Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

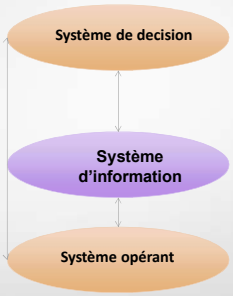
A. Benghalia
abderaouftb@yahoo.fr

1

Décider des actions.

Collecter, mémoriser, traiter et distribuer l'information.

Assurer les fonctions :
. facturer les clients,
. fabriquer les produits,



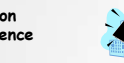
Système d'information :
« Ensemble de moyens humains et matériels et
de méthodes permettant de réaliser les traitements nécessaires sur les différentes formes d'information pour la bonne conduite de l'organisation »

3

Diversité des produits

Augmentation de la qualité

Augmentation de la concurrence



Augmentation de la réactivité

Ouverture des marchés

Améliorer en permanence la qualité, le coût et la productivité :
C'est l'enjeu majeur des entreprises

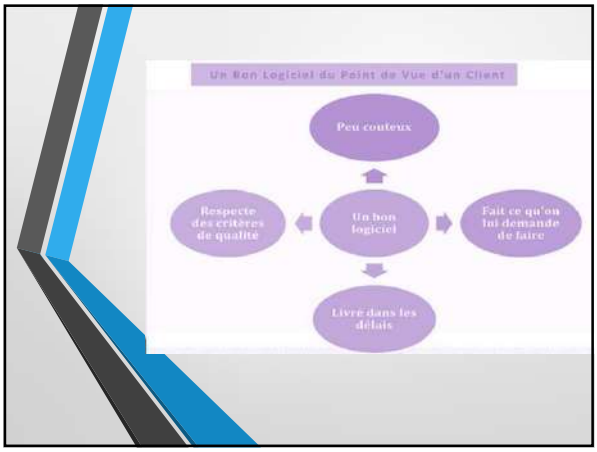
Quels outils donner au décideur pour comprendre, dimensionner, piloter, et gérer ces systèmes

UEF32 – 2016/2017

C'est quoi un logiciel ?

- D'une manière assez abstraite, on peut dire qu'un logiciel s'exécute sur un ordinateur et traite automatiquement de l'information.
- Nous connaissons tous des logiciels. Les traitements de textes, les tableurs, les jeux vidéo, les agendas, les moteurs de recherche, les sites sociaux ou les outils de dessins sont tous des logiciels.
- Ils s'exécutent tous sur un ou plusieurs ordinateurs et traitent de l'information.

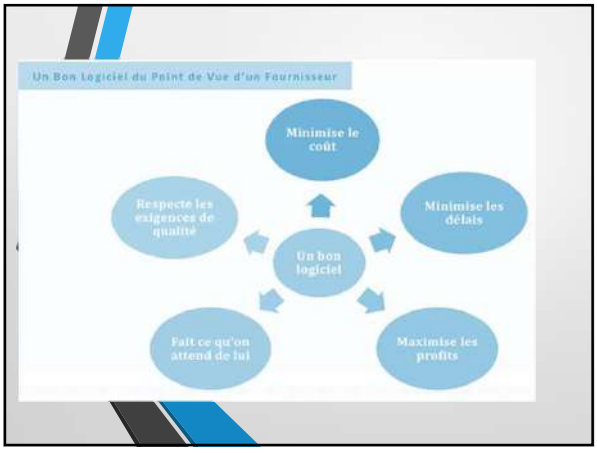
5



(Rappel)

- Qu'est ce qu'un bon logiciel ?
- - Il doit être maintenable : changement régulier, bonne documentation.
- - Le Software doit être robuste : plusieurs utilités.
- - Le Software doit être efficace : bonne utilisation des ressources (Processeur, Mémoires…)
- - Le Software doit fournir une interface appropriée à l'utilisateur : l'interaction Homme/Machine.

7



LOGICIEL – GENIE LOGICIEL

LOGICIEL – GENIE LOGICIEL

- Il est facile de donner une définition abstraite et des exemples de logiciels, il est bien plus compliqué de proposer une définition concrète qui délimite ce qu'est un logiciel et ce que cela n'est pas.
- Est-ce qu'une version préliminaire d'un logiciel est un logiciel ? Est-ce que le code d'un logiciel est un logiciel ? Est-ce que la documentation fait partie du logiciel ?
- En effet, un logiciel est une entité informatique composée d'un ou plusieurs fichiers (code source, binaire, librairie, documentation, guide d'installation, etc.).

LOGICIEL – GENIE LOGICIEL

- Dans l'ensemble de fichiers qui composent un logiciel, il n'existe pas de fichier obligatoire. En fait, cela dépend de l'état d'avancement du logiciel. En effet, sans code source il n'est pas réellement possible d'exécuter un logiciel. Le code source d'un logiciel n'est pourtant pas présent au début de la vie du logiciel ni même lors de la livraison.
- Une fois cette définition posée, il est aisé de comprendre que le but du génie logiciel est de définir la façon dont construire l'ensemble des fichiers qui caractérisent un logiciel.

LOGICIEL – GENIE LOGICIEL

- Autrement dit, le génie logiciel répond aux questions suivantes : Comment construire le code source ? Comment construire la documentation ? etc.
- Il n'est pas rare d'utiliser plusieurs autres termes tels que programme, application ou produit à la place du terme logiciel. Ces termes sont des synonymes.

Concepts du Génie logiciel

13

Contexte et enjeux

- Réduire les différents coûts de développement du logiciel en optimisant le processus de production d'un logiciel,
- Produire des logiciels de qualité selon les besoins des client,
- Satisfaire les clients en respectant le délai de livraison et le coût de réalisation prévu.

Logiciel maintenable, robuste, fiable, efficace et doit offrir une interface utilisateur appropriée

14

- Le génie logiciel (software engineering) représente l'application de principes d'ingénierie au domaine de la création de logiciels,
- Le « génie logiciel, Software Engineering » a été introduit en 1968 lors d'une conférence internationale sur la « crise du logiciel » pour faire face à la mauvaise qualité des logiciels et le non respect des besoins des clients,
- Il consiste à identifier et à utiliser des méthodes, des pratiques et des outils permettant de maximiser les chances de réussite d'un projet logiciel,
- Le génie logiciel et la méthodologie s'efforcent de couvrir tous les aspects de la vie du logiciel (Cycle de vie standard d'un logiciel).

15

Logiciel : Client – Utilisateur

- Si un logiciel traite automatiquement de l'information, c'est parce que ce traitement intéresse quelqu'un.
- Un logiciel a pour vocation de servir des utilisateurs.
- L'utilisateur d'un logiciel est une personne qui interagi directement avec le logiciel car elle a besoin des services rendus par celui-ci.
- Lors de la naissance du logiciel, l'utilisateur est donc la personne qui émet ses souhaits quant aux services que devra rendre le logiciel..

16

Logiciel : Client – Utilisateur

- Le terme consacré est exigences.
- Un utilisateur exprime donc ses exigences sur le futur logiciel.
- Le terme Client est souvent employé pour désigner la personne qui exprime les exigences.
- L'utilisation du terme Client plutôt que du terme Utilisateur vise à introduire la notion marchande du logiciel.
- Le client est la personne qui paye la construction du logiciel (Le propriètaire)

17

- Il est important de noter que le génie logiciel s'adresse principalement aux développeurs et pas aux utilisateurs.
- Comme nous le verrons, les exigences des utilisateurs sont principalement exprimées en langage naturel.
- La traduction vers des fichiers interprétables par un ordinateur est donc à la charge du développeur.
- Le génie logiciel a pour objectif de professionnaliser les développeurs et propose des solutions afin de faciliter ce travail de traduction.

18

Modélisation et Concepts

19

Modélisation - Modèle

- La modélisation est la conception d'un modèle.
- Objectif principal de la modélisation = maîtriser la complexité
- Modéliser = abstraire la réalité pour mieux comprendre le système à réaliser
- Le modèle doit être relié au monde réel
- un modèle permet de modéliser le fonctionnement d'un ensemble de programmes informatiques.
- Un modèle permet de communiquer : entre les membres de l'équipe génie logiciel et même avec les utilisateurs pour vérifier et valider la cohérence de l'architecture logiciel et leur contexte,

20

Activité :
Est une transformation qui ajoute de la valeur. elle implique un certain nombre de ressources (des personnes, de l'argent , des matières et du matériel) pour transformer un objet d'entrée en un objet de sortie.

Processus :
Est un enchainement logique d'activités dans le temps afin de réaliser un but.

.

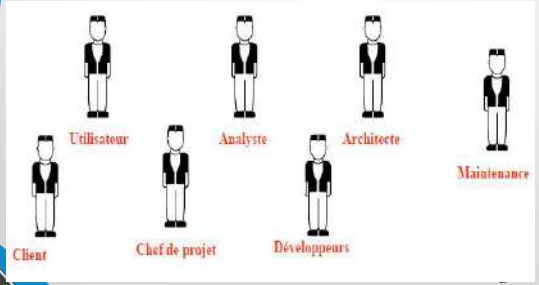
21

Pourquoi et comment modéliser en orienté objet

- Relier le modèle au monde réel par la notion d'objet
- Orienté objet = abstraire et décomposer le système informatique en objets
  - Le monde réel est constitué d'objets physiques ou immatériels
  - Tracer les objets virtuels de modélisation depuis les objets du monde réel
    - Relier les objets (réels) du problème et les objets (virtuels) de la solution

22

Acteurs de projet logiciel



Introduction

- La conduite de projet -> Comment produire un logiciel ?
- Produire -> code source + … -> exécutable.
- **Participants :**
  - Utilisateur (Utiliser)… >a besoin d'un logiciel
  - Propriétaire (Posséder)…> demande au développeur de réaliser le logiciel
  - Développeur (Développer)…> réalise le logiciel qui correspond au besoin de l'utilisateur

24

Introduction

25

Introduction

- Si l'utilisateur exprime ses exigences sur le futur logiciel, c'est pour que celui-ci soit développé.
- Plusieurs termes existent et permettent de catégoriser les différents métiers nécessaires au développement du logiciel.
- Le terme développeur est le terme le plus générique. Un développeur est une personne qui participe au développement du logiciel.

26

- Les termes maitrise d'ouvrage (MOA) et Maîtrise d'œuvre (MOE) sont très souvent employés dans l'industrie.
- La MOA désigne la société utilisatrice alors que la MOE désigne la société qui prend en charge les développements.
- Le terme métier est parfois utilisé pour désigner l'utilisateur.
- Une personne du métier est un utilisateur, souvent non informaticien, qui utilise le logiciel dans son métier de tous les jours.
- Le plus grand problème du génie logiciel est de faire en sorte que les développeurs construisent un logiciel qui répond parfaitement aux exigences des utilisateurs.
- C'est un problème de traduction.

27

- **Comment traduire les exigences dans des fichiers qui constituent un logiciel ?**
- **Autrement dit, comment rendre les exigences des utilisateurs interprétables par un ordinateur ?**

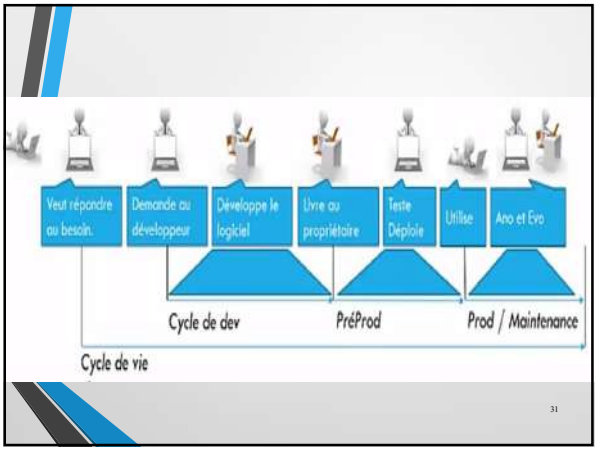
28

Cycle de vie détaillé d'un logiciel

29

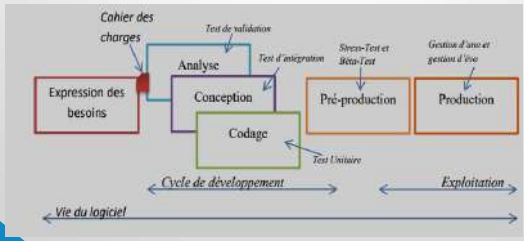
- Projet de développement :
  - L'utilisateur a un besoin
  - Le propriétaire exprime clairement ce besoin
    - Cahier des charges
  - Le développeur spécifie ce qu'il est capable de produire
    - Réponse au cahier des charges (spécification)
  - Le propriétaire accepte la proposition du développeur
    - Cycle de développement
  - Enfin, le développeur livre le logiciel au propriétaire
    - Livraison

30



31

- I) Développement d'un logiciel
- II) Production et maintenance



32

I) Développement d'un logiciel

- Nous avons vu que la vie d'un logiciel commence dès qu'un utilisateur exprime ses exigences.
- En génie logiciel, le terme consacré pour cette activité est **expression des exigences**.
- Le terme anglais est **requirement** pour parler des exigences.
- L'expression des exigences est à la charge de l'utilisateur qui a pour objectif de lister toutes ses exigences, de vérifier leurs compatibilités et de fixer les priorités entre elles.
- Le cahier des charges est le document d'aboutissement de cette activité. Le cahier des charges contient toutes les exigences exprimées par un utilisateur.

33

I) Développement d'un logiciel

- Dès que des exigences sont exprimées, le développement du logiciel commence véritablement. Trois activités sont alors réalisées par les développeurs

34

1 – L'analyse du besoin (Analysis)

- Lors de cette activité, les développeurs ont pour objectif de bien comprendre les exigences des utilisateurs afin de savoir quel logiciel ils peuvent développer.
- Cette activité permet de spécifier les contours du logiciel, de définir les services qu'il devra fournir mais aussi les services qu'il n'a pas à fournir parce que ses utilisateurs n'en ont pas besoin ou parce qu'ils sont trop couteux à développer.
- C'est en réalisant l'analyse que les développeurs peuvent estimer le coût de développement du logiciel.

35

1 – L'analyse du besoin (Analysis)

- A l'issue de l'analyse du besoin, les développeurs peuvent rédiger les tests de validation.
- Ces tests sont des scénarios d'utilisation du logiciel. Ils expriment ce que fera le logiciel.
- Ces tests sont aussi appelés des tests de recette car si le logiciel passe ces tests lorsqu'il sera développé, alors l'utilisateur pourra payer le développement

36

2- L'architecture ou conception (Design)

- Cette activité permet aux développeurs de prévoir l'organisation du logiciel en différents modules de code. L'organisation en modules à deux objectifs.
- Le premier vise à définir les qualités du logiciel en termes de performance, de tolérance aux pannes ou de réutilisabilité par exemple.
- Le deuxième vise à préparer la séparation des taches de développement afin de diminuer les délais de développement.
- Pour autant, lorsque les différents modules seront développés, il faudra alors les intégrer afin de constituer le logiciel.

37

2- L'architecture ou la conception (Design)

- La conception doit alors absolument anticiper tous les problèmes d'intégration.
- A cet effet les tests d'intégration sont rédigés pendant la conception.
- Ils ont pour objectif de spécifier la manière dont les différents modules pourront communiquer et donc être intégrés.

