Génie Logiciel **Objet**

Jean Quilbeuf
Jean.Quilbeuf@univ-ubs.fr

INF 1503

Séances

- Cours: Mercredi 8h15 - 10h15
- TD: Mercredi 14h - 16h

Évaluation

- Controle Continu:
  - 2 devoirs sur table
  - Quizz en début de cours
  - Mini-projet
- Partiel

Note finale: 1/3 Partiel + 2/3 CC

Aujourd'hui

- Rappels sur la programmation orientée objet
- Exercice introductif
- Introduction du Cours

Programmation orientée objet

Découper l'application en objets: chaque objet a un contenu et des actions possibles sur ce contenu. Les objets interagissent entre eux.

- Objet: attributs et opérations
- Classes
- Héritage

Objet: attributs et opérations

Attributs: Information nécessaire pour représenter l'état de l'objet

Opérations: Actions possibles sur l'état de l'objet

Objet: attributs et opérations

Attributs: Information nécessaire pour représenter l'état de l'objet

Opérations: Actions possibles sur l'état de l'objet

Exemple: Fenêtre (Interface Graphique)

Attributs:

- position
- taille
- visibilité
- titre
- ...

Opérations:

- agrandir
- déplacer
- fermer
- minimiser
- ...

Classe

Ensemble d'objets ayant le même type d'attributs et les mêmes opérations.

- deux objets de la même classe ont les mêmes attributs (la valeur des attributs peut varier d'un objet à l'autre)
- deux objets de la même classe ont les mêmes opérations
- pour les langages typés, la classe peut-être un type

Classe

Ensemble d'objets ayant le même type d'attributs et les mêmes opérations.

- deux objets de la même classe ont les mêmes attributs (la valeur des attributs peut varier d'un objet à l'autre)
- deux objets de la même classe ont les mêmes opérations
- pour les langages typés, la classe peut-être un type

Example: la classe des fenêtres définit

- les attributs (position, taille, visibilité, titre . . .)
- les opérations (agrandir, déplacer, fermer, minimiser, . . .)

communes à toutes les fenêtres

En pratique (C++)

Déclaration d'une classe:

```
class Personne {
  public:
    string nom;
    int age;
    Personne(string n, int a) : nom(n), age(a) {}
    void presentation(){
      cout << "Je suis " << nom << endl;
    }
    void anniversaire() { age++; }
};
```

En pratique (C++)

Déclaration d'une classe:

```
class Personne {
  public:
    string nom;
    int age;
    Personne(string n, int a) : nom(n), age(a) {}
    void presentation(){
      cout << "Je suis " << nom << endl;
    }
    void anniversaire() { age++; }
};
```

Création d'un objet (utilisation d'un constructeur)

```
Personne jd = Personne("John Doe",30);
jd.presentation(); // affiche "Je suis John Doe"
```

Héritage

L'héritage permet de caractériser un sous-ensemble d'une classe ayant des propriétes particulières.

Exemple: les fenêtres avec menus sont un sous-ensemble des fenêtres.

Héritage

L'héritage permet de caractériser un sous-ensemble d'une classe ayant des propriétes particulières.

Exemple: les fenêtres avec menus sont un sous-ensemble des fenêtres.

On dit que la classe fille (sous-ensemble) hérite de la classe mère.
La classe fille contient tous les attributs et opérations de la classe mère.
La classe fille peut avoir des attributs et opérations supplémentaires.

Dans notre exemple: le menu est un atribut suplémentaire, afficher le menu est une opération supplémentaire

En pratique (C++)

La classe “Actif” hérite de la classe “Personne”:

```
class Actif : Personne {
  public:
    string metier;
    Actif(string n, int a,string m) : Personne(n,a), metier(m) {
    void presentation(){
      cout <<  nom << ", " << metier << endl;
    }
};
```

Note: presentation() est redéfinie par “Actif”

En pratique (C++)

La classe “Actif” hérite de la classe “Personne”:

```
class Actif : Personne {
  public:
    string metier;
    Actif(string n, int a,string m) : Personne(n,a), metier(m) {
    void presentation(){
      cout <<  nom << ", " << metier << endl;
    }
};
```

Note: presentation() est redéfinie par “Actif”

Utilisation:

```
Actif ted = Actif("Ted Mosby",35,"Architctecte");
ted.presentation(); //affiche "Ted Mosby, Architecte"
```

Rappels objets

- objet = attributs + opérations
- classe = objets ayant des attributs et opérations communs
- héritage = spécification d'un sous-ensemble d'une classe

Questions ?

