
uls fichiers à consu
7 caraty gr1a4 26/11/17 14:20:02 0 3
chiers d'entête : se
8 caraty gr5a8 26/11/17 14:35:23 0 3
ntes littérales, par
9 caraty gr1a4 26/11/17 14:20:02 0 5
lter pour la réutili
.....
```

Fichiers texte de paquets-réseaux
stockés suivant leur ordre de réception
(n° **rouge**) dans la messagerie

Paquet-réseau représenté sur 2 lignes
- ligne 1. **Identificateur** de paquet-réseau (**vert**)
  - expéditeur,
  - destinataire,
  - temps d'émission (date et heure),
  - indicateur de fin de message,
  - numéro du bloc de données
- ligne 2. **Bloc de données** (**bleu**) formaté sur `t` caractères

Simulation de la ligne de communication 2/3

```
1 caraty gr9a12 26/11/17 14:16:37 0 1
Commentez vos source
2 caraty gr1a4 26/11/17 14:20:02 0 2
 de commenter les fi
3 caraty gr5a8 26/11/17 14:35:23 0 1
Programmez sans nomb
4 caraty gr1a4 26/11/17 14:20:02 0 1
Il est indispensable
5 caraty gr5a8 26/11/17 14:35:23 1 4
 ex. 255, ".txt").
6 caraty gr1a4 26/11/17 14:20:02 0 4
uls fichiers à consu
7 caraty gr1a4 26/11/17 14:20:02 0 3
chiers d'entête : se
8 caraty gr5a8 26/11/17 14:35:23 0 3
ntes littérales, par
9 caraty gr1a4 26/11/17 14:20:02 0 5
lter pour la réutili
.....
```

Un **message** est constitué d'un **nombre quelconque** de paquets-réseaux de **même identificateur**

Ex: l'identificateur de message du paquet-réseau n°3 est le suivant

```
caraty gr5a8 26/11/17 14:35:23
```

La **longueur du message** est celle du **numéro de bloc de données** lorsque **l'indicateur de fin de message** est à **1**

Ex: repérée dans le paquet-réseau n°5

```
caraty gr5a8 26/11/17 14:35:23 1 4
```

La longueur du message est 4, n° du dernier bloc de donnée :

```
ex. 255, ".txt").
```


Simulation de la ligne de communication 3/3

```
1 caraty gr9a12 26/11/17 14:16:37 0 1
Commentez vos source
2 caraty gr1a4 26/11/17 14:20:02 0 2
 de commenter les fi
3 caraty gr5a8 26/11/17 14:35:23 0 1
Programmez sans nomb
4 caraty gr1a4 26/11/17 14:20:02 0 1
Il est indispensable
5 caraty gr5a8 26/11/17 14:35:23 1 4
 ex. 255, ".txt").
6 caraty gr1a4 26/11/17 14:20:02 0 4
uls fichiers à consu
7 caraty gr1a4 26/11/17 14:20:02 0 3
chiers d'entête : se
8 caraty gr5a8 26/11/17 14:35:23 0 3
ntes littérales, par
9 caraty gr1a4 26/11/17 14:20:02 0 5
lter pour la réutili
.....
```

A la réception, les **paquets-réseau** constituant un **même message** sont **entrelacés** avec **d'autres paquets-réseau** de **messages différents**.

Ex: paquets-réseau n°1, 2 et 3

et dans un **ordre séquentiel** des **numéros** de **blocs de données** qui n'est **pas garanti**

Ex: Le message d'identification

```
caraty gr5a8 26/11/17 14:35:23
```

est constitué, par ordre de réception, des paquets-réseau n° 3, n° 5, n° 8 et n° 13 qui correspondent au bloc de données n° 1, n° 4, n° 3 et n° 2

Organisation mémoire – Analyse préliminaire 1/2

L'organisation en mémoire se fait à la réception d'un paquet-réseau
Après analyse du problème,

A la réception d'un **paquet-réseau**
// Organisation en mémoire du paquet-réseau
si le **message** correspondant au paquet-réseau est un message **déjà en cours de réception**
- **mémoriser le bloc de données** correspondant dans le conteneur de blocs avec les autres blocs de données déjà reçus du message

sinon
- **mémoriser le message** dans le conteneur de messages avec les autres messages en cours de réception
- **mémoriser le bloc de données** (premier bloc reçu du message) du paquet-réseau reçu dans le conteneur de blocs associé au message

Edition des résultats – Analyse préliminaire 2/2

Edition des résultats attendus
Si le **message** est réceptionné dans son **intégralité**
- Affichage à l'écran du **message ordonné**
- Ajout du **message ordonné** en fin de la mailbox du destinataire
- **Mise à jour** éventuelle du **fichier log**

Remarque : Mise à jour du fichier log à intégrer lors de l'organisation mémoire et la perte de paquet

Intégralité du message nombre de paquets reçus égal à la longueur du message
Ex: Le message d'identification

