

# Développement à l'Aide de Langages Dynamiques : Smalltalk



ÉCOLE NATIONALE SUPÉRIEURE  
ÉLECTRONIQUE, INFORMATIQUE & RADIOCOMMUNICATIONS  
B O R D E A U X

Nada Ayad, Damien Cassou et Annabelle Souc

module IT308: Méthodologies et outils logiciels

# Présentation des Langages Dynamiques : Typage Dynamique

- Variables non typées
- Valeurs typées
- Pas de vérification de type par les compilateurs
- Ruby, Perl, Python...



**Ruby**  
*A Programmer's Best Friend*



# Spécificités des Langages Dynamiques

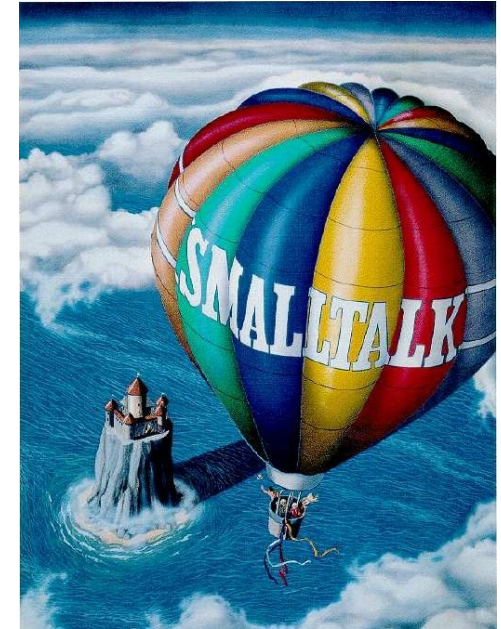
- Liaisons dynamiques
- Prototypages rapides
- Absence d'aide des compilateurs
- Moins de protection des méthodes

# Smalltalk

- Typage dynamique
- Tout est objet : même les entiers et les classes
- Tout est modifiable : classes ouvertes etc.
- Tout est cohérent : pas d'exception à la règle
- Multi plates-formes (machine virtuelle)
- Garbage collector
- Closures (blocs)

# Smalltalk

- Héritage simple
- Classes publiques
- Méthodes publiques
- Variables d'instances protégées
- Pas d'interface mais classes abstraites
- Valeur de retour par défaut : l'objet courant



# Smalltalk : Historique

- 1972 : Première version inspirée par Simula
- 1976 : Nouvelle conception :
  - Hiérarchie des classes avec racine unique
  - Syntaxe du byte-code fixée
  - Sémaphores
- 1978 : Expérimentation avec les 8086
- 1980 : Norme Smalltalk-80

# Smalltalk : Syntaxe

- Nombre limité de mots réservés :
  - `self`, `super`, `true`, `false`, `nil`, `thisContext`
  - 52 en Java, 38 en Ruby, 76 en C#
- Nommage des méthodes :  
`envoie: unMessage a: unePersonne`
- Appel de méthodes :  
`adam := Personne new.`  
`eve := Personne new.`  
`adam envoie: 'Fallait pas' a: eve`

# Smalltalk : Syntaxe

- Pas de constructeur :
  - seulement des méthodes qui retournent de nouvelles instances

```
Personne class>>nom: uneChaine
|personne|
personne := self new.
personne nom: uneChaine.
^ personne
```

```
eve := Personne nom: 'Eve'.
```



# Smalltalk : Syntaxe

- Utilisation de fonctions anonymes :  
`salaries select: [:salarie | salarie estFemme]`
- Pas d'instruction de contrôle (if, while, etc.) :  
`(eve messageRecu) ifTrue: [eve senVeut]`
- Création d'une classe :  
`Object subclass: #Personne  
 instanceVariableNames: 'prenom nom genre'  
 classVariableNames: ''  
 poolDictionaries: ''  
 category: 'Annuaire-modele'`