38

3 – La réalisation de code

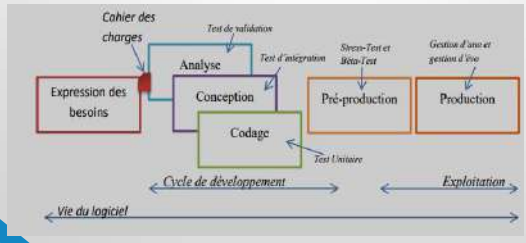
- Cette activité consiste à réaliser le code source du logiciel.
- C'est lors de cette activité que des choix de codage sont effectués (choix des bibliothèques, choix des conventions de codage, etc.).
- Les tests unitaires sont effectués lors de cette activité.
- Un test unitaire vise à tester un élément atomique d'un code (une classe pour les langages objet).

39

- Une méthode de développement précise la façon dont ces trois activités doivent être réalisées.
- Historiquement, les première méthodes préconisaient de réaliser ces trois activités séquentiellement, les unes après les autres.
- Il fallait commencer l'analyse du besoin, s'attaquer à la conception une fois l'analyse terminée et enfin commencer la réalisation de code dès que la conception avait bien délimitée les modules ainsi que la séparation des taches. Aujourd'hui plusieurs méthodes existent.
- La plupart d'entre elles sont itératives. Elles préconisent la réalisation de plusieurs boucles de développement : analyse, conception, codage, analyse conception, codage, etc.

40

- I) Développement d'un logiciel
- **II) Production et maintenance**



41

II) Production et maintenance

- Les personnes qui construisent le code source d'un logiciel considèrent trop souvent que logiciel est fini lorsque le code source est écrit.
- Pour les personnes qui travaillent à l'exploitation du logiciel, c'est au contraire à ce moment que le logiciel prend réellement vie.
- C'est en effet à ce moment que le logiciel est utilisable et que la moindre anomalie ou la moindre panne doit être traitée.

42

II) Production et maintenance

- Classiquement avant de déployer un logiciel sur un ou sur plusieurs ordinateurs, il faut effectuer un test de déploiement.
- **La pré-production** consiste à déployer le logiciel sur des ordinateurs afin de réaliser un test grandeur nature de son utilisation.
- Bien souvent, la pré-production utilise exactement les mêmes ordinateurs que ceux sur lesquels le logiciel sera exécuté. Si cela n'est pas le cas, ces ordinateurs se doivent d'être très similaires.

43

II) Production et maintenance

- **Les béta-test** sont effectués lors de la pré-production. L'objectif est de fournir le logiciel à des utilisateurs / testeurs
  - Ceux-ci remontent les anomalies qu'ils détectent lors de l'utilisation qu'ils font du logiciel.
- **Le stress-test** est aussi effectué lors de la pré- production. L'objectif est de chercher les conditions limites d'exploitation du logiciel (en termes de nombre d'utilisateur ou de fréquence d'utilisation).

44

II) Production et maintenance

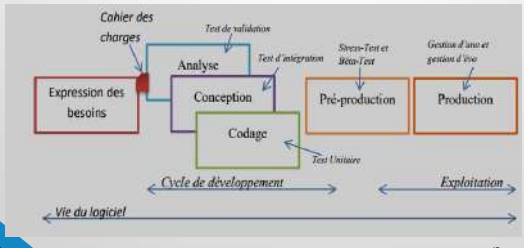
- Une fois la pré-production passée, le logiciel est alors mis en **production**. Lorsqu'il est en production un logiciel peut subir des pannes.
- Le rôle de la maintenance est d'assurer le traitement de ces pannes.
- Qui n'a pas vu son logiciel préféré planter sans que l'on sache trop pourquoi ? Tout bon utilisateur sait qu'il faut parfois redémarrer la machine afin de pouvoir à nouveau utiliser son logiciel.
- Dans le cas de logiciel complexe, s'exécutant sur plusieurs machines, les plantages arrivent aussi. Il faut alors relancer le logiciel en s'assurant qu'aucune donnée n'a été perdue. C'est là une des taches réalisée par la maintenance.

45

- La maintenance est à dissocier de la gestion d'évolution. En effet, la maintenance n'effectue aucune modification des services que rend le logiciel.
- En maintenance, il est possible de corriger des bugs du logiciel mais cela ne couvre pas l'extension du logiciel afin qu'il assure de nouvelles fonctionnalités.
- L'ajout de fonctionnalité est une évolution du logiciel.
- Lorsqu'une demande d'évolution est exprimée par un utilisateur, il faut alors reprendre le cycle de développement.
- L'évolution doit être analysée. Son impact sur la conception doit être définit. Puis enfin l'évolution doit être codée.
- On peut entendre les termes de **gestion d'ano** et de **gestion d'évo**. Les ano sont des anomalies. Les évo sont des évolutions.

46

- Le schéma ci-dessous représente tous les termes que nous venons de présenter.
- Il propose une vue très générale du génie logiciel. Cette vision générale permet alors de mieux comprendre la diversité du génie logiciel.



47

Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaoufb@yahoo.fr

1

Conduite de projet logiciel

2

Gestion de projet logiciel

3

Diagramme de Gantt ?

- Etabli par Henry Gantt en 1885
- Le diagramme de Gantt, est utilisé en gestion de projet,
- Pour représenter visuellement l'état d'avancement des différentes tâches qui constituent un projet.
- Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin.

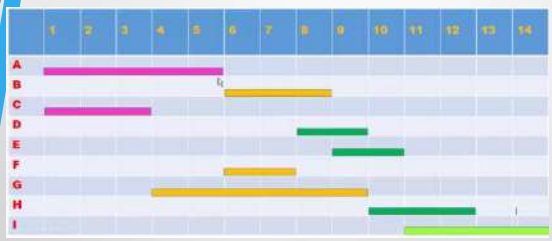
4

Exemple : Diagramme de Gantt

Tâche (x)	P(X)	S(X)	Durée en jours
A	----	B F	5
B	A	E	3
C	----	G	3
D	F	I	2
E	B	I	2
F	A	D	2
G	C	H	6
H	G	----	3
I	D E	----	4

5

Exemple : Diagramme de Gantt



6

Chemin critique

- Dans un projet, on désigne par "chemin critique" les tâches ou un ensemble de tâches qui ne peuvent pas être repoussées sans que la date de fin du projet ne soit repoussée.
- Une tâche figure sur le chemin critique
  - si une modification de sa date de début ou de sa durée génère une modification de la date de fin du projet.

7

Chemin critique

8

- **Remarques :**
- Le diagramme de GANTT sera modifié au fur et à mesure de l'avancement du projet.
- Il faut mettre à jour ce diagramme régulièrement. Le chemin critique peut évoluer en fonction de l'avancement, du retard, ou de toute modification sur une tâche.

9

- le diagramme de Gantt est le parfait outil pour communiquer sur l'avancement du projet.
- On utilise le diagramme **"PERT"** **(Program Evaluation and Review Technique)** pour ordonnancer le projet lorsque le nombres de tâches et les interdépendances deviennent conséquents.
- Interdépendances
  - Au cours d'un projet, les différentes tâches à réaliser ne sont pas indépendantes.
  - En fait, certaines ne pourront commencer qu'à l'achèvement des précédentes.
  - D'autres plus indépendantes pourront être réalisées en parallèle.

10

PERT

Program Evaluation and Review Technique

Technique d'Elaboration et de mise à jour de Programme

11

Qu'est-ce qu'un diagramme de PERT

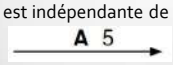
- Cette méthode a été mise au point en 1957 aux Etats-Unis, lors du développement du missile POLARIS.

- Ce projet mobilisait 250 fournisseurs principaux et environ 9000 sous-traitants.

Le délai initial prévu de 6 ans a pu être ramené à 2 années et demi.

12

Qu'est-ce qu'un diagramme de PERT

- I) Tâche
  - Une tâche fait évoluer l'œuvre vers son état final , elle consomme donc du temps , de l'énergie , de la matière et de ce fait coûte .
  - Chaque tâche est représentée par une flèche (segment orienté dans le sens de l'écoulement du temps) dont la longueur est indépendante de la durée de la tâche .
- Exemple 

- A = Nom de la tâche
- 5 = Durée de la tâche

13

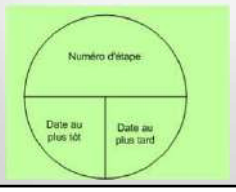
Qu'est-ce qu'un diagramme de PERT

- II) Tâche fictive
  - Tâche d'une durée nulle
  - Chaque tâche fictive est représentée par une flèche pointillée , sa durée est nulle, elle ne consomme aucune ressource , elle ne coûte donc rien .

14

Qu'est-ce qu'un diagramme de PERT

- III) Etape
  - On appelle étape le début ou la fin d'une tâche .
  - Une étape sera représentée par un cercle ou une autre figure géométrique .
  - Une étape est de durée nulle , elle ne coûte donc rien .



15

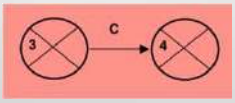
Qu'est-ce qu'un diagramme de PERT

- Le diagramme PERT propose de calculer, à l'aide d'un graphe, l'enchaînement optimal des tâches.
- Le réseau met en évidence les relations entre les tâches et les étapes .
- Chaque tâche est identifiée par sa durée moyenne et sa précédence.
- Un graphe de dépendances est utilisé. Pour chaque tâche, sont indiquées une date "au plus tôt" et une date « au plus tard ».
- Le diagramme permet de déterminer le chemin critique qui conditionne la durée minimale du projet.

16

Convention

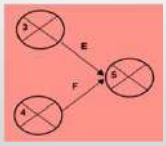
- Des tâches CONSECUTIVES sont des tâches qui se suivent.
- Des tâches ANTERIEURES sont des tâches qui , par rapport à une autre , doivent être réalisées avant .
- Toute tâche a **une étape** début pour origine et une étape fin pour extrémité
- La tâche C commence à l'étape 3 et se termine à l'étape 4



17

Convention

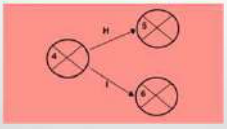
- Une étape ne peut être atteinte que lorsque les taches qui la précèdent sont toutes terminées.
- L'étape 5 ne sera atteinte que lorsque les tâches E et F seront terminées



18

Convention

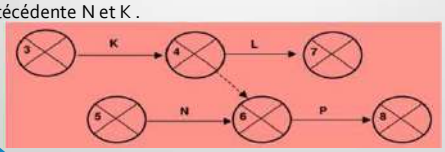
- Aucune tâche ne peut commencer tant que l'étape située a son origine n'est pas atteinte .
- Les tâches H et I ne pourront commencer que lorsque l'étape 4 sera atteinte .



19

Convention

- l'étape située à l'extrémité d'une fictive ne peut être atteinte qu'après l'étape située a son origine.
- Exemple :
- L'étape 6 ne peut être atteinte que : lorsque l'étape 4 est atteinte , et lorsque la tâche N est terminée .
- Lecture : La tâche L a pour antécédente K La tâche P a pour antécédente N et K .



20

Qu'est-ce qu'un diagramme de PERT

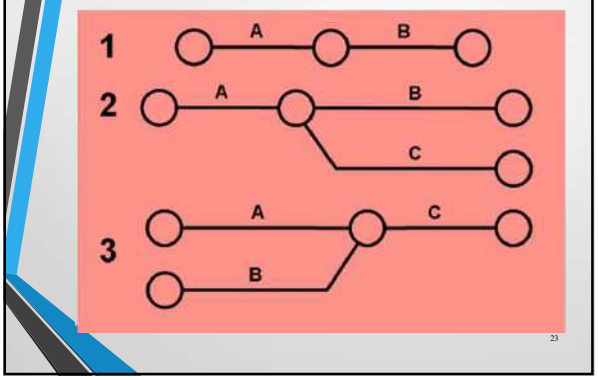
- Le but est de trouver la meilleure organisation possible pour qu'un projet soit terminé dans les meilleurs délais, et d'identifier les tâches critiques,
- c'est-à-dire les tâches qui ne doivent souffrir d'aucun retard sous peine de retarder l'ensemble du projet.

21

Exemple

- Représentez avec PERT, les données de chacune des assertions suivantes :
  - La tâche B ne peut commencer que lorsque la tâche A est entièrement terminée.
  - Les tâches B et C ne peuvent commencer que si la tâche A est terminée.
  - La tâche C ne peut commencer que lorsque les tâches A et B sont toutes deux terminées

22

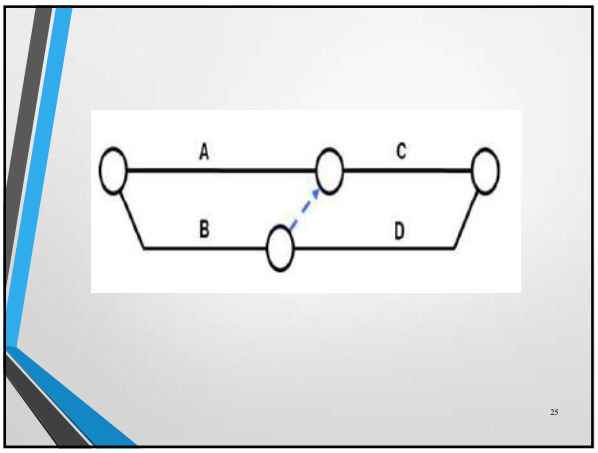


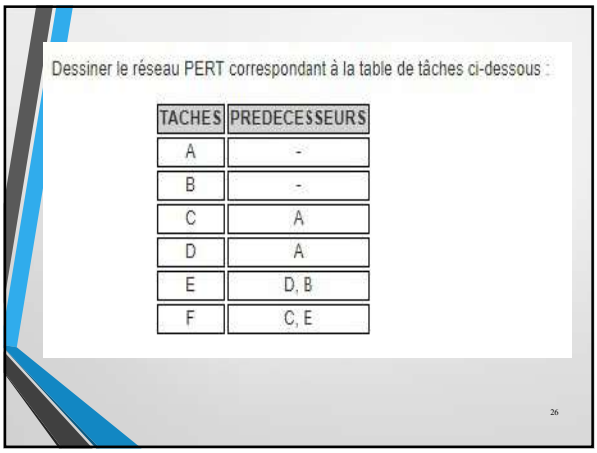
23

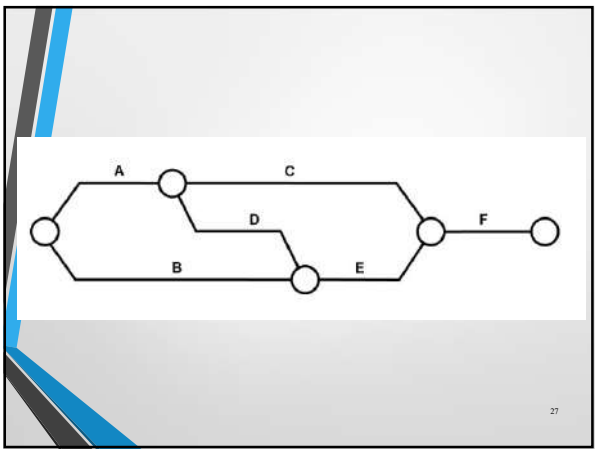
Dessiner le réseau PERT correspondant à la table de tâches ci-dessous :