```
caraty gr5a8 26/11/17 14:35:23
```

À la réception du paquet-réseau n° **5** (indicateur de fin de message) la longueur du message est repérée : **4** `(caraty gr5a8 26/11/17 14:35:23 1 4)`
Ce n'est qu'à la réception du paquet n° **13** que le message est reçu dans son intégralité : **3** paquets-réseau du message ont déjà été reçus : n° **3**, n° **5**, n° **8**

Message ordonné suivant l'ordre séquentiel des numéros de blocs de données est constitué, par ordre de réception, des paquets-réseau n° **3**, n° **5**, n° **8** et n° **13** qui correspondent au bloc de données n° **1**, n° **4**, n° **3** et n° **2**
Le message ordonné est constitué des blocs n° **1**, n° **2**, n° **3** et n° **4**

Analyse – Identification de conteneurs 1/2

L'analyse du problème fait émerger 2 fonctionnalités
Mémoriser les messages en cours de réception
Mémoriser les blocs de données constituant un message donné
Qui pose le problème du choix du conteneur le plus adapté

Rappel : *un conteneur est un TAD (structure de données + fonctions) qui permet les opérations élémentaires en mémoire (stocker et rechercher) des éléments de type quelconque*

Organisation mémoire de la messagerie

- un **conteneur des messages** (en cours de réception) **et par message** (en cours de réception)
  - **un conteneur des blocs** de données constituant le message

Analyse – Identification de conteneurs 2/2

Quels conteneurs choisir ?

Un nombre quelconque de messages peut être transmis
La longueur des messages (nb de ses blocs de données) est quelconque

- un tableau statique n'est pas le plus adapté
- Une gestion en mémoire dynamique est préférable
  - choix d'une gestion ad hoc
  - **choix d'un TAD conteneur en mémoire dynamique**

TAD « spécifiques » : pile, liste, file…
TAD « élémentaires » : tableau dynamique, chaîne, …

Pour la gestion des messages
séquence de messages en cours avec accès direct sans notion d'ordre : **?**

Pour la gestion des blocs de données
séquence de blocs accessible par une seule extrémité et ordonnée par numéro de bloc : **?**

Structuration des données – du paquet-réseau

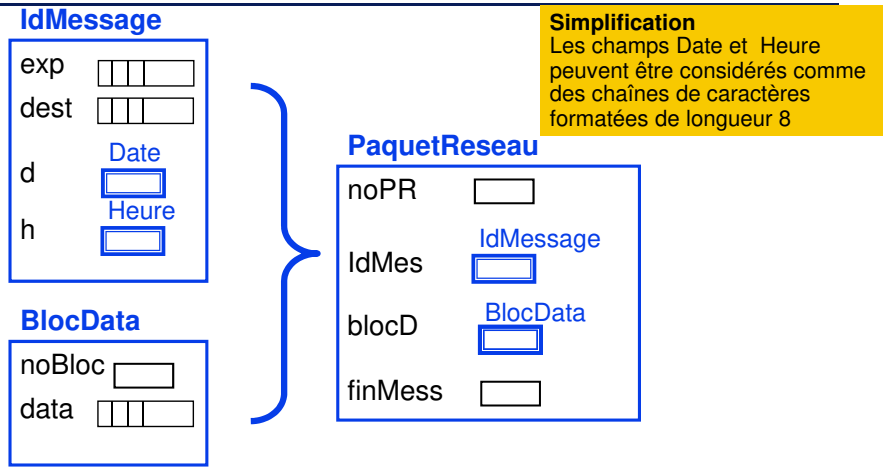
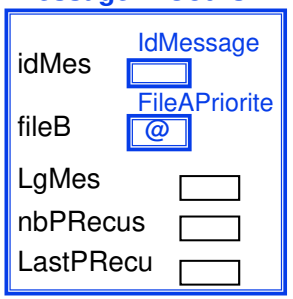


Schéma des données concrètes des composants `IdMessage`, `BlocData` et `PaquetReseau`

Structuration des données – Les conteneurs : Liste et File à Priorité

Schéma de la donnée concrète du composant `Messagerie`



Rappel : Les éléments de liste du composant **Liste** sont d'un type généraliste **Item** à spécialiser
Les éléments de liste de la messagerie sont les messages en cours de réception de type `MessageEnCours`
Spécialisez `Item` en `MessageEnCours`

Rappel : Les éléments de file du composant **FileAPriorité** sont d'un type généraliste **Item** à spécialiser
Les éléments de file sont les blocs de données du message de type `BlocData`
Rem : renommer dans tout le composant le type **Item** en **ItemF**
Spécialisez `ItemF` en `BlocData`

Fonctionnalités de (ré)organisation mémoire et/ou d'édition de résultat(s) – Algorithmique 1/3

Fonctionnalités à intégrer dans l'algorithme d'organisation en mémoire

1) **Si** (le paquet-réseau **pR** traité correspond à un message en cours de réception **m**)
- entrer dans la file à priorité des blocs de données du message le bloc de données du paquet-réseau

Sinon
- ajouter à la liste des messages en cours de réception un élément (**m** de type **MessageEnCours**) de même identificateur que le paquet-réseau
- ajouter au fichier log, la trace de détection de nouveau message

mettre à jour **m.nbPRecus et m.lastPRecu**

entrer dans la file à priorité des blocs de données du message le bloc de données du paquet-réseau

Rem : Pour le test, balayer la liste des messages en cours et tester si l'identificateur du paquet-réseau traité est « égal » à l'identificateur du message en cours
=> Ajouter au composant **IdMessage** la fonction :

