4. Conclusion (quelques remarques)

Patterns, idiomes, anti-patterns

- Il existe une grande variété de « *design patterns* »
  - Au-delà des GRASP patterns et des 23 patterns du GoF
  - P. ex. « inversion du contrôle » ou « injection de dépendances »…
  - Ou spécifiques à un domaine (p. ex. programmation concurrente ou programmation répartie)
- Idiomes de programmation
  - Techniques d'implémentation spécifiques au langage de programmation (alors que les *design patterns* sont agnostiques)
- Il existe aussi des « anti-patterns »…
  - « Modèles » de mauvais choix et erreurs de conception courantes
    - Exemples : interblocages et famines (synchronisation), plat de spaghetti, duplication de code, surcharge des interfaces utilisateurs…

Utilisation des patterns

- Attention !
  - Un *design pattern* n'est pas
    - La solution à un problème mais un modèle de solution
    - Une méthode de conception
    - Une règle applicable mécaniquement (rôle du développeur)
  - La maîtrise des *design patterns* demande
    - Un effort de synthèse (reconnaître, abstraire…)
      - Ils sont nombreux, parfois se ressemblent, ou se composent…
    - De l'expérience (apprentissage, expérimentation)
  - Ne pas surutiliser les *design patterns* au risque de rendre les applications inutilement complexes
    - Bien identifier ce qui est susceptible d'évolution
    - Privilégier la simplicité, pas l'utilisation de *patterns* à tout prix
  - En cas de composition de patterns (parfois utile), maîtriser la complexité !