Exercice introductif

But: se mettre en situation de modification d'un code inconnu

- travail en groupe de deux
- chaque groupe a une implémentation différente de la même fonctionalité
- plusieurs modifications successives à faire

Finalité: comparer les différentes implémentations

Application: calculatrice NPI

Notation polonaise inversée

Notation postfixe d'expressions arithmétiques

- sans parenthèses
- sans ordre de priorité entre les opérateurs

On note d'abord les arguments puis l'opération à leur appliquer. Un argument peut-être une expression.

Exemples:

classique	NPI
$8-4$	$8\ 4\ -$
2×5	$2\ 5\ \times$
$(8-4)\times(2\times 5)$	$8\ 4\ -\ 2\ 5\ \times\ \times$
$2+(8-4)\times(2\times 5)$	$2\ 8\ 4\ -\ 2\ 5\ \times\ \times\ +$

Intérêt: facile à parser.

Évaluation d'une expression en NPI

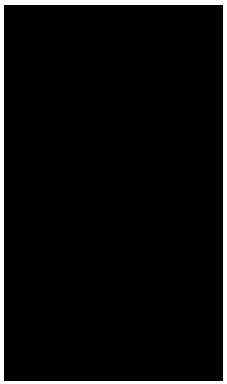
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile

2	8	4	-	2	5	×	×	+



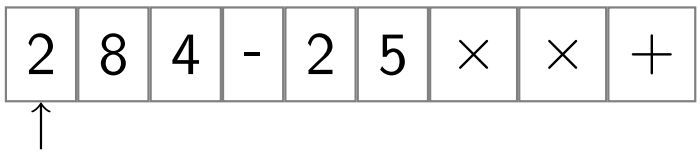
Évaluation d'une expression en NPI

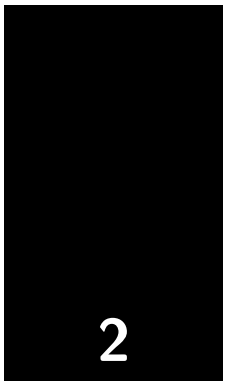
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





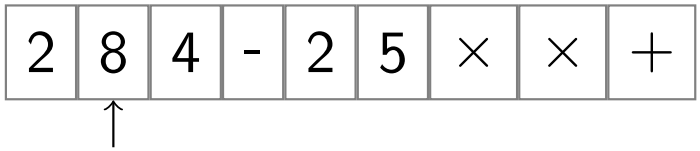
Évaluation d'une expression en NPI

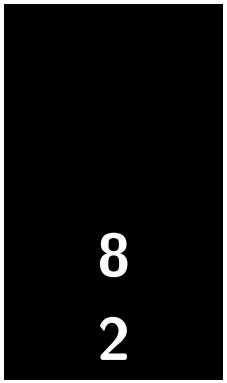
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





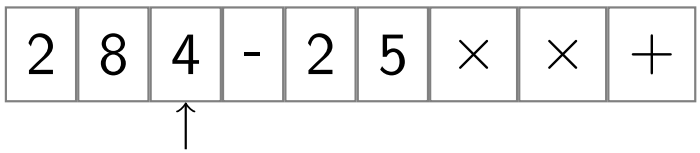
Évaluation d'une expression en NPI

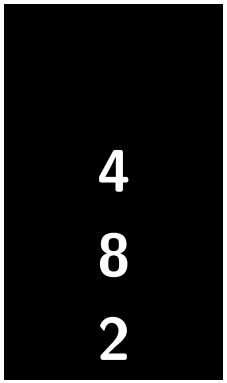
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





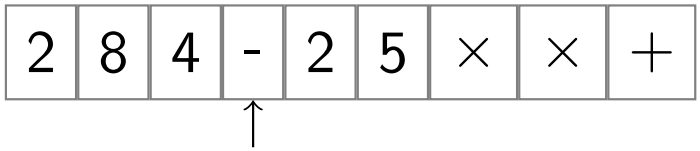
Évaluation d'une expression en NPI

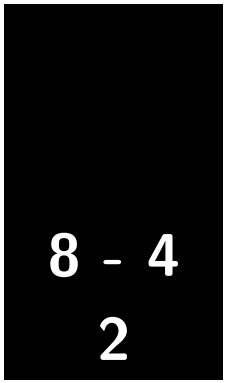
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