```
bool estEgal(const IdMessage& id1, const idMessage& id2);
```


Fonctionnalités de (ré)organisation mémoire et/ou d'édition de résultat(s) 2/3

2) Si (une fin de message est détectée dans le paquet-réseau traité **pR**)
- mettre à jour dans le message en cours de réception **m** la longueur du message égal au numéro du bloc de données du paquet-réseau **pR**
- ajouter au fichier log, la trace de détection de fin de message

Rem : Mise à jour (**m.LgMes = pR.blocD.noBloc**)

3) Si (le message est reçu dans son intégralité)
afficher le message complet (blocs de données ordonnés) du message suivant les formats définis
- le message complet (blocs de données ordonnés) à l'écran
- le message complet (blocs ordonnés) en fin de la mailbox du destinataire
- la trace d'archivage dans le fichier log

Rem : Test booléen du message en cours de réception **m** reçu dans son intégralité (**m.LgMes == m.nbPRecus**)

Fonctionnalités de (ré)organisation mémoire et/ou d'édition de résultat(s) 3/3

4) Si (perte de paquet estimée pour un message en cours de réception)
- supprimer le message en cours dans la liste des messages en cours
- ajouter au fichier log, la trace de perte de paquet suivant le format défini

Rem : Pour un message en cours de réception **m**, on estime une perte de paquet en fonction du numéro du paquet-réseau **pR** en cours de traitement (**pR.noPR ≥ m.lastPRecu + 10**)

Composant de File à Priorité

Munir le type des éléments de la file de la relation d'ordre

1) Ajouter au composant **BlocData** la fonction **enOrdre**

```
/**
 * @brief relation d'ordre entre 2 positions
 * @param[in] b1 : le 1er bloc de données
 * @param[in] b2 : le 2ème bloc de données
 * @return true si b1 et b2 sont ordonnés, false sinon
 */
bool enOrdre(const BlocData& b1, const BlocData& b2)
```

Créer le composant de file à priorité (FileAPriorite .h et .cpp) par recopie du composant de file (File .h et .cpp)

2) Modifier la fonction **entrer** de **FileAPriorite**

