Conception Object

Introduction

Alain Plantec

UBO

15 octobre 2012

Prérequis pour apprendre

La Conception objet : une affaire de style personnel certes **mais aussi surtout de savoir faire**

- Pour apprendre à Programmer :
  - Il faut programmer :)
  - Cycles courts - on commence "petit"
  - Regarder comment développent les autres - comprendre les bonnes pratiques
  - Il faut poser des questions (en cours, sur les listes...)

Prérequis pour apprendre

La Conception objet : une affaire de style personnel certes **mais aussi surtout de savoir faire**

- Pour apprendre à Programmer :
  - Il faut programmer :)
  - Cycles courts - on commence "petit"
  - Regarder comment développent les autres - comprendre les bonnes pratiques
  - Il faut poser des questions (en cours, sur les listes...)

Prérequis pour apprendre

La Conception objet : une affaire de style personnel certes **mais aussi surtout de savoir faire**

- Pour apprendre à Programmer :
  - Il faut programmer :)
  - Cycles courts - on commence "petit"
  - Regarder comment développent les autres - comprendre les bonnes pratiques
  - Il faut poser des questions (en cours, sur les listes...)

Prérequis pour apprendre

La Conception objet : une affaire de style personnel certes **mais aussi surtout de savoir faire**

- Pour apprendre à Programmer :
  - Il faut programmer :)
  - Cycles courts - on commence "petit"
  - Regarder comment développent les autres - comprendre les bonnes pratiques
  - Il faut poser des questions (en cours, sur les listes...)

Objectifs du cours

Apprendre à faire des choix judicieux pour la conception et la programmation objet

- Anticipation des changements
- Code évolutif
- Code simple qui répond au besoin **mais pas plus**

Tout système évolue, un système qui n'évolue pas est un système mort.

Objectifs du cours

Apprendre à faire des choix judicieux pour la conception et la programmation objet

- Anticipation des changements
- Code évolutif
- Code simple qui répond au besoin **mais pas plus**

Tout système évolue, un système qui n'évolue pas est un système mort.

Objectifs du cours

Apprendre à faire des choix judicieux pour la conception et la programmation objet

- Anticipation des changements
- Code évolutif
- Code simple qui répond au besoin **mais pas plus**

Tout système évolue, un système qui n'évolue pas est un système mort.

Prévoir le changement

Il faut identifier les parties susceptibles de changer, les abstraire et limiter la dépendance vis à vis d'un contexte particulier.

- Pas facile et coûteux
- La complexité du code augmente

- Il s'agit de **trouver le bon compromis** entre évolutivité et simplicité.
- Prise en compte du temps disponible pour décider et faire

Prévoir le changement

Il faut identifier les parties susceptibles de changer, les abstraire et limiter la dépendance vis à vis d'un contexte particulier.

- Pas facile et coûteux
- La complexité du code augmente
- Il s'agit de **trouver le bon compromis** entre évolutivité et simplicité.
- Prise en compte du temps disponible pour décider et faire

Prévoir le changement

Il faut identifier les parties susceptibles de changer, les abstraire et limiter la dépendance vis à vis d'un contexte particulier.

- Pas facile et coûteux
- La complexité du code augmente

- Il s'agit de **trouver le bon compromis** entre évolutivité et simplicité.
- Prise en compte du temps disponible pour décider et faire

Éviter la mauvaise conception

- Rigidité : difficulté à changer le code, cout important du changement
- Fragilité : des erreurs non-anticipées surviennent (attention au code "auto-protecteur") ; Erreurs en cascade en cas de changement
- Immobilité : code trop complexe pour être compris et réutilisé → **duplication de code**
- Viscosité : "verrue" plus rapide, facile à très court terme que de préserver une conception correcte → **code spaghetti**

Éviter la mauvaise conception

- Rigidité : difficulté à changer le code, cout important du changement
- Fragilité : des erreurs non-anticipées surviennent (attention au code "auto-protecteur") ; Erreurs en cascade en cas de changement
- Immobilité : code trop complexe pour être compris et réutilisé → **duplication de code**
- Viscosité : "verrue" plus rapide, facile à très court terme que de préserver une conception correcte → **code spaghetti**

Éviter la mauvaise conception

- Rigidité : difficulté à changer le code, cout important du changement
- Fragilité : des erreurs non-anticipées surviennent (attention au code "auto-protecteur") ; Erreurs en cascade en cas de changement
- Immobilité : code trop complexe pour être compris et réutilisé → **duplication de code**
- Viscosité : "verrue" plus rapide, facile à très court terme que de préserver une conception correcte → **code spaghetti**

Éviter la mauvaise conception

- Rigidité : difficulté à changer le code, cout important du changement
- Fragilité : des erreurs non-anticipées surviennent (attention au code "auto-protecteur") ; Erreurs en cascade en cas de changement
- Immobilité : code trop complexe pour être compris et réutilisé $\rightarrow$ **duplication de code**
- Viscosité : "verrue" plus rapide, facile à très court terme que de préserver une conception correcte $\rightarrow$ **code spaghetti**

Éviter la mauvaise conception

- Rigidité : difficulté à changer le code, cout important du changement
- Fragilité : des erreurs non-anticipées surviennent (attention au code "auto-protecteur") ; Erreurs en cascade en cas de changement
- Immobilité : code trop complexe pour être compris et réutilisé $\rightarrow$ **duplication de code**
- Viscosité : "verrue" plus rapide, facile à très court terme que de préserver une conception correcte $\rightarrow$ **code spaghetti**

You aren't gonna need it (YAGNI)

- Il n'y a pas de principe stricte et universel.
- Il n'y a pas qu'une seule bonne solution
- YAGNI : faire simplement ce dont on a besoin (mais le faire bien)