# Squeak



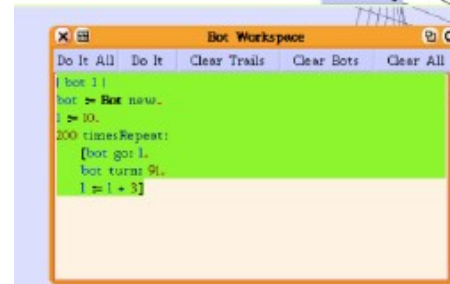
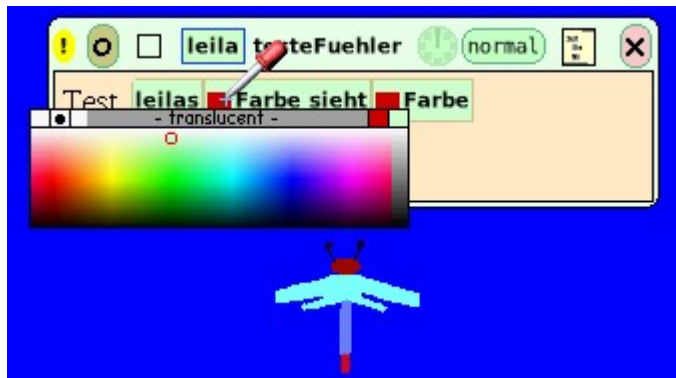
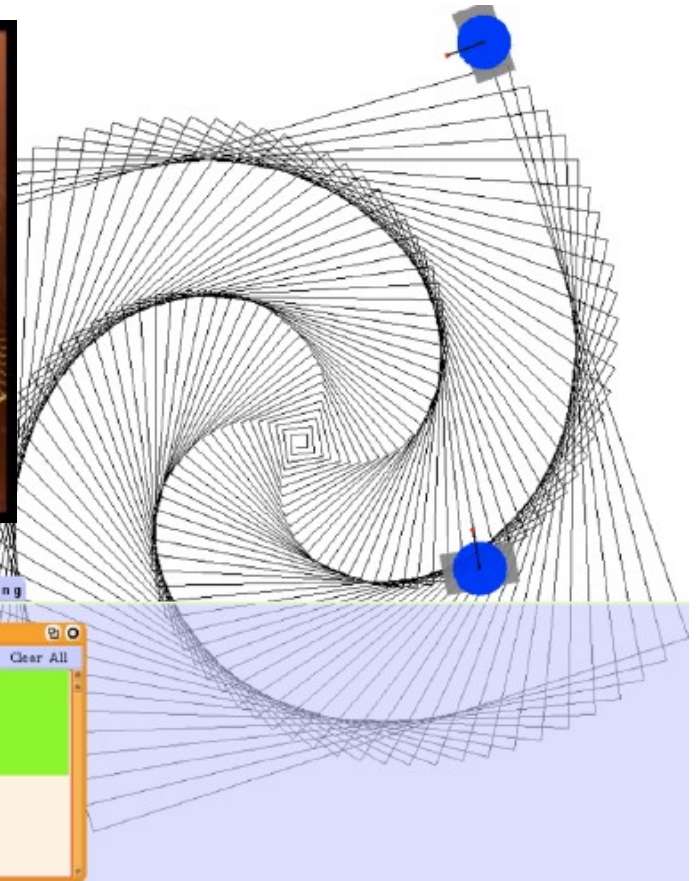
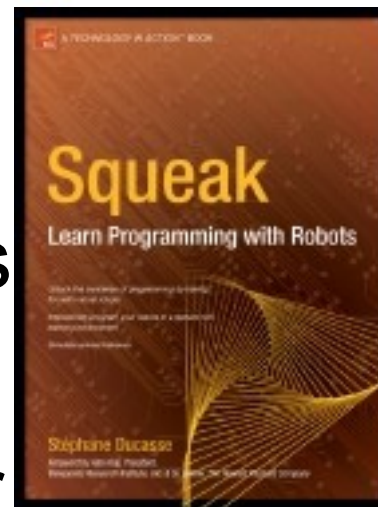
- Implémentation de Smalltalk
- Environnement de développement
- Pas de fichiers sources
- Machine virtuelle implémentée en Smalltalk
- Dernière version stable : décembre 2006

# Squeak : Installation

- Téléchargement de la machine virtuelle : 1 exécutable ([www.squeak.org](http://www.squeak.org))
- Téléchargement d'une image : 2 fichiers
- Drag&Drop de l'image sur la machine virtuelle
- C'est installé !