```
void entrer(FileAPriorite& f, const Item& it);
```

algorithme : quelque soit la donnée concrète (tableau dynamique, chaîne, …)
insérer l'élément **it** dans la file à priorité de manière à ce que deux éléments successifs **e1** et **e2** soient tels que **enOrdre(e1, e2)**

Architecture logicielle 1/4

Composant `IdMessage` (`.h` et `.cpp`) de gestion d'identificateurs de message
Composant `BlocData` (`.h` et `.cpp`) de gestion de blocs de données
Composant `PaquetReseau` (`.h` et `.cpp`) de gestion de paquets-réseau
Composant `MessageEnCours` (`.h` et `.cpp`) de gestion de messages en cours
Composant `Messagerie` (`.h` et `.cpp`) de gestion de la messagerie
...
Tous les composants techniques du cours/TD/TP (sauf la pile)
(ré-)utilisés liste, tableau dynamique, chaîne
ou conçu file à priorité (adaptée à partir du composant de File)
avec les **items** correspondants **spécialisés**

L'application le **`main` : point d'entrée du logiciel**

Attention : En cas de besoin de **deux listes** d'éléments de **types distincts (T1** et **T2)**
(1) **recopier** le composant Liste dans **Liste2** (.h et .cpp)
(2) **remplacer** dans le composant Liste2 **Item** par **Item2**
Vous pourrez alors **spécialiser Item** et **Item2** par les deux types souhaités

Architecture logicielle – `IdMessage` 2/4

Composant `IdMessage` (`.h` et `.cpp`) d'identificateur unique de message

Sa donnée concrète (structure nommée `IdMessage`)
- ...cf. Transp. 11

Spécification
- ✓ `lire` Lire la variable `id` de type `IdMessage`
  à partir d'un flot en entrée `is` (de type `istream`) ouvert
  Précondition : le flot est en état d'être lu
- ✓ `ecrire` Ecrire la variable `id` de type `IdMessage`
  dans un flot de sortie ouvert
  Précondition : le flot est en état d'être écrit
- ✓ .... Toute fonction utile

Même modèle **pour les Composants** `BlocData et PaquetReseau`

Attention **Tous les flots sont à déclarer et à assigner au fichier physique dans le `main()` et passé si besoin en paramètre**

Architecture logicielle – `MessageEnCours` 3/4

Composant `MessageEnCours` (`.h` et `.cpp`) de gestion des messages en cours de réception

Sa donnée concrète (structure nommée `MessageEnCours`)
- ... Transp. 12

Spécification

```
/** Initialiser et allouer en mémoire dynamique
    la variable m de type MessageEnCours */
void initialiser(MessageEnCours& m, const PaquetReseau& p);

/** Désallouer la variable m de type MessageEnCours */
void detruire(MessageEnCours& m);
```

✓.... Toute fonction utile

Architecture logicielle - `Messagerie` 4/4

Composant `Messagerie` (`.h` et `.cpp`) de gestion de la messagerie

Sa donnée concrète (structure nommée `Messagerie`)
- ... cf. Transp. 12

Spécification

```
/** allouer et désallouer la variable m de type Messagerie */
void initialiser(Messagerie& m);
void detruire(Messagerie& m);

/** @brief Reception d'un paquet-réseau */
void recevoirPaquetReseau(std::istream& is, PaquetReseau& p);