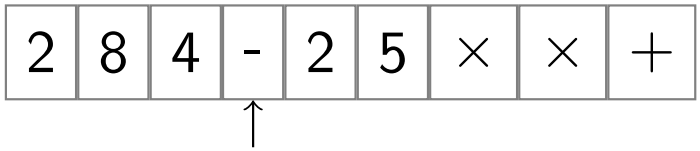
Évaluation d'une expression en NPI

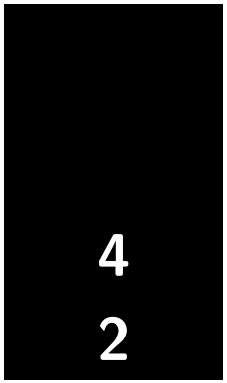
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





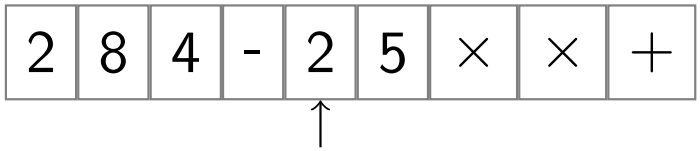
Évaluation d'une expression en NPI

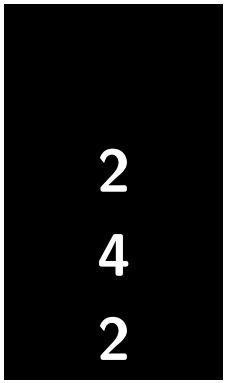
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





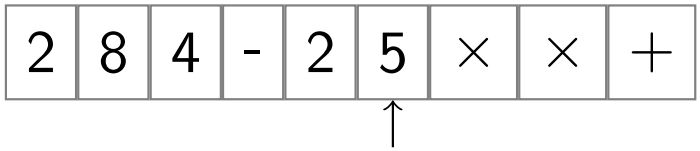
Évaluation d'une expression en NPI

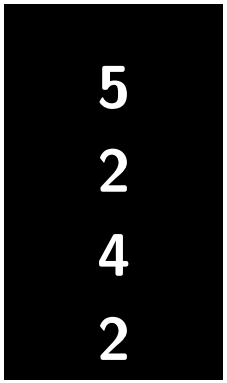
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





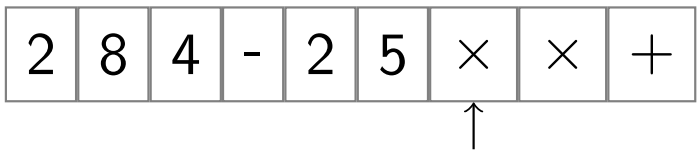
Évaluation d'une expression en NPI

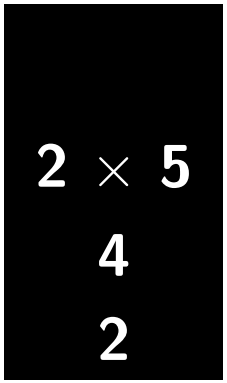
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





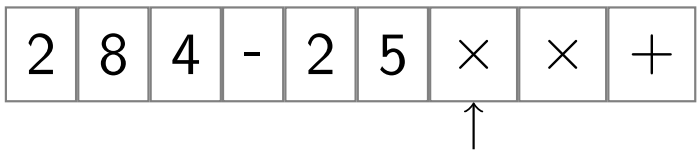
Évaluation d'une expression en NPI

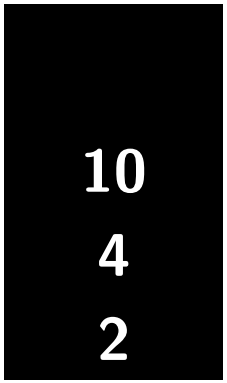
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





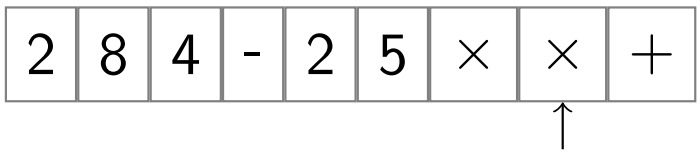
Évaluation d'une expression en NPI

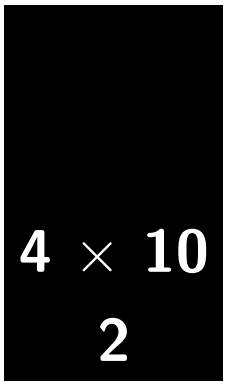
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