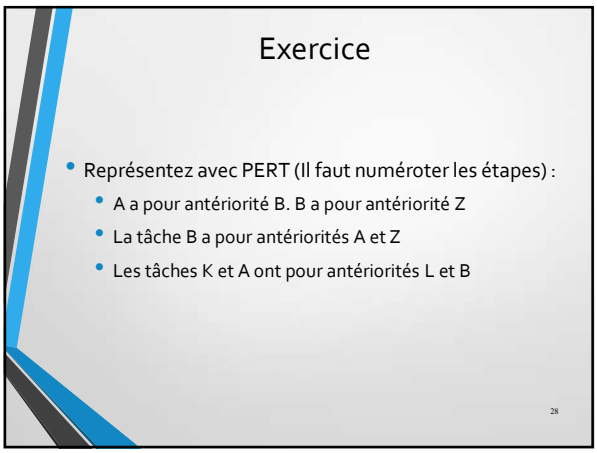
TACHES	PREDECESSEURS
A	-
B	-
C	A, B
D	B

24



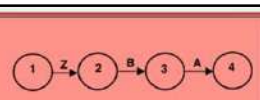






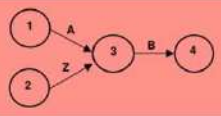
Exercice n°1 :

A a pour antériorité B
B a pour antériorité Z



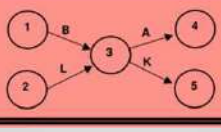
Exercice n°2 :

La tâche B a pour antériorités A et Z



Exercice n°3 :

Les tâches K et A ont pour antériorités L et B



29

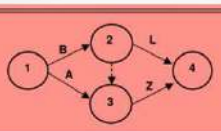
Exercice

- Représentez avec PERT (Il faut numéroter les étapes) :
  - La tâche Z a pour antériorités A et B. La tâche L a pour antériorité B
  - La tâche Z a pour antériorité A. La tâche L a pour antériorités A et B

30

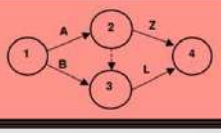
Exercice n°4 :

La tâche Z a pour antériorités A et B
La tâche L a pour antériorité B



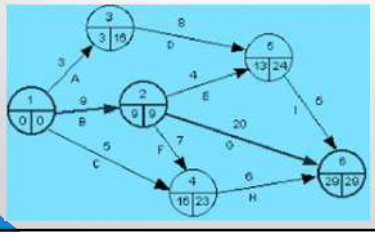
Exercice n°5 :

La tâche Z a pour antériorité A
La tâche L a pour antériorités A et B



31

Comment construire un réseau PERT ?



32

Les niveaux (les rangs)

- Sont du niveau 1 : Les tâches qui n'ont pas de tâches antérieures .
- Sont du niveau 2 : Les tâches qui ont pour antécédentes les tâches de rang 1 .
- Sont du niveau 3 : Les tâches qui ont pour antécédentes les tâches de rang 2 .

33

Construire un réseau PERT

- En 5 étapes on peut construire un réseau PERT
  - 1- Insérer une étape « Début » sur laquelle partent les tâches sans précédents ;
  - 2- Insérer une étape « Fin » sur laquelle arrivent les tâches sans suivants;
  - 3- Insérer les autres tâches du projet à l'aide du calendrier du suivant;
  - 4- Déterminer les dates de début au plutôt et les dates de début au plus tard;
  - 5- Déterminer le chemin critique.

34

Construire un réseau PERT

- En principe, un projet se compose de plusieurs tâches : A,B,C,D…
- Chaque tâche est associée à une durée de réalisation estimée

35

Exemple : pour comprendre

- Soit le projet suivant qui se compose de 5 tâches:

Tâche	Antériorité	Durée en jours
A	-	2
B	-	4
C	A	1
D	B	3
E	C D	2

36

Déterminer les niveaux (les rangs)

- Pour déterminer les niveaux de réalisation on a plusieurs méthodes, la plus simple méthode est la suivante:

Tâche	Antériorité	Durée en jours	Niveaux
A	-	2	0 tâche sans antériorité
B	-	4	0 tâche sans antériorité
C	A	1	1
D	B	3	1
E	C D	2	2

37

P(x) calendrier des précédents et S(x) calendrier des suivants

- On note P(x) le calendrier des précédents (les antériorité connues déjà avec le nombre des tâches composant le projet et leurs durées

Tâche	P(x)	S(x)	Durée en jours
A	-	C (on fixe la ligne ou elle existe A)	2
B	-	D (on fixe la ligne ou elle existe B)	4
C	A	E (on fixe la ligne ou elle existe C)	1
D	B	E (on fixe la ligne ou elle existe D)	3
E	C D	- (sans suivant) elle n'existe pas dans p(x)	2

La tâche FIN

38

On obtient le tableau suivant

Tâche	P(X)	S(X)	Niveaux	Durée en jours
A	-	C	0	2
B	-	D	0	4
C	A	E	1	1
D	B	E	1	3
E	C D	-	2	2

39

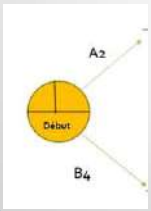
Réaliser le graphe PERT

- 1- Insérer une étape « Début » sur laquelle partent les tâches sans précédents ;
- 2- Insérer une étape « Fin » sur laquelle arrivent les tâches sans suivants;
- 3- Insérer les autres tâches du projet à l'aide du calendrier du suivant;
- 4- Déterminer les dates de début au plutôt et les dates de début au plus tard;
- 5- Déterminer le chemin critique.

40

Réaliser le graphe PERT

- 1- Insérer une étape « Début » sur laquelle partent les tâches sans précédents ;



Tâche	P(X)	S(X)	Niveaux	Durée en jours
A	-	C	0	2
B	-	D	0	4
C	A	E	1	1
D	B	E	1	3
E	CD	-	2	2

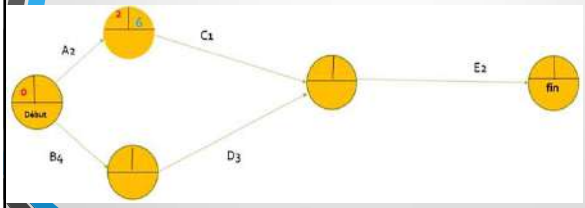
41

- 1- Insérer une étape « Début » sur laquelle partent les tâches sans précédents ;
- **2- Insérer une étape « Fin » sur laquelle arrivent les tâches sans suivants;**

Tâche	P(X)	S(X)	Niveaux	Durée en jours
A	-	C	0	2
B	-	D	0	4
C	A	E	1	1
D	B	E	1	3
E	CD	-	2	2

42

- 1- Insérer une étape « Début » sur laquelle partent les tâches sans précédents ;
- 2- Insérer une étape « Fin » sur laquelle arrivent les tâches sans suivants;
- **3- Insérer les autres tâches du projet à l'aide du calendrier du suivant;**

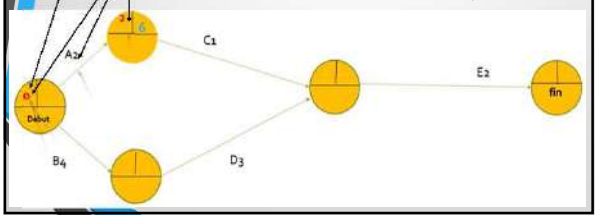


43

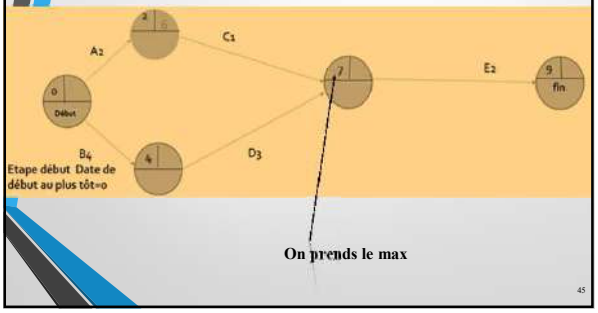
4- Déterminer les dates de début au plutôt et les dates de début au plus tard;

Une règle : l'étape début commence à la date 0

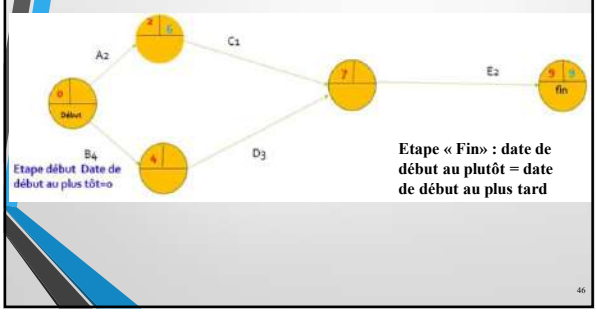
(0+2) = 2



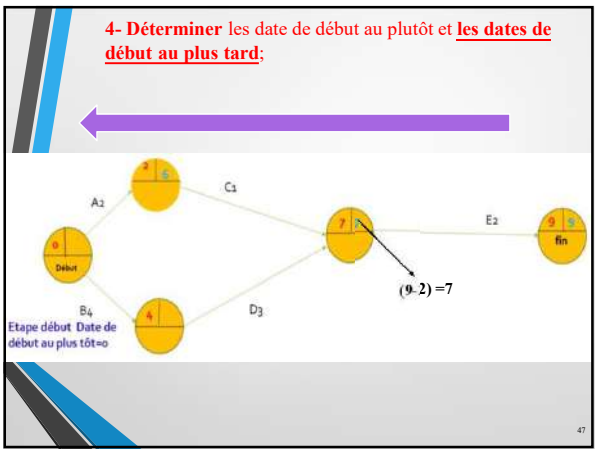
4- Déterminer les date de début au plutôt et les dates de début au plus tard;



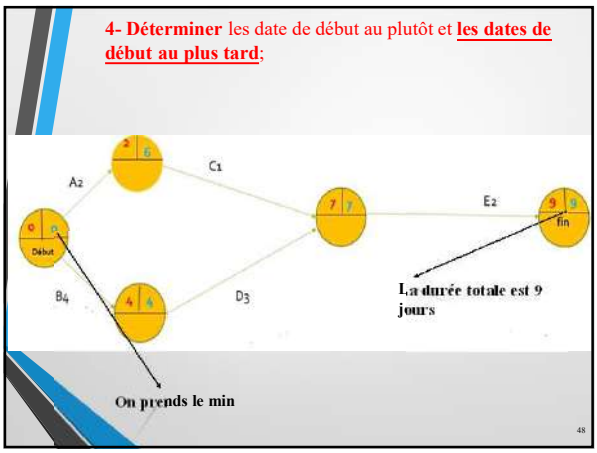
4- Déterminer les date de début au plutôt et **les dates de début au plus tard**;



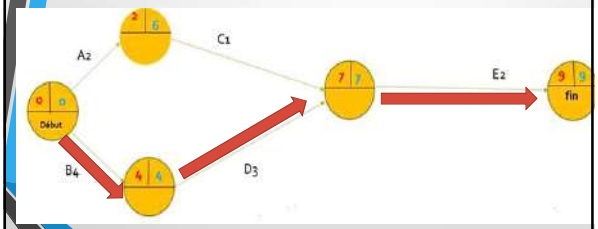
4- Déterminer les date de début au plutôt et **les dates de début au plus tard**;



4- Déterminer les date de début au plutôt et **les dates de début au plus tard**;



- 5- Déterminer le chemin critique.
  - Une tâche est critique si sa de début au plutôt = sa date de début au plus tard
  - Elle ne supporte aucun retard
  - L'ensemble des tâches critique compose le chemin critique



49

Calcul des marges

- Pour tracer le diagramme, il est souhaitable que les flèches ne se croisent pas.
- Pour déterminer la date au plus tôt d'une tâche, il faut parcourir le diagramme de gauche à droite et calculer le temps du plus long des chemins menant du début du projet à cette tâche.
- S'il y a plusieurs sous-chemins, on effectue le même calcul pour chacun et on choisit la date la plus grande.

50

Calcul des marges

- Pour déterminer la date au plus tard d'une tâche, il faut parcourir le diagramme de droite à gauche, et soustraire de la date au plus tard de la tâche suivante la durée de la tâche dont on calcule la date au plus tard.
- S'il y a plusieurs sous-chemins, on effectue le même calcul pour chacun et on choisit la date la plus petite.

51

- On appelle **"marge"** d'une tâche le retard qu'il est possible de tolérer dans la réalisation de celle-ci, sans que la durée optimale prévue du projet global en soit affectée.
- Il est possible de calculer trois types de marges : la marge totale, la marge certaine et la marge libre.
- La marge totale d'une tâche indique le retard maximal que l'on peut admettre dans sa réalisation (sous réserve qu'elle ait commencé à sa date au plus tôt) sans allonger la durée optimale du projet.
- Elle se calcule en retirant la durée de la tâche en question à l'écart qu'il peut y avoir entre sa date de au plus tôt de début et sa date au plus tard de fin :

Marge totale tâche "ij" = Date au plus tard "étape j" - Date au plus tôt "étape i" - Durée tâche "ij"

52

- La marge libre d'une tâche indique le retard que l'on peut admettre dans sa réalisation (sous réserve qu'elle ait commencé à sa date au plus tôt) sans modifier les dates au plus tôt des tâches suivantes et sans allonger la durée optimale du projet.
- Elle se calcule en retirant la durée de la tâche en question à l'écart qu'il peut y avoir entre ses dates au plus tôt de début et de fin :
- **Marge libre tâche "ij" = Date au plus tôt "étape j" - Date au plus tôt "étape i" - Durée tâche "ij"**
- Un retard correspondant à la marge libre d'une tâche reste sans conséquence sur les marges des tâches qui lui succèdent. Il est donc possible de cumuler des retards, s'inscrivant dans leur marge libre, pour plusieurs tâches successives, sans remettre en cause la durée optimale prévue pour le projet.