/** @brief Organiser un paquet-réseau dans la messagerie */
void traiterPaquetReseau(Messagerie& m,
                         const PaquetReseau& p);
```

Développement agile par sprints (1/2)

En Génie logiciel, le processus de développement de production d'un logiciel est fondé sur un **cycle de développement court** (analyse, spécification, codage, test) permettant de progresser de façon sure dans le développement logiciel

Nous choisissons un **cycle de développement agile** par validation de **sprints**

Attention **Chaque Sprint** donne lieu à un développement matérialisé par un **projet** (nommé `Sprintx`, avec `x` son numéro) **à valider**

Lorsque vous avez validé le `Sprintx` :
conservez intact ce Sprint
et passez au développement du sprint suivant

Développement agile par sprints (2/2)

Attention à la copie des sources dans le nouveau projet

Pour éviter le partage de source
(1) vous recopierez toutes les entités logicielles (utiles) du `sprintx` dans le répertoire des sources de votre nouveau projet `Sprinty`
Pour la copie des sources
<clic droit> sur le projet *sprintx*
sélectionnez « Ouvrir le dossier dans l'Explorateur de fichier »
sélectionnez les sources à copier puis <CTRL C> (copier)
<clic droit> sur le projet *sprinty*
sélectionnez « Ouvrir le dossier dans l'Explorateur de fichier »
<CTRL V> (coller)

(2) pour visualiser les sources copiés dans le projet `Sprinty`
sélectionnez le projet, puis « ajouter », « élément existant »
en sélectionnant tous les sources utiles qui seront alors automatiquement répartis dans les parties *entête* (.h) et *source* (.cpp)
Vous adapterez alors les fichiers sources en fonction des objectifs du nouveau sprint

Développement en 6 sprints – Sprint #1 1/6

Sprint #1

Ajoutez un élément de type Projet de nom `Sprint1`.

- **Analyse fonctionnelle** : Lecture et affichage de toutes les informations des paquets-réseaux transmis via le fichier texte simulant la ligne de communication (JDT de développement).
- **Spécification** : Développez les composants (.h et .cpp) **`IdentificateurMessage`** et **`PaquetReseau`** ainsi que le programme de test de ces composants correspondant à l'analyse fonctionnelle précédente.
- **Codage** : Programmez les entrées-sorties à partir de flot. Ces flots pourront être connectés à des fichiers texte ou aux entrées-sorties standards.
- **Test** : Exécutez le programme de test sur le JDT de développement que vous trouverez sur **`COMMUN`** dans le répertoire **`projet`**.
- Vérifiez que les résultats obtenus sont bien les résultats attendus.
  Bilan du test : la vérification précédente valide le Sprint#1.

Développement en 6 sprints – Sprint #2 2/6

Sprint #2

Ajoutez un projet `Sprint2` et copiez dans son répertoire les sources de Sprint #1.
Visualisez les composants dans le projet.

- **Analyse fonctionnelle** : Organisez en mémoire les messages en cours de transmission. Chaque identificateur de message de paquet-réseau lu sur la ligne de transmission, sera ajouté à la liste des messages en cours (via l'initialisation d'une variable de type `MessageEnCours`) si et seulement si le message n'est pas déjà un message en cours de transmission. Les blocs de données ne seront pas encore organisés en mémoire.
  Indication : Vous pourrez utiliser une liste implémentée à partir d'un conteneur dynamique.
- **Spécification** : Ajoutez la fonction **`TraiterPaquetReseau`** au composant **`Messagerie`**.
- **Codage** : Développez une première version de la fonction **`TraiterPaquetReseau`** (correspondant à l'analyse fonctionnelle donnée) ainsi que le programme de test.
- **Test** : Vérifiez (par affichage) pour chaque paquet réseau lu (à partir de la ligne de transmission) que la liste des messages en cours de transmission est bien la liste attendue.

Développement en 6 sprints – Sprint #3 3/6

Sprint #3

Ajoutez un projet `Sprint3` et copiez dans son répertoire les sources de Sprint #2.
Visualisez les composants dans le projet.

- **Analyse fonctionnelle** : Organisez en mémoire les blocs de données des paquets-réseau. Pour chaque paquet-réseau lu sur la ligne de transmission, organisez son bloc de données en mémoire.
  Indication : Vous pourrez utiliser une file développée en cours ou en TP.
- **Spécification** : fonction **`TraiterPaquetReseau`** du composant **`Messagerie.`**
- **Codage** : Développez une deuxième version de la fonction **`TraiterPaquetReseau`** (correspondant à l'analyse) ainsi que le programme de test.
- **Test** : Vérifiez (par affichage) pour chaque paquet réseau lu (à partir de la ligne de transmission) que le conteneur des blocs de données est bien le conteneur attendu (les blocs sont stockés suivant leur ordre d'arrivée sur la ligne de transmission).

Développement en 6 sprints – Sprint #4 4/6

Sprint #4

Ajoutez un projet `Sprint4` et copiez dans son répertoire le composant de File sur COMMUN que vous renommerez `FileApriorite` et le composant Position (`.h` et `.cpp`) du TP 6.