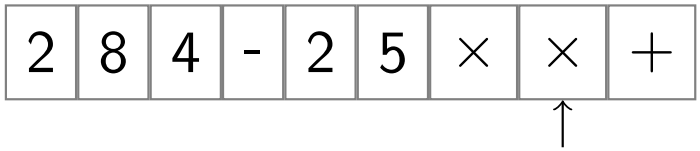
Évaluation d'une expression en NPI

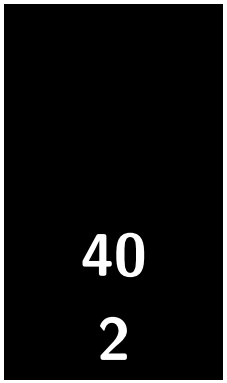
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





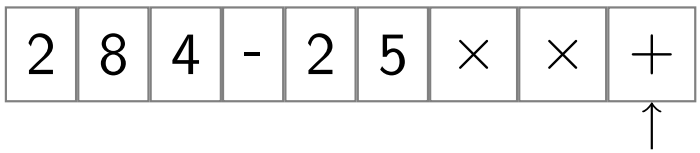
Évaluation d'une expression en NPI

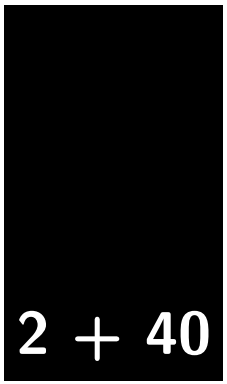
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





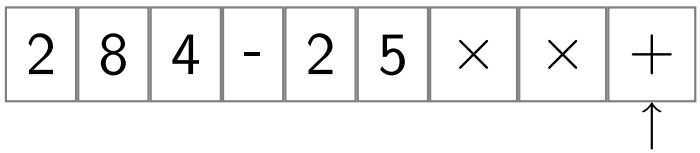
Évaluation d'une expression en NPI

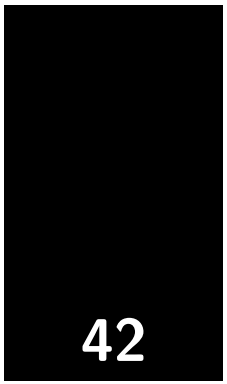
Algorithme (avec une pile de nombres)

Lire l'expression de gauche à droite. On rencontre:

- un nombre: on l'empile
- un opérateur: dépile deux nombres, fait l'opération, empile le résultat

on retourne le sommet de la pile





Code

Chaque groupe

- récupère son code
  `http://people.irisa.fr/Jean.Quilbeuf/code/`
- le compile (make)
- teste

Ensuite 3 modifications à faire.

Modification 1

L'addition, la soustraction et la multiplication sont implémentées.

Rajouter la division

Limite de temps: 5 min

M'envoyer une archive des sources dès que ça tourne.
`jean.quilbeuf@univ-ubs.fr`

Modification 2

On veut écrire des expressions impliquant la variable x
Exemple: x 2 +
L'utilisateur doit préciser la valeur de x, qui devient un paramètre de l'évaluation.

Rajouter

- la possibilité d'avoir une variable x
- le calcul de l'expression avec la valeur donnée par l'utilisateur

Limite de temps: 10 min

M'envoyer une archive des sources dès que ça tourne.
`jean.quilbeuf@univ-ubs.fr`

Modification 3

Pour permettre à l'utilisateur de vérifier qu'il n'a pas fait d'erreur, on souhaite afficher une version parenthésée de l'expression en plus du résultat.

Exemple: Entrée: 8 4 - 2 5 * *
Sortie: ((8 - 4) * (2 * 5)) = 40

Afficher la version parenthésée de l'expression avant le résultat

Limite de temps: 20 min

M'envoyer une archive des sources dès que ça tourne.
`jean.quilbeuf@univ-ubs.fr`

Debriefing

Que signifie “meilleure implémentation” ?

- Plus performante ?
- Celle avec moins de bugs ?
- La plus “élégante” ?
- La plus facile à modifier ?
- La plus facile à comprendre ?
- La moins coûteuse à produire ?
- La plus robuste ?