# Utilisation 1: Apprentissage de la Programmation

- Créé à l'origine pour les enfants
- Nombreux projets
- Projet « One Laptop per Child »



# Utilisation 2: Environnement de Développement

The screenshot displays the Squeak development environment with the following components:

- System Browser: Collection**
  - Left pane: Collections-Abstract, Collections-Arrayed, Collections-SequenceableColl, Collections-SkipList, Collections-Streams, Collections-Strings, Collections-Support, Collections-Text, Collections-Unordered, Collections-Weak.
  - Center pane: Collection (SequenceableColl, ArrayedCollecti).
  - Right pane: -- all --, accessing, adapting, adding, arithmetic, comparing, converting, copying, enumerating, allSatisfy:, anySatisfy:, associationsDo:, collect:, collect:thenDo:, collect:thenSelect:, count:, detect:, detect:ifNone:, detect:ifPresent:.
  - Buttons: instance, class.
  - Navigation: browse, variables, hierarchy, inheritance, senders, implementors, versions.
  - Content: `collect: aBlock`  
"Evaluate aBlock with each of the receiver's elements as the argument. Collect the resulting values into a collection like the receiver. Answer the new collection."  
| newCollection |  
newCollection \_ self species new.  
self do: [:each | newCollection add: (aBlock value: each)].  
↑ newCollection
- System Browser: CollectionTest**
  - Left pane: Collections-Weak, Collections-Stack, CollectionsTests-Abstract, CollectionsTests-Arrayed, CollectionsTests-SequenceableColl, CollectionsTests-Streams, CollectionsTests-Strings, CollectionsTests-Support, CollectionsTests-Text.
  - Center pane: CollectionTest.
  - Right pane: -- all --, initialize-release, testing, tests, -- Required --, -- Override --, -- Uncommented --, testIfEmptyifNotEm, testIfEmptyifNotEm, testIfNotEmpty, testIfNotEmptyDo, testIfNotEmptyDoif, testIfNotEmptyifEm.
  - Buttons: instance, class.
  - Navigation: browse, variables, hierarchy, inheritance, senders, implementors, versions.
  - Content: `testIfNotEmpty`  
empty ifNotEmpty: [self assert: false].  
self assert: (notEmpty ifNotEmpty: [self]) == self.  
self assert: (notEmpty ifNotEmpty: [:s | s first]) = \*x
- Shout Workspace**
  - Code: `{1 . 2 . 3 } select: #odd.`  
Transcript show: d
- Transcript**
  - Content: ----SNAPSHOT----an Array(19 January 2007 9:55:49 am) squeak-dev-72-2.image priorSource: 1580722
- Tools** (vertical bar on the right): Selector Browser, Monticello Browser.

# Utilisation 3: Développement web

Seaside

View | Contents | Changes | Search

## Seaside

**Seaside is a framework for developing sophisticated web applications in Smalltalk.**

Seaside provides a layered set of abstractions over HTTP and HTML that let you build highly interactive web applications **quickly**, **reusably** and **maintainably**. Seaside includes:

- **Programmatic HTML generation.** A lot of markup is boilerplate: the same patterns of lists, links, forms and tables show up on page after page. Seaside has a rich API for generating HTML that lets you abstract these patterns into convenient methods rather than pasting the same sequence of tags into templates every time.
- **Callback-based request handling.** Why should you have to come up with a unique name for every link and form input on your page, only to extract them from the URL and request fields later? Seaside automates this process by letting you associate blocks, not names, with inputs and links, so you can think about objects and methods instead of ids and strings.
- **Embedded components.** Stop thinking a whole page at a time; Seaside lets you build your UI as a tree of individual, stateful component objects, each encapsulating a small part of a page. Often, these can be used over and over again, within and between applications - nearly every application, for example, needs a way to present a batched list of search results, or a table with sortable columns, and Seaside includes components for these out the box.
- **Modal session management.** What if you could express a complex, multi-page workflow in a single method? Unlike servlet models which require a separate handler for each page or request, Seaside

Seaside

Download  
Tutorial  
A Walk on the Seaside  
Videos  
Documentation  
Community  
Links  
License  
Hosting  
Support

Login

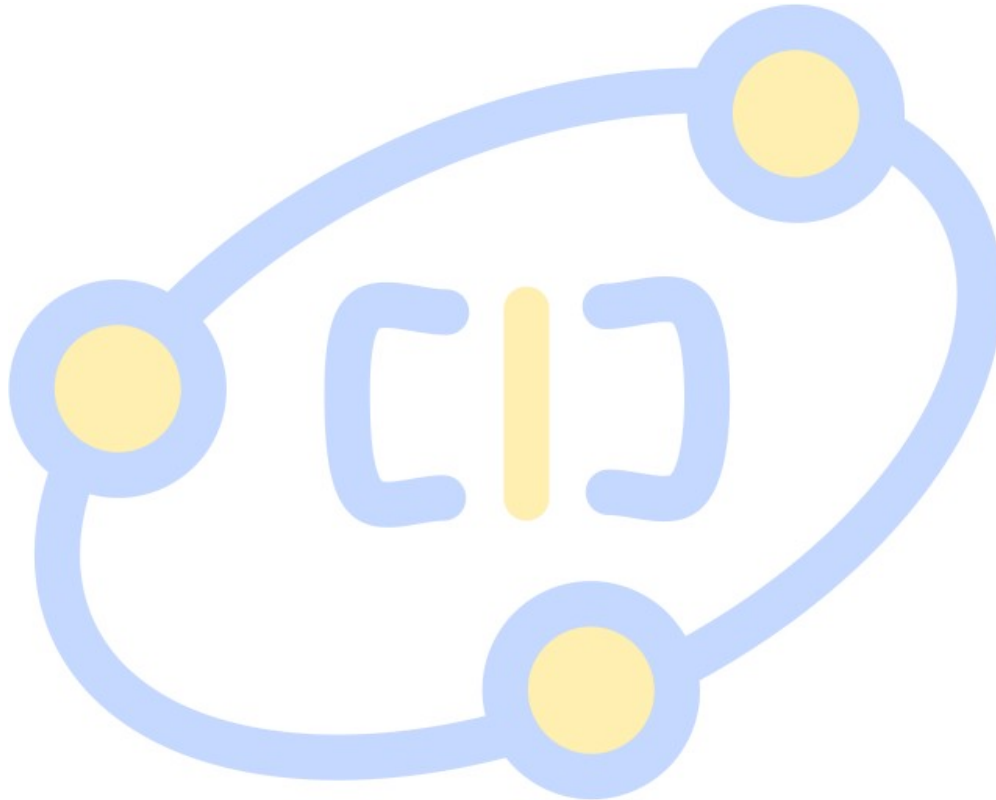
- Seaside
  - Orientation composant
  - Abstractions (http, html, javascript...)
- Magritte
  - Génération de formulaires
  - Méta modélisation

# Fonctionnalités de l'Environnement de Développement

- Interface sous forme d'un bureau
- Navigateur de code
  - Découpage en paquetages et catégories
  - Code source ouvert et disponible
  - Compilation incrémentale
- Outils
  - Outils de recherche de méthodes
  - Exécution du code et inspections rapides
  - Navigateur pour les tests...

# Et Maintenant

- Une démonstration !





# Conclusion

- Conclusion
  - Langages à typage dynamique
  - Smalltalk
  - Squeak
- Points de vue personnels