Visualisez les composants dans le projet.

- **Analyse fonctionnelle** : Développez un composant de file à priorité à partir du composant de file.
- **Spécification** : Composant **`FileAPriorite`** (.h et .cpp)
- **Codage** : Adaptez le composant de File que vous avez choisi à une file à priorité en codant la fonction **`enOrdre`** dans le composant Position (choisissez un critère simple, par exemple, l'ordre de l'abscisse de la position) et en modifiant la fonction **`entrer`** de la file. Codez le programme de test (adaptez un test du TP 6).
- **Test** : Vérifiez (par affichage) que la file obtenue est bien la file attendue suivant l'ordre des positions que vous avez choisi.

Développement en 6 sprints – Sprint #5 5/6

Sprint #5

Ajoutez un projet `Sprint5` et copiez dans son répertoire les composants utiles du `Sprint3` et du `Sprint4`. Visualisez les composants dans le projet.

- **Analyse fonctionnelle** : Organisez en mémoire les blocs de données des paquets réseau dans le conteneur file à priorité développé au sprint #4.
- **Spécification** : Composant **`Messagerie`** (`.cpp`)
- **Codage** : Développez une troisième version de la fonction **`TraiterPaquetReseau`** (correspondant à l'analyse) ainsi que le programme de test.
- **Test** : Vérifiez (par affichage) pour le JDT de développement que les blocs de données sont bien affichés suivant leur ordre de n° de bloc.

Développement en 6 sprints – Sprint #6 6/6

Sprint #6

Ajoutez un projet `Sprint6` et copiez dans son répertoire les composants utiles du `Sprint5`.
Visualisez les composants dans le projet.
Ajouter au produit logiciel obtenu les fonctionnalités suivantes

- **Analyse fonctionnelle** : Ajoutez les fonctionnalités de la messagerie demandée
  (1) dès qu'un message est intégralement reçu : – affichage à l'écran du message,
  – ajout du message à la *mailbox*, – mise à jour du fichier *log*,
  (2) mise à jour du fichier log en cas de détection
  – de nouveau message, – de fin de message, – de perte de paquet et
  – de fin de transmission avec la liste des messages en cours restant.
- **Spécification** : Composant **`Messagerie`**
- **Codage** : On développera une quatrième version de la fonction **`TraiterPaquetReseau`** (correspondant à l'analyse) ainsi que le programme de test.
- **Test** : Vérifiez pour le JDT de développement que *l'affichage à l'écran* et les *fichiers mailbox* et *log* créés sont bien les fichiers attendus.

Evaluation de projet 1/3

NS	Note de Sprint	[15 pts]
PA	Présentation de l'application	[1 pt]
GD	Graphe la dépendance	[0,5 pt]
BV	Bilan de validation	[1 pt]
BP	Bilan de projet	[0,5 pt]
QC	Qualité de code	[2 pts]
RS	Respect des spécifications	[1 pt]

Comme en IAP : principe de modulation [1 pt], pénalités [-1pt] (absence -n° de groupe, -sommaire (paginé), -pagination de l'ensemble du dossier (dont les annexes)).

Attention au 0/20
Le code de l'application doit absolument être structuré en composants (.h et .cpp)

Evaluation de projet 2/3

PA - Présentation de l'application (au plus une page, police 12)

GD - Inutile d'indiquer dans le graphe la dépendance du .cpp d'un composant à son entête (.h)
Pertinence des fonctions des composants

BV - Donner le bilan de validation de tous les sprints validés sur le JDT (Jeu de Données de Test) de développement. Il s'agit de répondre à la question « votre logiciel est-il 0-défaut sur le JDT de développement pour les différents sprints développés.
Attention : Pour le sprint de plus haut niveau validé en développement, donnez en annexe les traces d'exécution sur le JDT de développement et les fichiers résultat d'exécution

BP - Le retour d'expérience (les difficultés, ce qui est réussi, ce qui pourrait être amélioré)

Evaluation de projet 3/3

QC - Donner le code du sprint de plus haut niveau validé à la recette (préciser le numéro de Sprint)
Dans la qualité de code, la qualité des composants sera prise en compte :
Cartouche pour tous les composants (.h et .cpp)
Documentation de l'entête (.h) : structure de données et prototypes de fonction (rôle, paramètres, mode de passage et préconditions)
Rappel : La documentation des prototypes de fonction n'est pas demandée dans le .cpp
Modularité et taille des modules (dont le `main`)
Factorisation (absence de code redondant)
Absence de nombres magiques

RS - Le respect des spécifications comprend entre autres toutes les spécifications données pour la composition du dossier de développement logiciel (cf. p. 3 et 4 du projet)

Développement agile par sprints (2/2)

Conseil

dans le cas où votre programme ne fonctionne pas sur l'ensemble des fonctionnalités demandées

Les projets Sprint sont indispensables pour montrer l'état d'avancement de votre logiciel. Chaque sprint validé correspond à un bon travail que vous pouvez démontrer (tests réussis) et qui vous rapporteront des points