53

Exercice : Soit le projet suivant

Tâche (X)	P(X)	Durée en jours
A	—	6
B	—	2
C	A	3
D	C	4
E	A	1
F	C E	7
G	A B	2
H	G	3
I	D F	4

Question

- Dessiner un réseau PERT
- Déterminer le chemin critique
- Calculer toutes les marges totales et libres
- Interpréter le graphe obtenu

55

Synthèse

- La marge libre d'une tâche est la durée dont on peut décaler sa date de fin sans retarder la date de début au plus tôt des tâches suivantes.
- La marge totale d'une tâche est la durée dont on peut décaler sa date de fin sans retarder la date de fin du projet.
- L'inégalité marge totale > marge libre est toujours vérifiée.
- Les tâches à marge nulle sont appelées critiques.
- La suite des tâches critiques est nommée "chemin critique"

56

Synthèse

- La marge libre d'une tâche est la durée dont on peut décaler sa date de fin sans retarder la date de début au plus tôt des tâches suivantes.
- La marge totale d'une tâche est la durée dont on peut décaler sa date de fin sans retarder la date de fin du projet.
- L'inégalité marge totale > marge libre est toujours vérifiée.
- Les tâches à marge nulle sont appelées critiques.
- La suite des tâches critiques est nommée "chemin critique"

57

Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaoufb@yahoo.fr

1

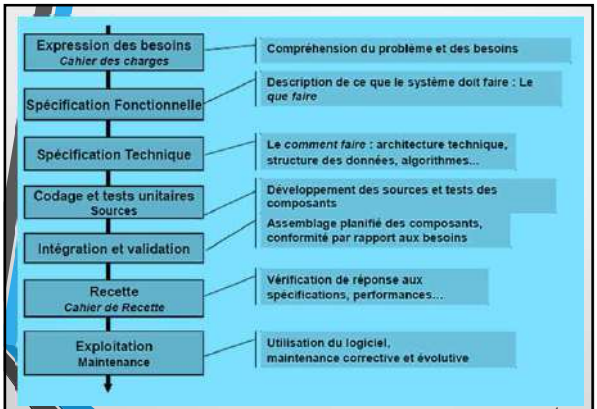
Méthodes de développement logiciel

2

Introduction

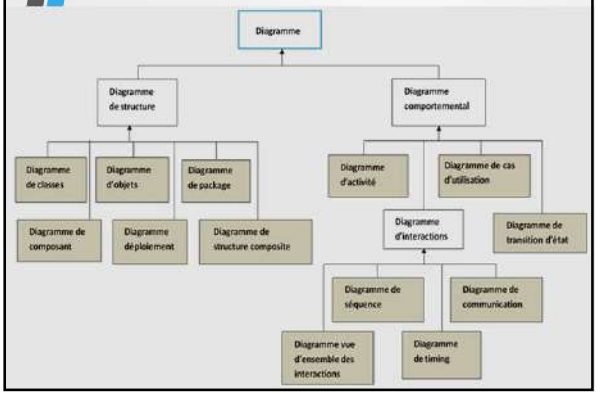
- **Le Génie Logiciel**
- Ensemble de moyens mis en œuvre pour la construction de logiciels.
- Visant le développement et l'évolution de systèmes logiciels de grande taille et de haute qualité en respectant les contraintes de coûts et de temps en gérant les différents risques

3



4

UML - vue statique- vue dynamique-



Définition d'un Processus

- Un processus ou une démarche, est dans la majorité des cas, un ensemble d'activités inter reliées et menées dans le but de définir un produit.
- Un processus ou une démarche est un ensemble cohérent d'activités permettant la construction d'un SI.
- Il représente l'ensemble des étapes permettant de spécifier
  - • Les décisions à prendre
  - • Comment les prendre
  - • Dans quel ordre

6

Définition d'un Processus

- La partie produit est la cible désirée d'un développement de SI.
- La partie processus (aussi appelée « partie démarche ») est la route à suivre pour atteindre ce but.
- Plus le projet est grand plus le processus de construction devient complexe et se doit d'être :
  - **clair et précis pour une utilisation optimale de la modélisation.**

7

Présentation d'UP

- UP (Unified Process, **« processus unifié »)** est un processus de développement logiciel itératif, centré sur l'architecture, piloté par des cas d'utilisation et orienté vers la diminution des risques
- C'est un processus pouvant être adaptée à une large classe de systèmes logiciels, à différents domaines d'application, à différents types d'entreprises, à différents niveaux de compétences et à différentes tailles de l'entreprise.

8

Caractéristiques du processus unifié

- Le processus unifié est à base de composants,
- Le processus unifié utilise le langage UML (ensemble d'outils et de diagramme),
- Le processus unifié est piloté par les cas d'utilisation,
- Centré sur l'architecture,
- Itératif et incrémental

9

UP est piloté par les cas d'utilisation d'UML

- Le but principal d'un système informatique est de satisfaire les besoins du client.
- Le processus de développement sera donc accès sur l'utilisateur.
- Les cas d'utilisation permettent d'illustrer ces besoins. Ils détectent puis décrivent les besoins fonctionnels (du point de vue de l'utilisateur),
- S'il est vrai que les cas d'utilisation guident le processus de développement, ils ne sont pas sélectionnés de façon isolée, mais doivent absolument être développés "en tandem" avec l'architecture du système.

10

UP est itératif

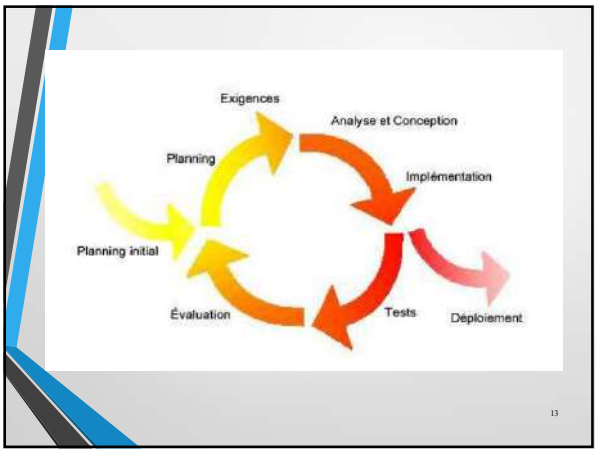
- Le développement d'un logiciel peut s'étendre sur plusieurs mois. Tout le logiciel ne sera pas développer d'un coup.
- On peut découper le travail en plusieurs parties qui sont autant de mini projets. Chacun d'entre eux représentant une itération qui donne lieu à un incrément.
- Une itération désigne la succession des étapes de l'enchaînement d'activités,
- tandis qu'un incrément correspond à une avancée dans les différents stades de développement.

11

UP est itératif

- A chaque itération, les développeurs identifient et spécifient les cas d'utilisations pertinents,
- créent une conception en se laissant guider par l'architecture choisie,
- implémentent cette conception sous forme de composants et vérifient que ceux ci sont conformes aux cas d'utilisation.
- Dés qu'une itération répond aux objectifs fixés le développement passe à l'itération suivante.

12



13

Avantages d'un processus itératif

- Permet de limiter les retards, les coûts, en termes de risques, aux dépenses liées à une itération.
- Identification des problèmes dès les premiers stades de développement et non pas en phase de test comme avec l'approche « classique »).
- Permet d'accélérer le rythme de développement grâce à des objectifs clairs et à court terme.
- Permet de prendre en compte le fait que les besoins des utilisateurs et les exigences correspondantes ne peuvent être intégralement définis à l'avance et se dégagent peu à peu des itérations successives

14

UP est centré sur l'architecture

- Dès le démarrage du processus, on aura une vue sur l'architecture à mettre en place.
- L'architecture d'un système logiciel peut être décrite comme les différentes vues du système
- L'architecture logicielle équivaut aux aspects statiques et dynamiques les plus significatifs du système.
- L'architecture émerge des besoins de l'entreprise, tels qu'ils sont exprimés par les utilisateurs et autres intervenants et tels qu'ils sont reflétés par les cas d'utilisation.

15

UP est centré sur l'architecture

- Les cas d'utilisation doivent une fois réalisés, trouver leur place dans l'architecture.
- L'architecture doit prévoir la réalisation de tous les cas d'utilisation.
- L'architecture et les cas d'utilisation doivent évoluer de façon concomitante.
- L'architecture se dévoile peu à peu, au rythme de la spécification et de la maturation des cas d'utilisation, qui favorisent, à leur tour, le développement d'un nombre croissant de cas d'utilisation.

16

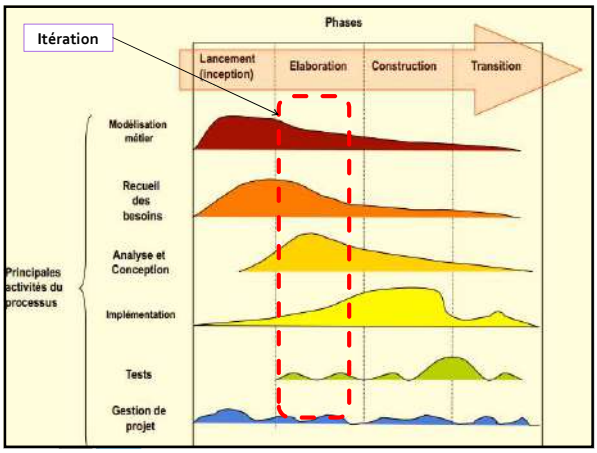
Cycle de vie du processus unifié

- Le processus unifié répète un certain nombre de fois une série de cycles.
- Tout cycle se conclut par la livraison d'une version du produit aux clients et s'articule en
- 4 phases et Chaque phase se subdivise à son tour en itérations.
- Un groupe de 4 phases = un cycle
- Les 4 phases qui composent le processus UP

1-Création 3-Construction

2- Elaboration 4-Transition

17



Cycle de vie du processus unifié

- Chaque **cycle** (Création, Construction, Elaboration, Transition) se traduit par une nouvelle version du système.
- Ce produit se compose d'un corps de code source réparti sur plusieurs composants pouvant être compilés et exécutés
- Une **itération** comporte les activités suivantes :
  - Expression des besoins
  - Analyse
  - Conception
  - Implémentation
  - Test

19

Les phases du processus UP

- **Analyse des besoins**
- Cette phase Comprend :
- La définition des objectifs pour établir les limites du projet
- La réduction des risques majeurs
- L'identification des cas d'utilisation principaux
- L'identification d'une architecture provisoire
- L'étude de rentabilité

La planification de la phase d'élaboration

20

Les phases du processus UP

- **Elaboration**
- Permet de définir et de construire l'architecture de base du système de manière stable.
- Identifie et décrit la majorité des besoins utilisateurs en précisant la plupart des cas d'utilisation.
- Précise l'étude commerciale puisqu'à l'issue de cette phase, le chef de projet doit être en mesure de prévoir les activités et d'estimer les ressources nécessaires à l'achèvement du projet.

21

Les phases du processus UP

- **Construction**
- Consiste à concevoir et implanter l'ensemble des éléments opérationnel.
- Développement du logiciel exécutable.
- L'architecture de référence devient un produit complet.
- Le produit contient tous les cas d'utilisation que les chefs de projet ont décidé de mettre au point pour cette version.
- La première version peut encore avoir des anomalies qui peuvent être en partie résolues lors de la phase de transition.

22

Les phases du processus UP

- **Transition**
- Permet de faire passer l'application des développeurs aux utilisateurs finaux.
- C'est la mise en production du logiciel, avec la conversion des données, la formation des utilisateurs, le déploiement, les beta-tests.
- Un groupe d'utilisateurs essaye le produit et détecte les anomalies et défauts.
- Peut nécessiter la mise en œuvre d'un service d'assistance ou la correction des anomalies. détectées (ou le report à la version suivante).

23

Les activités du processus UP

- **Expression des besoins**
- L'expression des besoins comme son nom l'indique, permet de définir les différents besoins :
- Recenser les besoins fonctionnels (du point de vue de l'utilisateur) qui conduisent à l'élaboration des modèles de cas d'utilisation
- Le modèle de cas d'utilisation présente le système du point de vue de l'utilisateur et représente sous forme de cas d'utilisation et d'acteurs, les besoins du client.

24

Les activités du processus UP

- **Analyse**
- L'objectif de l'analyse est d'accéder à une compréhension des besoins et des exigences du client.
- Il s'agit de livrer des spécifications pour permettre de choisir la conception de la solution.
- Un modèle d'analyse livre une spécification complète des besoins issus des cas d'utilisation et les structure sous une forme qui facilite la compréhension (scénarios), la préparation (définition de l'architecture), la modification et la maintenance du futur système.

25

Les activités du processus UP

- **La conception**
- permet d'acquérir une compréhension approfondie des contraintes liées au langage de programmation, à l'utilisation des composants et au système d'exploitation.
- Elle détermine les principales interfaces et les transcrit à l'aide d'une notation commune.
- Elle constitue un point de départ à l'implémentation :
- elle décompose le travail d'implémentation en sous-système

elle créée une abstraction transparente de l'implémentation

26

Les activités du processus UP

- **Implémentation**
- L'implémentation est le résultat de la conception pour implémenter le système sous formes de composants,
- Les objectifs principaux de l'implémentation sont de planifier les intégrations des composants pour chaque itération, et de produire les classes et les sous-systèmes sous formes de codes sources.

27

Les activités du processus UP

- **Test**
- Les tests permettent de vérifier des résultats de l'implémentation en testant la construction.
- Pour mener à bien ces tests, il faut les planifier pour chaque itération, les implémenter en créant des cas de tests, effectuer ces tests et prendre en compte le résultat de chacun.

28

Avantages d'UP

- Permet d'accélérer le rythme de développement grâce à des objectifs clairs à court terme
- Les éléments sont intégrés progressivement et non pas en fin de cycle
- Permet de limiter les risques de retard car : identification des problèmes dès les premiers stades de développement => les premières itérations sont déroulées et vous examinez tous les composants du processus au fur et à mesure
- Peut permettre de fournir rapidement un produit avec des fonctionnalités réduites
- L'itération favorise la réutilisation du code

29

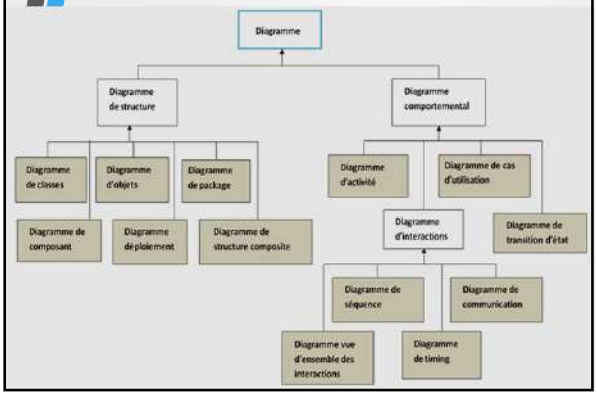
Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaouf@yahoo.fr

30

UML - vue statique- vue dynamique-

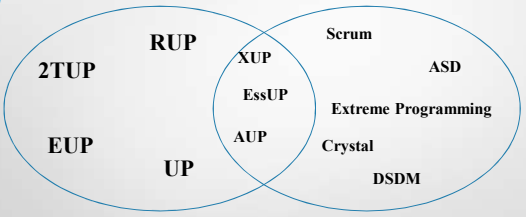


Introduction : Processus Unifié

- Plusieurs processus unifiés, pas un seul : il y a tellement de systèmes, de techniques variés qu'il serait impensable d'envisager un processus qui soit adapté à tous les projets possibles,
- Autrement dit que le développement avec ce processus réponde bien aux objectifs tout en respectant les contraintes
- Piloté par les risques : on a vu qu'ils sont nombreux dans le développement logiciel

32

Introduction



Introduction : Processus Unifié

- Incrémental : définir des incréments de réalisation est en effet la meilleure pratique de gestion des risques d'ordre à la fois technique et fonctionnel.
- Chaque incrément confirme la preuve de faisabilité auprès de l'équipe de développement et du client.
- De plus, le suivi des incréments constitue un excellent contrôle des couts et délais
- Itératif : non seulement à chaque cycle on ajoute une fonctionnalité mais de plus on améliore les fonctionnalités précédentes

34

Introduction : Processus Unifié

- On peut citer par exemple: inadéquation aux besoins des utilisateurs, le non respect des couts et délais
- Orienté composant : Un composant est un module indépendant, qui pourrait servir pour d'autres projets.
- Le découpage en modules de ce type de processus se fait aussi bien en modélisation qu'en production, et permet la réutilisation logicielle.
- Orienté utilisateur : Les utilisateurs sont à l'origine du développement

35

RUP (Rational Unified Process)

36

Qu'est ce que « RUP » ?