Debriefing

L'exercice que vous avez fait illustre (modestement) le problème de la maintenance du logiciel,

le besoin de modifier un logiciel sous contraintes:

- code plus ou moins connu,
- qui fonctionne, ou pas,
- code potentiellement critique,
- documentation plus ou moins présente,
- but du logiciel plus ou moins connu,
- temps limité

La maintenance représente une part important du coût d'un logiciel, plus que le développement[1]

[1] `http://www.irahul.com/2005/10/software-development-vs-software.html`

Génie Logiciel

Ensemble des méthodes, outils, et techniques visant à la production du logiciel.

Buts:

- satisfaire l'utilisateur/client (logiciel correct, performant, utilisable)
- permettre la maintenance (extension, réutilisation)
- contrôler les délais, les coûts, la qualité

Satisfaire l'utilisateur/client

- respecter les spécifications,
- faire un logciel facile à utiliser,
- le fournir en temps et en heure,
- compatibilité avec d'autres logiciels

$\Longrightarrow$ produire un bon logiciel

Permettre la maintenance

- faciliter l'identification des bugs
- permettre l'extension du logiciel,
- permettre la réusabilité de certaine partie,
- faciliter la portabilité vers d'autres environnments

$\Longrightarrow$ bien produire un logiciel

Permettre la maintenance

- faciliter l'identification des bugs
- permettre l'extension du logiciel,
- permettre la réusabilité de certaine partie,
- faciliter la portabilité vers d'autres environnments

$\Longrightarrow$ bien produire un logiciel

Aspects liés: bien produire aide a produire un bon logiciel dans les temps et dans le budget.

Nécessité du Génie logiciel

Pas forcément nécessaire pour les petits projets:

Nécessité du Génie logiciel

Indispensable pour les plus grand projets:

Plus un projet est gros, plus il a de chances d'échouer.

Comment faire ?

Accompagner le cycle de vie du logiciel:

- depuis la formulation des besoins,
- pendant la conception et le développement,
- pendant la maintenance,
- jusqu'à la fin de l'utilisation (ou du support) du logiciel

Processus permettant de définir et d'organiser les différentes étapes du projet.

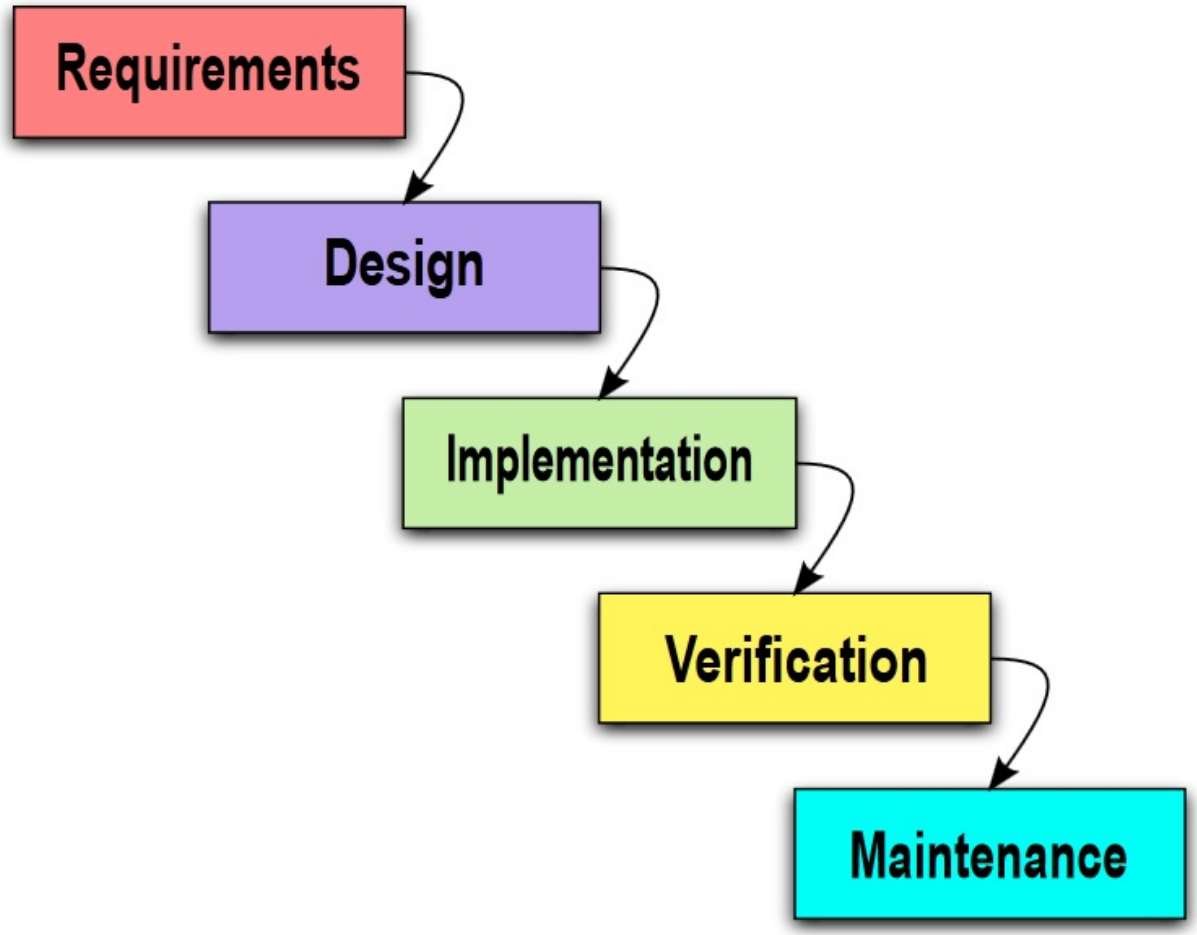
Outils pour aider:

- à accomplir les étapes
- à communiquer entre les étapes

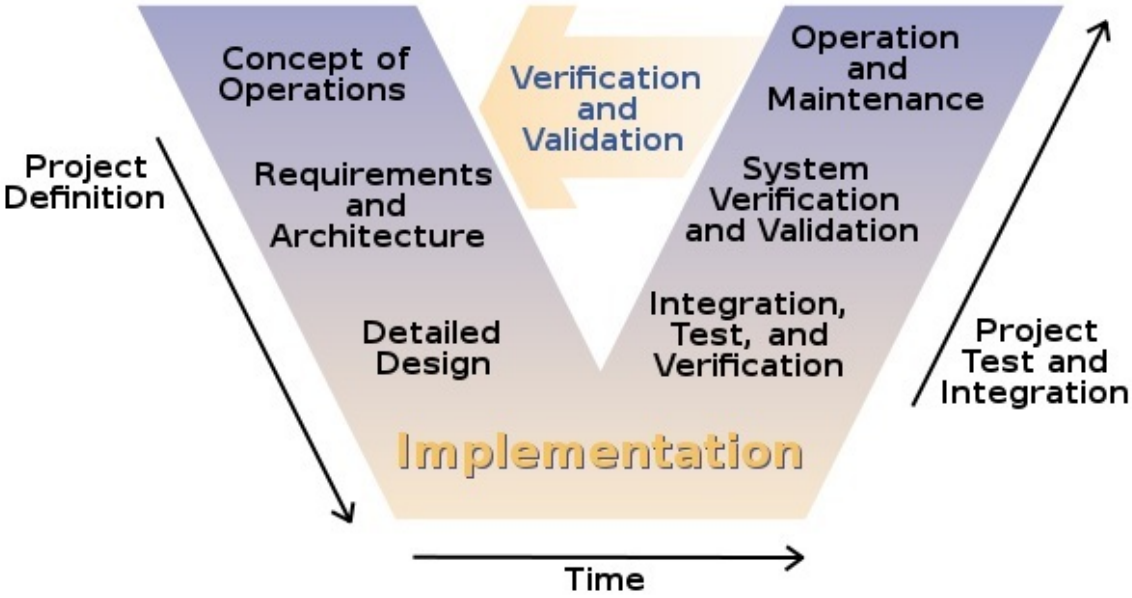
Exemples de processus

- En cascade
- En V
- En spirale

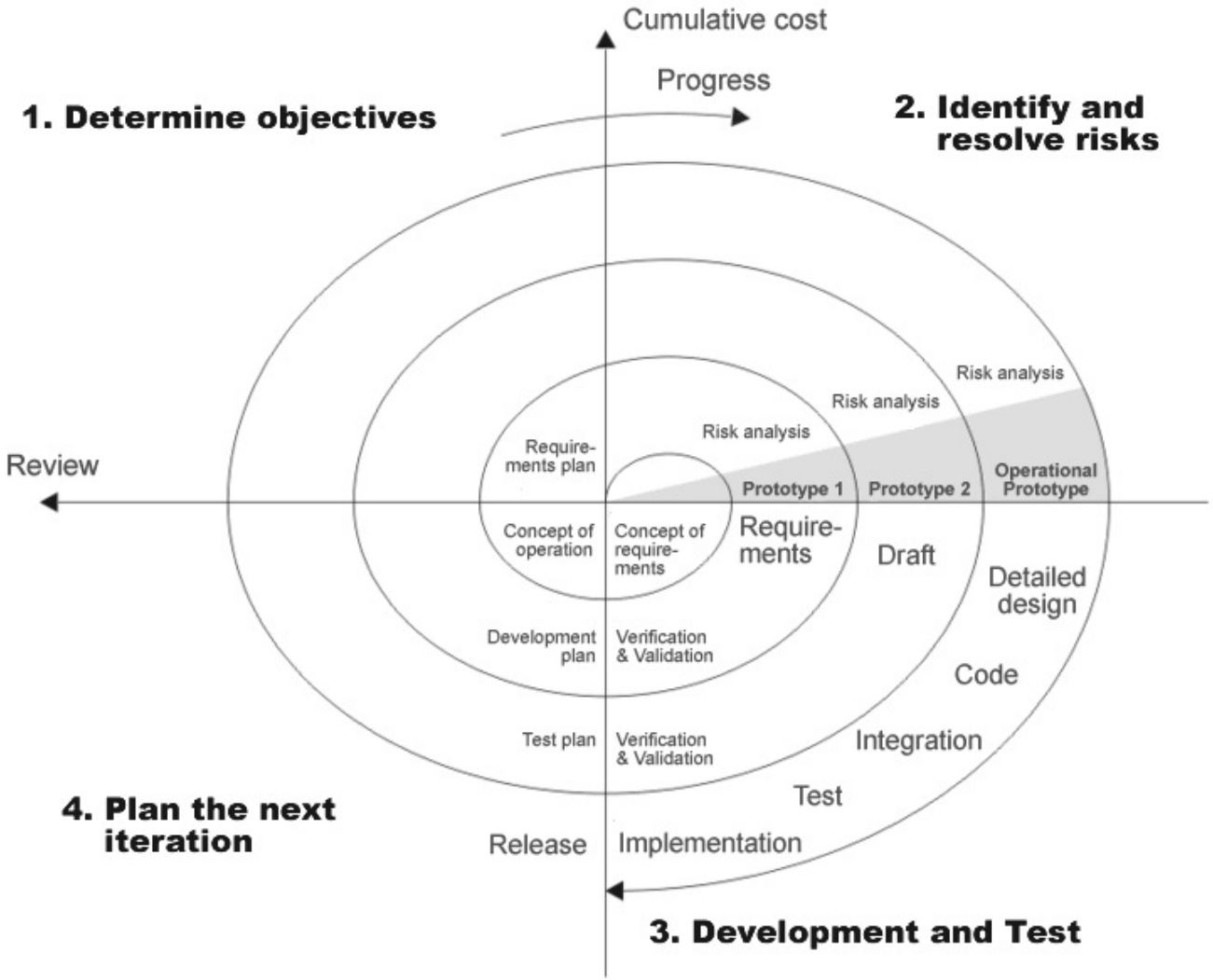
Exemples de processus – Cascade



Exemples de processus – V



Exemples de processus – Spirale



Exemples de processus

- En cascade
- En V
- En spirale

Le choix de processus dépend

- du type de système (cahier des charges fixe ou variable ?)
  - Cascade, V: bien pour les projets très contraints (aéronautique, systèmes critiques)
  - Spirale: pour les projets où le cahier des charges évolue
- de l'entité (entreprise, organisation) en charge du développment

Outils

Pour la communication entre les différentes étapes du processus de développment: UML

différents diagrammes pour différentes étapes

- fin de l'analyse des besoins:
  - diagramme de cas d'utilisation
  - diagramme de séquence (exemple de fonctionnement)
  - diagramme d'activité
- fin de la conception
  - diagramme de classe complet
  - digaramme de composant indiquant le déploiement
- ...

But de ce cours

- apprendre a modéliser les besoins en UML
- comprendre le lien entre le code et le modèle, générer l'un depuis l'autre
- réfléchir à la conception des logiciels (patrons de conception)