- RUP
  - Rational Unified Process
  - Dérivée de UP (UP a été crée en 1996)
  - Commercialisée par IBM en 1998
- Méthode de développement logiciel
  - Itérative
  - Incrémentale
  - Pilotée par les cas d'utilisation.
  - Centrée sur l'architecture et la réduction des risques
  - Produit de qualité.

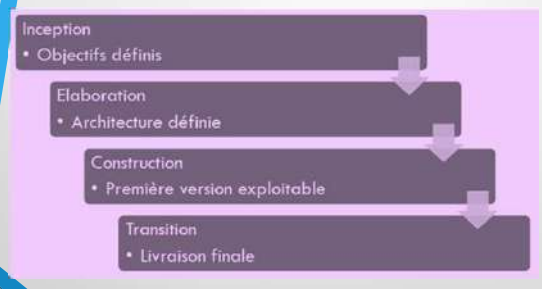
37

Principes de RUP

- Pilotée par les cas d'utilisation
- Construction d'un système à base de composants
- Adaptable aux changements (Agilité)
- Gestion des risques
- Livraison de qualité
- Concentrée sur le code exécutable
- Travail en équipe
- Gestion de projet

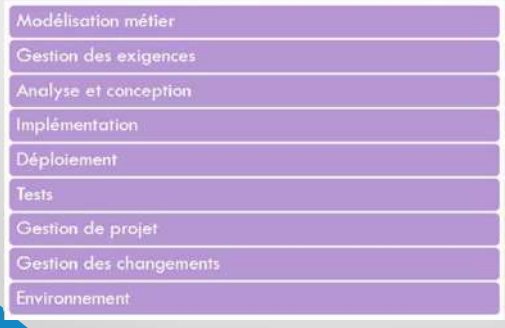
38

Phases de RUP

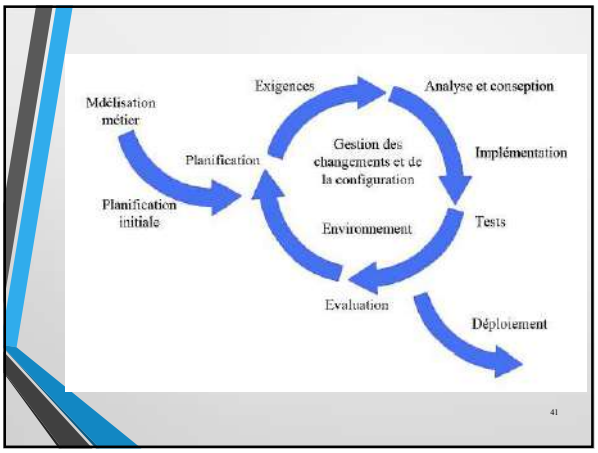


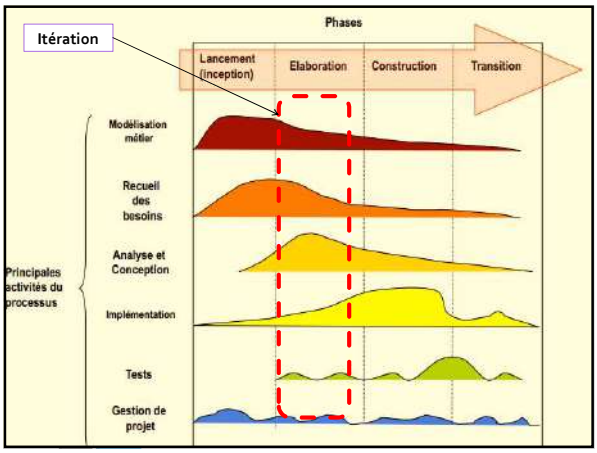
39

Activités de RUP



40





Avantages de la méthode RUP

- Traçabilité à partir des Uses Cases jusqu'au déploiement
- Approche basée sur l'architecture
- Gestion des risques dans les projets
- Réutilisation

43

2TUP (2 Track Unified Process)

44

Présentation de TTUP

- Processus créé par Valtech
  - un groupe français de conseil en technologies, e-business, management, et de formation professionnelle.
  - C'est un grand groupe (1100 employés) présent à l'international
- Pourquoi TTUP ?
  - Faire face aux contraintes de changement continuel imposées aux SI des entreprises

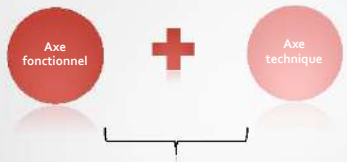
45

Présentation de TTUP

- Idée de base du TTUP : toute évolution imposée au SI peut se décomposer et se traiter parallèlement, suivant 2 axes (« 2 tracks ») :
  - Un axe fonctionnel
  - Un axe technique
- La réalisation du système consiste à fusionner les résultats des deux branches

46

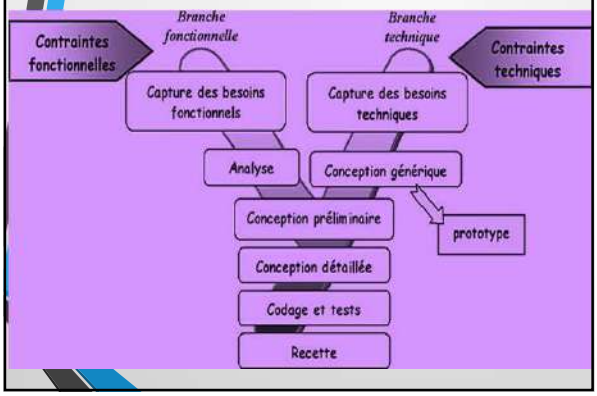
Présentation de TTUP



La réalisation du système consiste à fusionner les résultats des deux branches

47

Présentation de TTUP



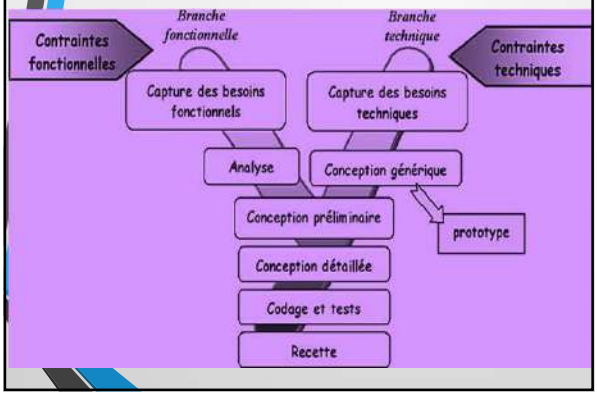
Présentation de TTUP

- **Une forme en Y**
- **I) Du coté de la branche fonctionnelle :**

- Capture des besoins fonctionnels : elle permet de déterminer un modèle des besoins focalisé sur le métier des utilisateurs.
  - Elle minimise le risque de produire un système inadéquat avec les besoins des utilisateurs.
  - L'objectif est de valider les spécifications et en vérifie la cohérence et l'exhaustivité.
- Analyse : étude des spécifications afin de savoir ce que le système va réellement réaliser en termes de métier. Découpage en composants.

49

Présentation de TTUP



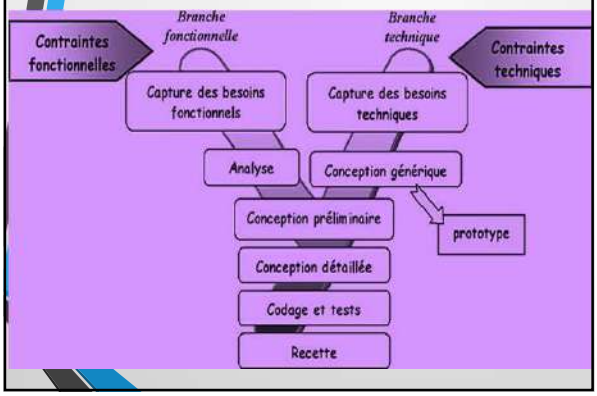
Présentation de TTUP

Du coté de la branche technique :

- Capture des besoins techniques : recensement des outils, des matériels et des technologies à utiliser ; des contraintes (temps de réponse maximal, contraintes d'intégration avec l'existant)
  - Permet de réaliser une première conception de l'architecture technique
- Conception générique : Découpage en composants nécessaires à la construction de l'architecture technique.
  - Il est généralement conseillé de réaliser un prototype pour assurer la validité de l'architecture.

51

Présentation de TTUP



Présentation de TTUP

Enfin, la branche du milieu :

-Conception préliminaire : étape délicate durant laquelle on intègre le modèle d'analyse dans l'architecture technique. Le but est d'associer les besoins fonctionnels aux composants.

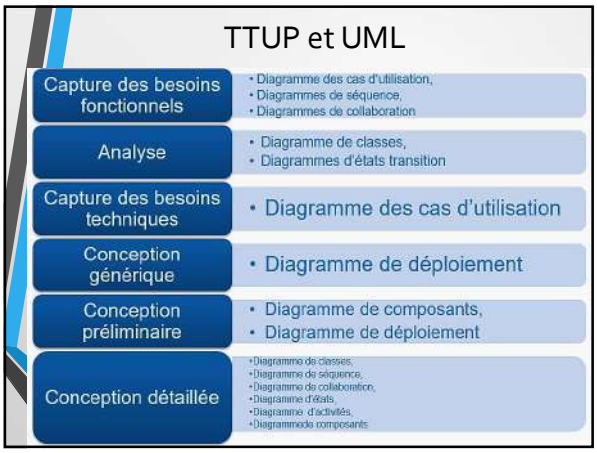
-Conception détaillée : conception de chaque fonctionnalité

-Etape de codage : phase de programmation de ces fonctionnalités, avec des tests au fur et à mesure

-Etape de recette : phase de validation des fonctions du système développé

53

TTUP et UML



Avantages d'utilisation d'une méthode de développement

- Permet de maîtriser le chemin (Point de départ→ Point d'arrivée)
- Gestion des risques : prise en charge de différents axes du projet
- Pratiques agiles : itératif, incrémental axée sur le développement
- Management de projet : découpage permet une meilleure gestion

55

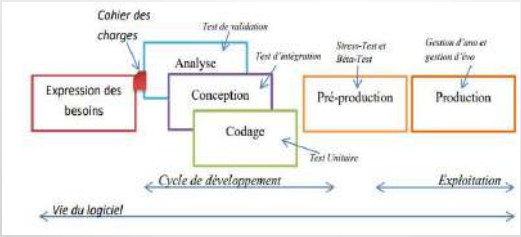
Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaoufb@yahoo.fr

1

Rappel



2

Méthodes de développement logiciel UP, RUP, TTUP

3

Modélisation

- Un modèle est la simplification/abstraction de la réalité
- Nous construisons donc des modèles afin de mieux comprendre les systèmes que nous développons
- Nous modélisons des systèmes complexes parce que nous sommes incapables de les comprendre dans leur totalité
- Le code ne permet pas de simplifier/abstraire la réalité

4

Des méthodes de modélisation

- L'apparition du paradigme objet à permis la naissance de plusieurs méthodes de modélisation
  - OMT, OOSE, Booch, Fusion, …
- Chacune de ces méthodes fournie une notation graphique et des règles pour élaborer les modèles
- Certaines méthodes sont outillées

5

Des méthodes de modélisation

- Entre 89 et 94 : le nombre de méthodes orientées objet est passé de 10 à plus de 50
- Toutes les méthodes avaient pourtant d'énormes points communs (objets, méthode, paramètres, …)
- Au milieu des années 90, G. Booch, I. Jacobson et J. Rumbaugh ont chacun commencé à adopter les idées des autres.
- Les 3 auteurs ont souhaité créer un langage de modélisation unifié

6

UML

- UML est le langage de modélisation orienté objet le plus connu et le plus utilisé au monde
- UML s'applique à plusieurs domaines
- UML n'est pas une méthode
  - RUP

7

UML - vue statique- vue dynamique-

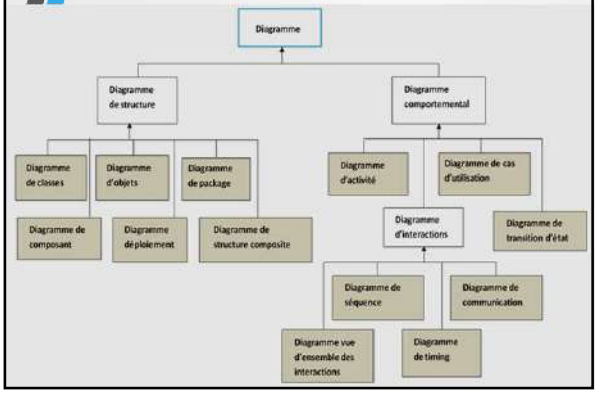


Diagramme de communication

- Un diagramme de collaboration représente la vue statique et la vue dynamique d'un ensemble d'élément
- Une collaboration définit des rôles (et non pas des classes!)

9

Diagramme de collaboration

- Le diagramme de collaboration fait partie des vues dynamiques du système d'information.
- **Origine**
- Les diagrammes de collaboration proviennent d'une adaptation des diagrammes d'objet de Booch (object diagram), des graphes d'interaction d'objet de Fusion (object interaction graph) et d'autres sources.

10

Diagramme de collaboration

- Un diagramme de collaboration représente la vue dynamique d'un ensemble d'élément
- Une collaboration définit des rôles
- Libre de placer les participants (objets).
- On fait des liens entre eux, et on les numérote (interactions)
- Pas de ligne de vie
- Objectifs :
  - Comportement collectif d'objets
  - En vue de réaliser une opération

11

Diagramme de collaboration

- **Définition**
- Ce type de diagramme montre les interactions et les liens entre objets (instances de classes et acteurs).
- Il s'intéresse à la structure de collaboration entre objets (séquencement, itération, concurrence, etc.).
- Il permet de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent.
- Les diagrammes de collaboration montrent les interactions entre les objets à travers la représentation chronologique d'envois de messages, mais le temps n'est pas représenté implicitement.
- La chronologie des interactions est indiquée par la numérotation de messages pour indiquer leur ordre d'envoi[2]

Diagramme de collaboration

- Ce diagramme est équivalent au diagramme de séquences.
- Par contre, l'aspect temporel n'apparaît pas, mais l'aspect chronologique est présent.
- Le diagramme de collaboration représente ces éléments réalisant un objectif particulier (une fonctionnalité du système).
- Il peut être utilisé pour illustrer l'exécution d'une opération ou d'un cas d'utilisation, ou simplement un scénario d'interactions dans le système.
- Il permet de concevoir un exemple d'interactions entre objets.

13

Eléments du Diagramme

- Les diagrammes de collaboration comportent :
- Les objets (instances de classe)
- Les liens entre objets (comme dans le diagramme de classe)
- Les interaction entre objets sous forme d'une suite de messages
- Les acteurs

14

OBJETS :

:Objet

- Représenté par un rectangle

- Nommage :
  - Nom de l'objet instancié

LenovoR500

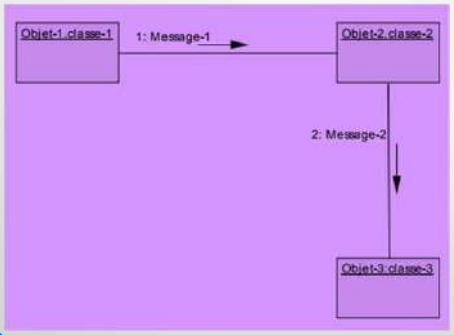
LenovoR500:Ordinateur

- Nom de l'objet et nom de la classe

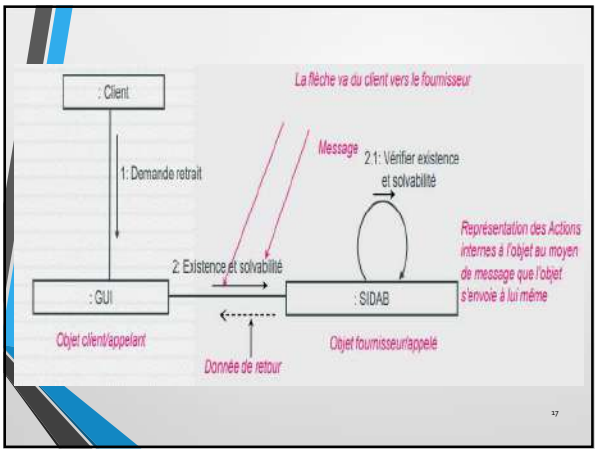
:Ordinateur

- Nom de la classe

15



16



Notion : Objet

- Un objet est une instance d'une classe.
- Il peut-être persistant ou transitoire : un objet persistant est un objet qui continue d'exister une fois que le processus qui l'a créé est terminé, un objet transitoire est un objet qui cesse d'exister une fois que le processus qui l'a créé est terminé.
- Les objets sont obtenus à partir des classes.
- Chaque objet est décrit par : Nom-objet :classificateur ou nom-objet :nom-classe.

18

Notion : Lien et Acteur

- **Lien**
- Un lien entre objets représente une connexion entre deux objets, il permet de mettre en exergue la collaboration entre objets, d'où le nom de diagramme de collaboration.
- Il est représenté sous forme d'un trait plein entre :
  - Deux objets
  - Un objet et un acteur (ou vice-versa)

19

Notion : Lien et Acteur

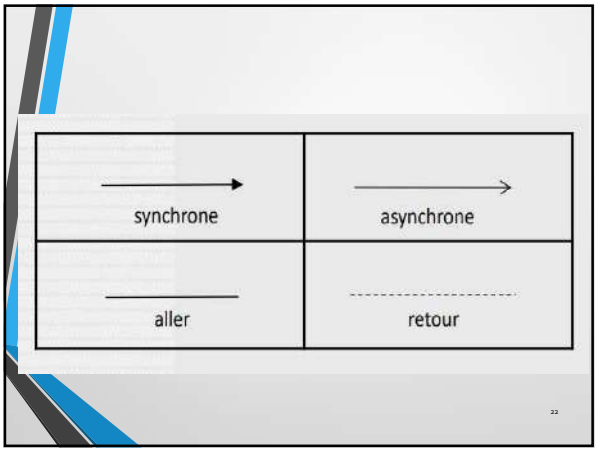
- **Acteur**
- La définition d'un acteur dans le contexte d'un diagramme de collaboration est la même que celle en vigueur dans les diagrammes de cas d'utilisation ou de séquence ;
- elle spécifie un utilisateur externe, ou un ensemble d'utilisateurs liés qui interagissent avec un système.

20

Notion : Message, Synchronisation des messages

- Synchrone : Un message est envoyé par à un objet à un autre, et le premier objet attends jusqu'à ce que l'action ai finie.
- Asynchrone : Un message est envoyé par à un objet à un autre, mais le premier objet n'attends pas la fin de l'action.
- Aller (plat) : Chaque flèche représente une progression d'une étape à une autre dans la séquence. La plupart asynchrone.
- Retour : Le retour explicite d'un objet à qui le message était envoyé.

21



22

Notion : Message, Synchronisation des messages

MESSAGES :

- Messages envoyés en même temps :
  Utilisation de la notation Chiffre-Lettre
- Messages invoqués plusieurs fois :
  *Contrainte de boucle * [i=0..9]*
- Messages envoyés à une condition :
  Ajout d'une expression logique [condition = true]
- Messages envoyés à lui même :

MESSAGE D'UN DIAGRAMME DE COLLABORATION

Notation

[Prédécesseurs/] [garde] [séquence :] [valeurRetour :=] signatureMessage

24

MESSAGE D'UN DIAGRAMME DE COLLABORATION

- ***Prédécesseurs*** : liste de numéros de séquence de messages séparés par une virgule.
- ***Garde :*** est une condition sous forme d'expression booléenne entre crochets.
- **séquence** : est le numéro de séquence du message
- **valeurRetour**
- **signatureMessage**

25

Notion : Message, Synchronisation des messages

- Dans un diagramme de collaboration il est possible de spécifier de manière très précise l'ordre et les conditions d'envoie des messages.
- Pour chaque message, il est possible d'indiquer : son numéro d'ordre qui indique le rang du message, c'est-à-dire son numéro d'ordre par rapport aux autres messages.
- Les messages sont numérotés à l'aide de chiffres séparés par des points. Ainsi, il est possible de représenter le niveau d'emboîtement des messages et leur précédence.

26

Notion : Message, Synchronisation des messages

- Exemple :
- l'envoi du message 1.3.5 suit immédiatement celui du message 1.3.4 et ces deux messages font partie du flot (de la famille de messages) 1.3.
- Pour représenter l'envoi simultané de deux messages, il suffit de les indexer par une lettre. Exemple : l'envoi des messages 1.3.a et 1.3.b est simultané.

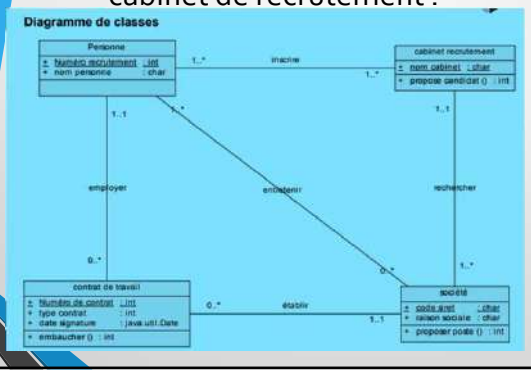
27

Notion : Message, Synchronisation des messages

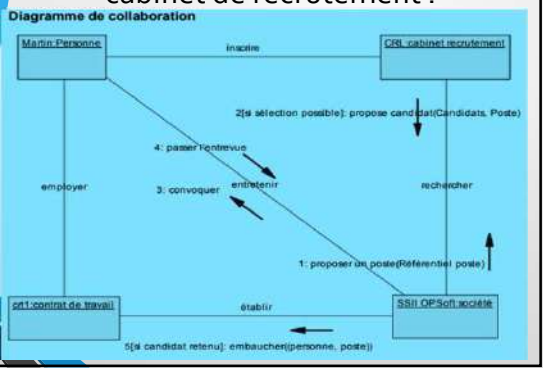
- la liste des messages prédécesseurs, composée d'une liste de numéros d'ordre suivie de '/ ', la liste des prédécesseurs définit quels messages doivent être échangés avant que le message courant ne puisse être envoyé.
- Exemple : numéros de séquence 1, 2, 4 avant 3 = '1,2,4/ 3'
- Heure de début / Heure de fin, heures définies par l'utilisateur et utilisées pour définir des contraintes. Exemple : [heure = midi] 1 : manger() Ce message n'est envoyé que s'il est midi

28

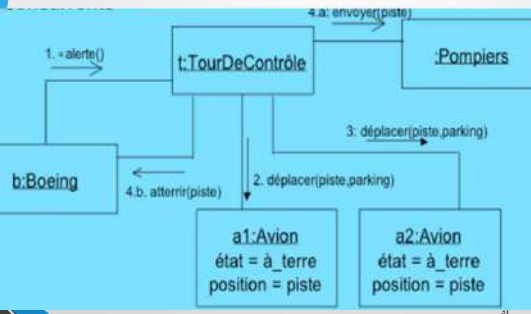
Exemples Recrutement via un cabinet de recrutement :



Exemples Recrutement via un cabinet de recrutement :



Exemple ???



Exercice

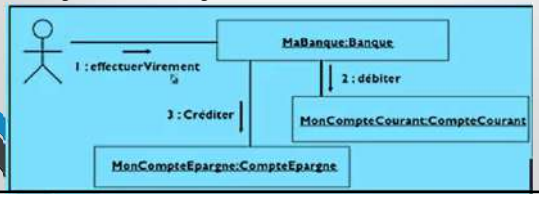
- On souhaite gérer les différents objets qui concourent à l'activité d'un magasin de vente de pièces (Par lot) :
- Le client demande au vendeur des renseignements sur les lots disponibles
- Le vendeur lui fournit toutes les informations nécessaires
- Le client commande alors la composition de son choix et le vendeur émet le bon de commande qu'il transmet à son ouvrier.
- Le vendeur édite ensuite la facture correspondante.
- L'ouvrier prépare la commande puis archive le bon de commande
- La facture est remise au client pour règlement une fois la commande est prête
- Une fois la facture réglée, le client récupère sa commande et quitte le magasin.

→ Modéliser cette situation à l'aide d'un diagramme de collaboration.

33

Conclusion

- Un diagramme de collaboration est identique à un diagramme d'objets auquel on ajoute des envois de message
- Les messages sont représentés par des flèches numérotés pour indiquer l'ordre des envois
- Un élément extérieur au système comme un acteur peut figurer dans un diagramme de collaboration



Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaoufb@yahoo.fr

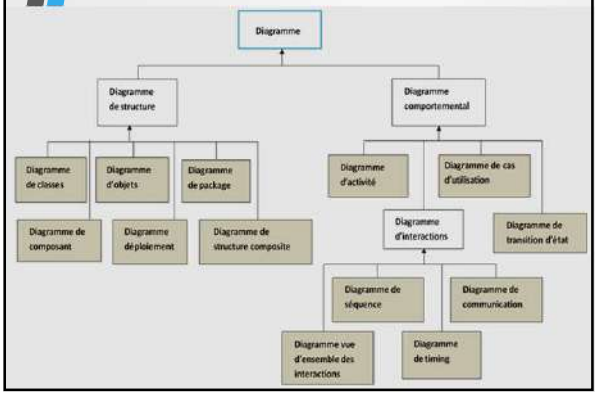
35

UML

- UML est le langage de modélisation orienté objet le plus connu et le plus utilisé au monde
- UML s'applique à plusieurs domaines
- UML n'est pas une méthode

36

UML - vue statique- vue dynamique-



Introduction

- Les diagrammes de composants et les diagrammes de déploiement sont les deux derniers types de vues statiques en UML.
- Les premiers décrivent le système modélisé sous forme de composants réutilisables et mettent en évidence leurs relations de dépendance.
- Les seconds se rapprochent encore plus de la réalité physique, puisqu'ils identifient les éléments matériels (PC, Modem, Station de travail, Serveur, etc.), leur disposition physique (connexions) et la disposition des exécutables (représentés par des composants) sur ces éléments matériels.

38

Introduction

- Un diagramme de composants est un diagramme représentant l'organisation et les dépendances liant les éléments physiques logiciels d'un système.
- Un diagramme de composants propose une vision statique de l'organisation des éléments d'un logiciel
- Un diagramme de composants montre les dépendances existant entre les composants d'un logiciel
- Un diagramme de composants ne montre pas les interactions entre les composants physiques logiciels

39

Diagramme de composants

- Offre une vue de haut niveau de l'architecture du système
- Utilisé pour décrire le système d'un point de vue implémentation
- Permet de décrire les composants d'un système et les dépendances entre ceux-ci
- Illustre comment grouper concrètement et physiquement les éléments (objets, interfaces, etc.) du système au sein de modules qu'on appelle composants

40

Diagramme de composants

- En UML, un composant est un élément logiciel remplaçable et réutilisable qui fourni ou reçoit un service bien précis.
- Il peut être vu comme une pièce détachée du logiciel. Les plug-ins, les drivers, les codecs, les bibliothèques sont des composants.
- La notion de composant est proche de celle d'objet, dans le sens de la modularité et de réutilisation
- Le composant est à l'architecture du logiciel
- l'objet est à l'architecture du code.

41

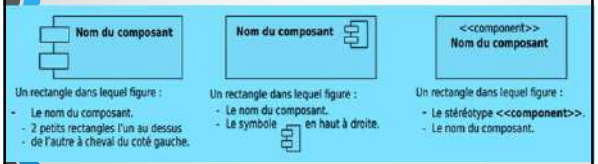
Diagramme de composants

- Les composants fournissent des services via des interfaces.
- Un composant peut être remplacé par n'importe quel autre composant compatible c'est-à-dire ayant les mêmes interfaces.
- Un composant peut évoluer indépendamment des applications ou des autres composants qui l'utilise à partir du moment ou les interfaces sont respectées

42

Diagramme de composants

- **Représentation graphique**
- **Les composants :** Il existe plusieurs possibilités pour représenter un composant :



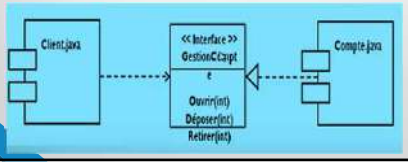
43

Diagramme de composants

- En UML les composants ne sont pas des éléments matériels mais des éléments logiciels
- Un composant est décrit par les interfaces qu'il offre et les interfaces qu'il requiert.
- Un composant offre donc des services (par exemple utilisables par d'autres composants) et peut aussi en nécessitez pour son propre fonctionnement.
- Tout composant peut être remplacé par un autre ayant les même interfaces.

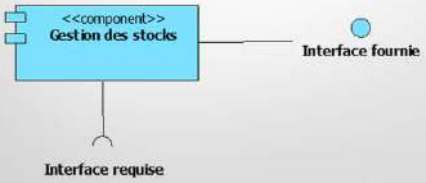
44

- Interface
- Une interface est un ensemble d'opérations servant à spécifier un service proposé par un composant ou par une classe
- L'interface contient les opérations mise à la disposition des autres composants
- Un composant peut implémenter plusieurs interfaces
- Un composant se doit de proposer une implémentation pour chacune de ses interfaces



45

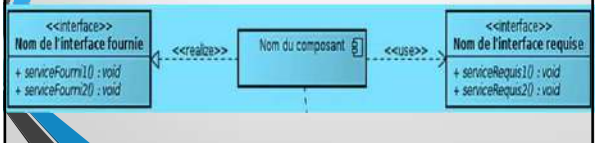
- Interface
- Il existe deux types d'interface :
- L'interface fournie est une interface qu'il est capable de mettre en œuvre
- L'interface requise est une interface dont il a besoin pour fonctionner.



46

- Les interfaces : il existe plusieurs possibilités pour représenter les interfaces :
- 1- Intégrées dans la représentation du composant :
- Nous reprenons l'une des trois représentations du composant que nous venons juste de voir et nous ajoutons un compartiment dans lequel nous listons les interfaces requises et fournies (grâce aux stéréotypes <<required interface>> et <<provided interface>>).

- 2. Dans un classeur séparé du composant dans lequel sont listés les différents services :
- Les interfaces requises sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype <<use>>.
- Les interfaces fournies sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype <<realize>> (le bout de la flèche est un triangle vide).



48

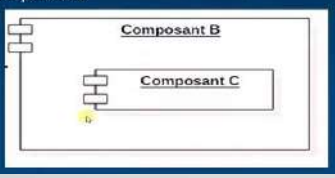
3. Avec des connecteurs d'assemblage :

- Les interfaces requises (représentées par un demi-cercle) et les interfaces fournies (représentées par un cercle) sont raccordées au composant par un trait.

49

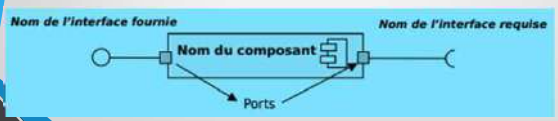
- **Dépendance** : un des éléments d'un composant A a besoin des services des éléments du composant B.

- **Contenance** : un composant peut être contenu dans un autre composant.



La notion d'un port

- Graphiquement, un port est représenté par un petit carré.
- On peut faire figurer le nom du port à proximité de sa représentation.



51

Exercice

- On considère une application constituée des fichiers suivants :
- un code source registre.cpp
- un programme exécutable registre.exe
- des librairies dynamiques personne.dll et cours.dll. Les librairies à liens dynamiques sont utilisées lors de l'exécution d'une application
- Question : Donnez le diagramme de composants correspondant

52

Conclusion

- Le diagramme de composants fait parti des diagrammes structuraux (statiques) d'UML.
- Il permet de représenter les différents éléments logiciels (composants) du système et leurs dépendances (relations qui les lient).
- Les composants fournissent des services via des interfaces.
- Un composant peut être remplacé par n'importe quel autre composant compatible c'est-à-dire ayant les mêmes interfaces

53

Diagramme de deploiement

- En UML, un diagramme de déploiement est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux.
- Les éléments utilisés par un diagramme de déploiement sont principalement les nœuds, les composants, les associations et les artefacts.
- Les caractéristiques des ressources matérielles physiques et des supports de communication peuvent être précisées par stéréotype.

54

Diagramme de deploiement

- Un diagramme de déploiement représente la façon dont déployer les différentes éléments d'un système
- Dans UML, les diagrammes de déploiement modélisent l'architecture physique d'un système.
- Les diagrammes de déploiement affichent les relations entre les composants logiciels et matériels du système, d'une part, et la distribution physique du traitement, d'autre part.

55

Diagramme de deploiement

- Les diagrammes de déploiement, que vous préparez généralement pendant la phase d'implémentation du développement, présentent la disposition physique des nœuds dans un système réparti, les artefacts qui sont stockés sur chaque nœud et les composants
- Les nœuds représentent des périphériques matériels tels que des ordinateurs, des détecteurs et des imprimantes, ainsi que d'autres périphériques qui prennent en charge l'environnement d'exécution d'un système.
- Les chemins de communication et les relations de déploiement modélisent les connexions dans le système.

56

Diagramme de deploiement

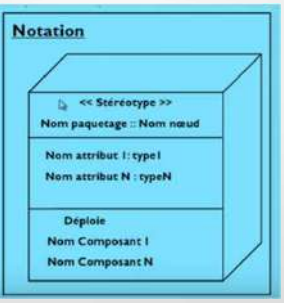
- Les diagrammes de déploiement mettant en évidence la configuration des nœuds de traitement en phase d'exécution, ainsi que leurs composants et artefacts,
- Ils montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels
- Les ressources matérielles sont représentées sous forme de nœuds
- Les diagrammes de déploiement peuvent montrer des instances de nœuds (un matériel précis), ou des classes de nœuds

57

les éléments des diagrammes de déploiement :

- **Nœuds dans les modèles UML**
- Dans les modèles UML, les nœuds sont des éléments de modèle qui représentent les ressources informatiques d'un système, telles que les ordinateurs personnels, les détecteurs, les périphériques d'impression ou les serveurs.
- Les nœuds peuvent être connectés à l'aide de chemins de communication pour décrire les structures de réseau.
- **Instances de nœud**
- Dans la modélisation UML, une instance de nœud est un élément de modèle qui représente une instanciation, ou occurrence réelle, d'un nœud.
- Les instances de nœuds sont basées sur des nœuds existants.

- Un nœud est représenté par un **cube**
- Le nom du nœud peut être précédé du nom du paquetage qui le contient
- Il est possible de développer le nœud de façon à faire apparaître le nom de ses **attributs**
- Il est possible de développer le nœud de façon à faire apparaître le nom des **composants** qu'il déploie



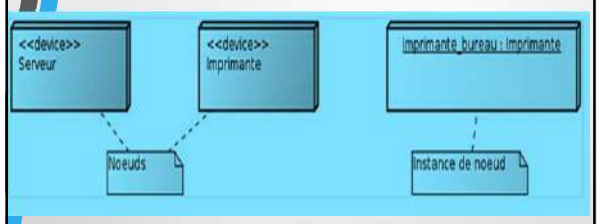
59

Nœud

- Un nœud est une ressource matérielle du système.
- En général, cette ressource possède au minimum de la mémoire et parfois aussi des capacités de calcul (des ressources humaines ou des périphériques sont des ressources modélisées par des nœuds).
- Les ressources matérielles sont quelquefois représentées avec le stéréotype <<device>> (généralement plutôt les périphériques mais également tout système informatique comme par exemple les ordinateurs de bureau).

60

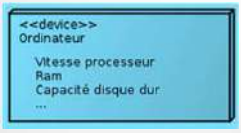
Nœud

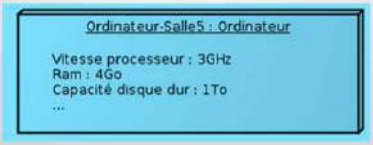


61

Nœud

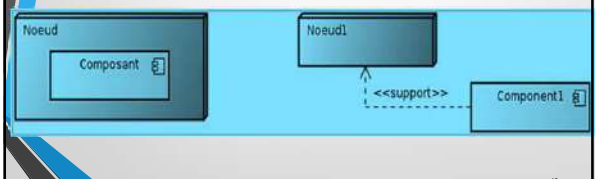
- Un nœud possède des attributs (quantité de mémoire, vitesse de processeur, marque, type…) que nous pouvons spécifier à l'intérieur.





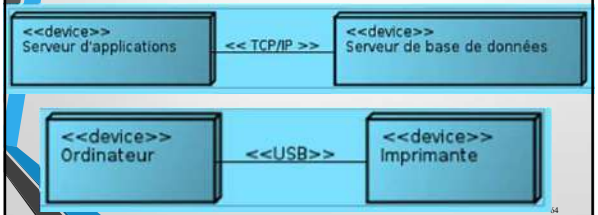
62

- Pour montrer qu'un composant est affecté sur un nœud, il faut :
- Soit en plaçant le composant dans le nœud.
- Soit en le reliant à l'aide d'une relation de dépendance (flèche en pointillées) stéréotypée «support» orientée du composant vers le nœud .



63

- Les différents nœuds qui apparaissent dans le diagramme de déploiement sont connectés entre eux par des lignes qui symbolisent un support de communication
- Le chemin de communication est donc un lien qui permet de modéliser de façon simpliste la communication entre 2 nœuds (liaison Ethernet, USB, …).



64

Artefacts

- **Dans les modèles UML, les artefacts sont des éléments de modèle qui représentent les entités physiques dans un système logiciel.**
- **Les artefacts représentent des unités physiques d'implémentation, telles que des fichiers exécutables, des bibliothèques, des composants de logiciel, des documents, et bases de documents.**

65

Artefacts

- L'artefact est le terme générique qui désigne n'importe quel élément produit du travail, c'est un élément concret et existant dans le monde réel (document, exécutable, fichier, base de donnée…).
- L'implémentation des modèles se fait sous forme d'artefacts.
- On dit que l'artefact est la manifestation du modèle qu'il implémente.

66

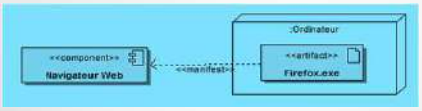
Artefacts

- Un artefact se représente par un rectangle contenant son nom et le stéréotype <<artifact>>
- Un artefact qui est la manifestation d'un composant est relié à celui-ci par une relation de dépendance (flèche en pointillées) stéréotypé <<manifest>> orienté de l'artefact vers le composant.

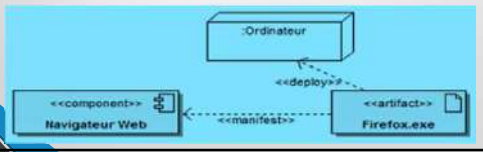
67

Artefacts

- L'artefact est placé soit :
- À l'intérieur du nœud dans lequel il est déployé.



- À l'extérieur du nœud dans lequel il est déployé, mais relié à celui-ci par une relation de dépendance stéréotypé <<deploy>>



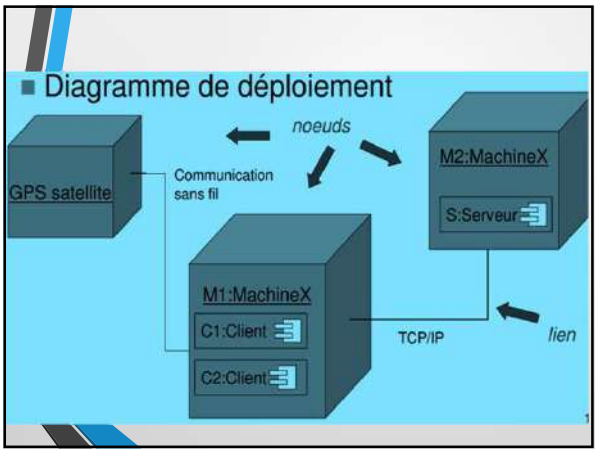


Diagramme de déploiement et diagramme de composants

- Les diagrammes de déploiement sont à distinguer des diagrammes de composants.
- Un diagramme de déploiement affiche les composants et artefacts en relation avec l'emplacement où ils sont utilisés dans le système déployé.
- Un diagramme de composants définit la composition des composants et artefacts dans le système

70

Conclusion : Besoin d'organisation

- Un modèle UML représente un système et son environnement
- Les diagrammes UML offrent différentes vues d'un même modèle
- Certains diagrammes sont complémentaires, d'autres non
- Certains diagrammes sont très abstrait, d'autres non
- Il est nécessaire de définir une organisation entre les diagrammes (Une méthode)

71

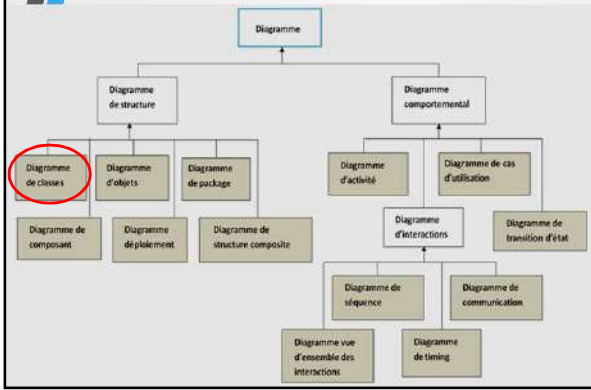
Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaouftb@yahoo.fr

1

4) Diagramme de classes - vue statique-



Introduction

- En UML, le modèle structurel ou statique d'un système est décrit à l'aide de :

  – Diagrammes de classes (description de tout ou d'une partie du système d'une manière abstraite en termes de classes et de relations).

  – Diagrammes d'objets (description d'exemples de configuration de tout ou partie du système, en termes d'objets, de valeurs et de liens).

3

Introduction

- Les diagrammes de cas d'utilisation modélisent à QUOI sert le système.
- Le système est composé d'objets qui interagissent entre eux et avec les acteurs pour réaliser ces cas d'utilisation.
- Les diagrammes de classe permettent de spécifier la structure et les liens entre les objets dont le système est composé.

4

Définition d'une classe

- Une classe est la description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun.
- Elle spécifie l'ensemble des caractéristiques qui composent des objets de même type.
- Une classe est composée d'un nom, d'attributs et d'opérations.
- Selon l'avancement de la modélisation ces informations ne sont pas forcément toutes connues.

5

Diagramme de classe

Intérêt

- Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet. Il est le seul obligatoire lors d'une telle modélisation. Il permet de:
  - Définir les futures composantes du système final.
  - Structurer le travail de développement de manière très efficace.
  - Séparer les composantes de manière à pouvoir répartir le travail de développement entre les membres (développeurs), dans le cas de travaux réalisés en groupe (surtout dans les milieux industriels).

6

Notion de classe

- La notion de classe est essentielle en programmation orientée objet : elle définit une abstraction, un type abstrait qui permettra plus tard d'instancier des objets.
- On distingue généralement des classes abstraites (qui ne peuvent pas être instanciées) et des classes "normales", qui servent à définir des objets.
- La classe intègre les concepts de type (en tant que «moule à instances »)

7

NOTATION :

- Tout nom de classe commence par une majuscule
- Une classe est un classeur représenté par un rectangle divisé en trois compartiments.

NomAttribut : type = ValeurInitiale

NomOpération(ListeArguments) : typeRetour

NomClasse

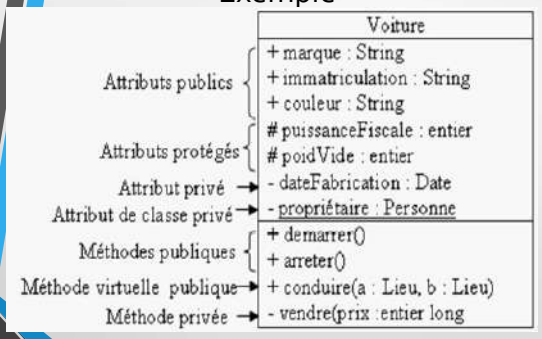
8

Attribut

- Les attributs définissent les propriétés des objets de la classe.
- Un attribut est défini par un nom, un type de données, une visibilité et peut être initialisé. Le nom de l'attribut doit être unique dans la classe.
- Les attributs représentent les données encapsulées dans les objets de cette classe.
- La syntaxe d'un attribut est la suivante :
- Visibilité nomAttribut [multiplicité] : typeAttribut = Initialisation ;

9

Exemple



10

Méthode

- Une méthode ou opération est un service qu'une instance d'une classe peut exécuter. Elle doit être unique. La notion de visibilité est la même que celle des attributs.
  - Opération : effectuée par l'objet
  - Méthode : effectuée par la classe
- Sa syntaxe est la suivante :

Visibilité nomFonction(directionParamètreN nomParamètreN : type ParamètreN) : typeRetour

11

Classe abstraite

- une classe abstraite est une classe dont l'implémentation n'est pas complète et qui n'est pas instanciable. Elle sert de base à d'autres classes dérivées (héritées). Pour indiquer qu'une classe est abstraite, on ajoute le mot-clé **abstract** derrière son nom.
- Le mécanisme des classes abstraites permet de définir des comportements (méthodes) dont l'implémentation (le code dans la méthode) se fait dans les classes héritées.

12

Classe abstraite

- Exemple :
- vous avez une classe Humain, à partir de laquelle dérivent la classe Homme et la classe Femme.
- En toute logique, Homme et Femme sont instanciables (les objets créés ont une existence en soi),
- Par contre la classe Humain sera déclarée abstraite car un objet Humain n'existe pas en tant que tel, puisqu'il manque l'information sur le sexe.
- Ici, la classe Humain servira à implémenter des méthodes qui seront utilisées à la fois pour Homme et pour Femme.

13

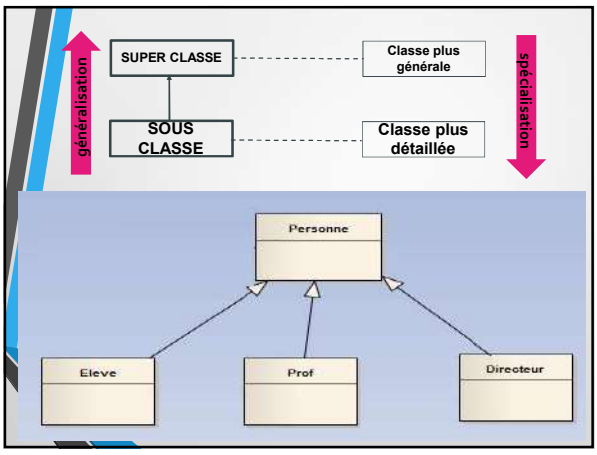
Relations entre les classes

14

Héritage (Généralisation/Spécialisation)

- Les propriétés principales de l'héritage sont :
- Tous les attributs qui se trouvent dans la Super-Classe sont vrais dans toutes les sous-classes.
- La classe enfant possède toutes les propriétés de ses classes parents, mais elle ne peut accéder aux propriétés privées de celle-ci.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées.

15



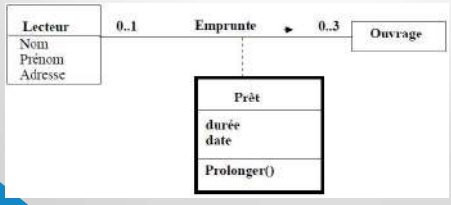
Association

- Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelle entre leurs instances ("est produit par", "est affilié à","se trouve à", "est conduit par"…).

17

Classe - Association

- Une classe-association possède à la fois les propriétés des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations.
- Une classe-association est reliée par un trait discontinu à l'association qu'elle représente.



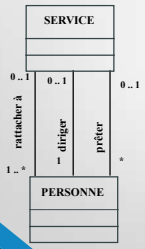
18

Agrégation et composition

- Une agrégation est une forme spéciale d'association exprimant une relation d'inclusion structurelle ou comportementale d'éléments (agrégés) dans un ensemble (agrégat).
- Graphiquement, on ajoute un losange vide du côté de l'agrégat.
- La composition, également appelée agrégation composite, décrit une contenance structurelle (contenance physique) entre instances. Ainsi, la destruction de l'objet composite implique ladestruction de ses composants.
- Graphiquement, on ajoute un losange plein du côté de l'agrégat.

19

MULTIPLICITES (cardinalités):



1	Un et un seul
0..1	Zéro ou Un
M..N	De M à N (entiers naturels)
0..*	De Zéro à plusieurs
*	
1..*	De Un à plusieurs
N	Exactement N (entier naturel)

20

Définition d'une base de donnée

- Une base de données (son abréviation est BD, en anglais DB, data base) est une entité dans laquelle il est possible de stocker des données de façon structurée et avec le moins de redondance possible afin d'être utilisées par des programmes et par des utilisateurs différents.
- Ainsi, la notion de base de données est généralement couplée à celle de réseau, afin de pouvoir mettre en commun ces informations, d'où le nom de base.
- On parle généralement de système d'information pour désigner toute la structure regroupant les moyens mis en place pour pouvoir partager des données.

21

Définition d'une base de donnée

- La gestion de la base de données se fait grâce à un système appelé SGBD (système de gestion de bases de données) ou en anglais DBMS (Data base management system).
- Le SGBD est un ensemble de services (applications logicielles) permettant de gérer les bases de données, c'est-à-dire :
  - permettre l'accès aux données de façon simple
  - autoriser un accès aux informations à de multiples utilisateurs
  - manipuler les données présentes dans la base de données (insertion, suppression, modification).

22

Définition d'une base de donnée

- Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle.

23

24

Passage de UML vers un schéma relationnel

- But du passage :
- Comme le modèle relationnel est très populaire, l'investissement des entreprises dans les bases de données relationnelles est tellement important, qu'il est délicat de le remettre en cause pour bénéficier des avantages liés aux technologies objet.

25

Passage de UML vers un schéma relationnel

- Une classe définit une structure de données à laquelle souscrivent des instances ;
- Elle correspond donc à une table du modèle relationnel : chaque attribut donne lieu à une colonne,
- chaque instance stocke ses données dans une ligne (T-uplet) et son OID sert de clé primaire.
- **OID** est un identifiant unique pour les objets (Object IDentifier).

26

Passage de UML vers un schéma relationnel

- Certains attributs de type complexe ne correspondent à aucun des types SQL ;
- on rencontre fréquemment ce cas pour les attributs représentant une structure de données.
- Un type complexe peut être conçu :
  - soit avec plusieurs colonnes, chacune correspondant à un champ de la structure.
  - soit avec une table spécifique dotée d'une clé étrangère pour relier les instances aux valeurs de leur attribut complexe.

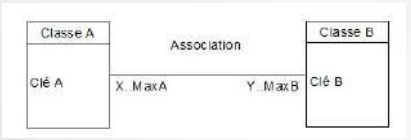
27

Règles de passage

28

Passage de UML vers un schéma relationnel

Modèle objet	Modèle relationnel
Classe	Table
Attribut de type simple	Colonne
Attribut de type composé	Colonne ou clé étrangère
Instance	T-uplet
Identifiant	Clé primaire
Association	Clé étrangère ou Table de liens
Héritage	Clé primaire identique sur plusieurs tables



	Max A=1	Max A>1
Max B=1	comparer les Min sinon si MinA=MinB Alors faire un choix	Mettre clé B dans A comme clé étrangère
Max B>1	Mettre clé A dans B comme clé étrangère	Créer une relation association comme clé A et clé B

30

Université d'Alger 1-Ben Youcef Ben Khedda
Faculté des Sciences
Département de Mathématiques et Informatique

Génie Logiciel 2 (GL 2)

A. Benghalia
abderaouftb@yahoo.fr

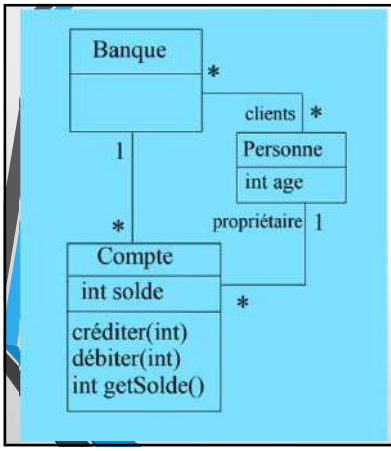
Introduction

- Nous avons déjà vu comment exprimer certaines formes de contraintes avec UML :
- les attributs dans les classes, les différents types de relations entre classes (généralisation, association, agrégation, composition), la cardinalité et la navigabilité des propriétés structurelles, etc. ;
- les contraintes de visibilité, les méthodes et classes abstraites, etc.
- Dans la pratique, toutes ces contraintes sont très utiles, mais se révèlent insuffisantes.
- Toutefois, UML permet de spécifier explicitement des contraintes particulières sur des éléments de modèle.

- **Exemple : Application bancaire**
  - Des comptes bancaires
  - Des clients
  - Des banques
- **Spécification**
- Un compte doit avoir un solde toujours positif
- Un client peut posséder plusieurs comptes
- Un client peut être client de plusieurs banques
- Un client d'une banque possède au moins un compte dans cette banque

Une banque gère plusieurs comptes

Une banque possède plusieurs clients



UML ne permet pas d'exprimer toutes les spécifications

Introduction

- **Problèmes**
- Pour spécifier complètement une application
- Diagrammes UML seuls sont généralement insuffisants
- Nécessité de rajouter des contraintes
- Comment exprimer ces contraintes ?
- Langue naturelle mais manque de précision, compréhension pouvant être ambiguë
- Langage formel avec sémantique précise : par exemple OCL

5

Object Constraint Language OCL

- OCL est un langage à objets déclaratif
- But : spécifier des contraintes dans les diagrammes UML,
- Il permet :
  - Typer les différentes variables (attributs et opérations)
  - Définir les assertions (invariants de classes, pré- et post-condition d'opération).
  - Préciser les contraintes dans différents diagrammes:
    - Contraintes temporelles,
    - Contraintes d'objets,
    - Contraintes de garde,
    - contraintes d'héritage…

6

Object Constraint Language OCL

- Langage formel (mais « simple » à utiliser) avec
- Une syntaxe, une grammaire, une sémantique (manipulable par un outil)

- OCL définit :
- Un environnement de base: type et opérations prédéfinis
- Et un langage de prédicat, muni d'une opération de navigation.
- L'opération de navigation permet de passer d'un objet à l'autre en suivant leurs liens (statiques).

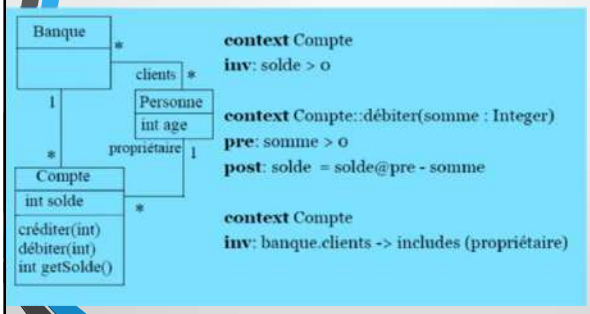
7

Object Constraint Language OCL

- OCL permet principalement d'exprimer deux types de contraintes sur l'état d'un objet ou d'un ensemble d'objets
- Des invariants qui doivent être respectés en permanence
- Des pré et post-conditions pour une opération
  - Précondition : doit être vérifiée avant l'exécution
  - Postcondition : doit être vérifiée après l'exécution
- Attention
- Une expression OCL décrit une contrainte à respecter et non pas le « code » d'une méthode

8

Exemple: application OCL



9

OCL

- C'est avec OCL (Object Constraint Language) qu'UML formalise l'expression des contraintes.
- Il s'agit donc d'un langage formel d'expression de contraintes bien adapté aux diagrammes d'UML, et en particulier au diagramme de classes.
- Il sert, entre autres, à spécifier des
  - Invariants sur des classes
  - Pré et post-conditions sur des opérations
  - Gardes sur transitions de diagrammes d'états ou de messages de diagrammes de séquence/collaboration

10

Contexte

- Une expression OCL est toujours définie ou évaluée dans un contexte donnée
- Il définit la portée des variables et le type d'utilisation
- Mot-clé : **context**
- Exemple
- **context** Compte
- L'expression OCL s'applique à la classe Compte, c'est-à-dire à toutes les instances de cette classe

11

Invariant

- Un invariant exprime une contrainte sur un objet ou un groupe d'objets qui doit être respectée en permanence
- Mot-clé : inv
- Exemple

context Compte

inv: solde > 0

- Pour toutes les instances de la classe Compte, l'attribut solde doit toujours être positif

12

Pré et Post- condition

- Pour spécifier une opération
- Pré-condition : état qui doit être respecté avant l'appel de l'opération
- Post-condition : état qui doit être respecté après l'appel de l'opération
- Mots-clés : pre et post

13

Pré et Post- condition

- Dans la post-condition, deux éléments particuliers sont utilisables
- Pseudo-attribut result : référence la valeur retournée par l'opération
- mon_attribut@pre : référence la valeur de mon_attribut avant l'appel de l'opération
- Syntaxe pour préciser la signature de l'opération

context ma_classe::mon_op(liste_param) : type_retour

14

Exemple

context Compte::débiter(somme : Integer)

pre: somme > 0

post: solde = solde@pre – somme

- la somme à débiter doit être positive pour que l'appel de l'opération soit valide
- après l'exécution de l'opération, l'attribut solde doit avoir pour valeur la différence de sa valeur avant l'appel et de la somme passée en paramètre.

15

Exemple

context Compte::getSolde() : Integer

post: result = solde

- Le résultat retourné doit être le solde courant

→ On ne décrit pas comment l'opération est réalisée mais des contraintes sur l'état avant et après son exécution

16

Types du modèle UML

- Toute expression OCL est écrite dans le contexte d'un modèle UML donné.
- Une contrainte OCL peut référencer une valeur de type de la manière suivante :

<nom_type_enuméré>::valeur

- Par exemple, la classe *Personne* possède un attribut *genre* de type *Genre*.
- On peut donc écrire la contrainte :

context Personne

inv : genre = Genre::femme

Dans ce cas, toutes les personnes doivent être des femmes.

17

Accès aux objets

- Dans une contrainte OCL associée à un objet, il est possible d'accéder aux caractéristiques (attributs, opérations et terminaison d'association) de cet objet, et donc, d'accéder de manière transitive à tous les objets (et leurs caractéristiques) avec lesquels il est en relation.
- La navigation permet de passer d'un objet à l'autre via une association
- Dans une contrainte OCL associée à un objet, on peut:
- Accéder à l'état interne de cet objet (ses attributs)
- Naviguer dans le diagramme : accéder de manière transitive à tous les objets (et leur état) avec qui il est en relation

18

Accès aux caractéristiques et aux objets

- self : référence l'objet de départ,
- Le propriétaire d'un compte doit avoir plus de 18 ans

context Compte

inv: self.propriétaire.age >= 18

- banque.clients : ensemble des clients de la banque

associée au compte (référence par transitivité)

- banque.clients.age : ensemble des âges de tous les clients

de la banque associée au compte

19

Accès aux caractéristiques et aux objets

- Une opération peut avoir des paramètres, il faut alors les préciser entre les parenthèses de l'opération.
- Par exemple, dans le contexte de la classe *Compte*, on peut utiliser les expressions suivantes :
- solde ;

self.solde ;

- getSolde() ;

self.getSolde() ;

- débiter(1000) ;

self.débiter(1000).

20

Accès aux objets

- Attributs ou paramètres d'une opération : utilise leur nom directement
- Objet(s) en association : on utilise au choix
  - Le nom de la classe associée (avec la première lettre en minuscule)
  - Le nom de l'association si elle nommée
  - Le nom du rôle d'association du coté de la classe vers laquelle on navigue s'il est nommé
- La navigation retourne
  - Si cardinalité est de 1 pour une association : un objet
  - Si cardinalité > 1 : une collection d'objets

21

Les collections

- 4 types de collections
- Set : ensemble au sens mathématique, pas de doublons, pas d'ordre
- { 1, 4, 3, 5 } : Set(Integer)
- OrderedSet : idem mais avec ordre (les éléments ont une position dans l'ensemble)
- Bag : comme un Set mais avec possibilité de doublons
- { 1, 4, 1, 3, 5, 4 } : Bag(Integer)
- Sequence : un Bag dont les éléments sont ordonnés

22

- Quelques opérations de base sur les collections que propose le langage OCL.

- size():Integer
  retourne le nombre d'éléments (la cardinalité) de self.
- includes(objet:T):Boolean
  vrai si self contient l'objet objet.
- excludes(objet:T):Boolean
  vrai si self ne contient pas l'objet objet.
- count(objet:T):Integer
  retourne le nombre d'occurrences de objet dans self

23