

# Développement de logiciels par objets avec UML (Unified Modeling Language)

***Pr. Jean-Marc Jézéquel***  
**IRISA - Univ. Rennes I**

Campus de Beaulieu  
F-35042 Rennes Cedex  
Tel : +33 299 847 192 Fax : +33 299 842 532  
e-mail : jezequel@irisa.fr  
<http://www.irisa.fr/prive/jezequel>

# Plan général

- Contexte du génie logiciel : quels sont les problèmes
  - Correction, taille, variabilité
- Développement par objet avec UML
  - Principes et motivations de l'approche objet avec UML
  - Les 9 vues d'un modèle UML
- Processus de développement avec UML
  - Expression des besoins, Analyse
  - Conception (notion de design patterns)
  - Implantation et validation

# Contexte du Génie Logiciel

- Ingénierie de la réalisation de systèmes d'information
- Programming-in-the-Small
  - Problème de la qualité interne d'un composant
- Programming-in-the-Large
  - Faire face à la construction de systèmes de plus en plus gros : non spécifique au logiciel, mais aggravé par sa "mollesse«
  - Problème de gestion de la complexité, de communication, etc.
  - Maîtrise du processus de développement: délais,coûts,qualité
- Programming-in-the-Duration
  - Problème de la maintenance corrective et évolutive
  - Notion de ligne de produits

# Programming-in-the-Small :

## problème de la validité du logiciel

```
Acquérir une valeur positive n
Tant que n > 1 faire
    si n est pair
        alors n := n / 2
    sinon n := 3n+1
Sonner alarme;
```

- Prouver que l'alarme est sonnée pour tout n?
- Indécidabilité de certaines propriétés
  - problème de l'arrêt de la machine de Turing...

### ■ Recours au test

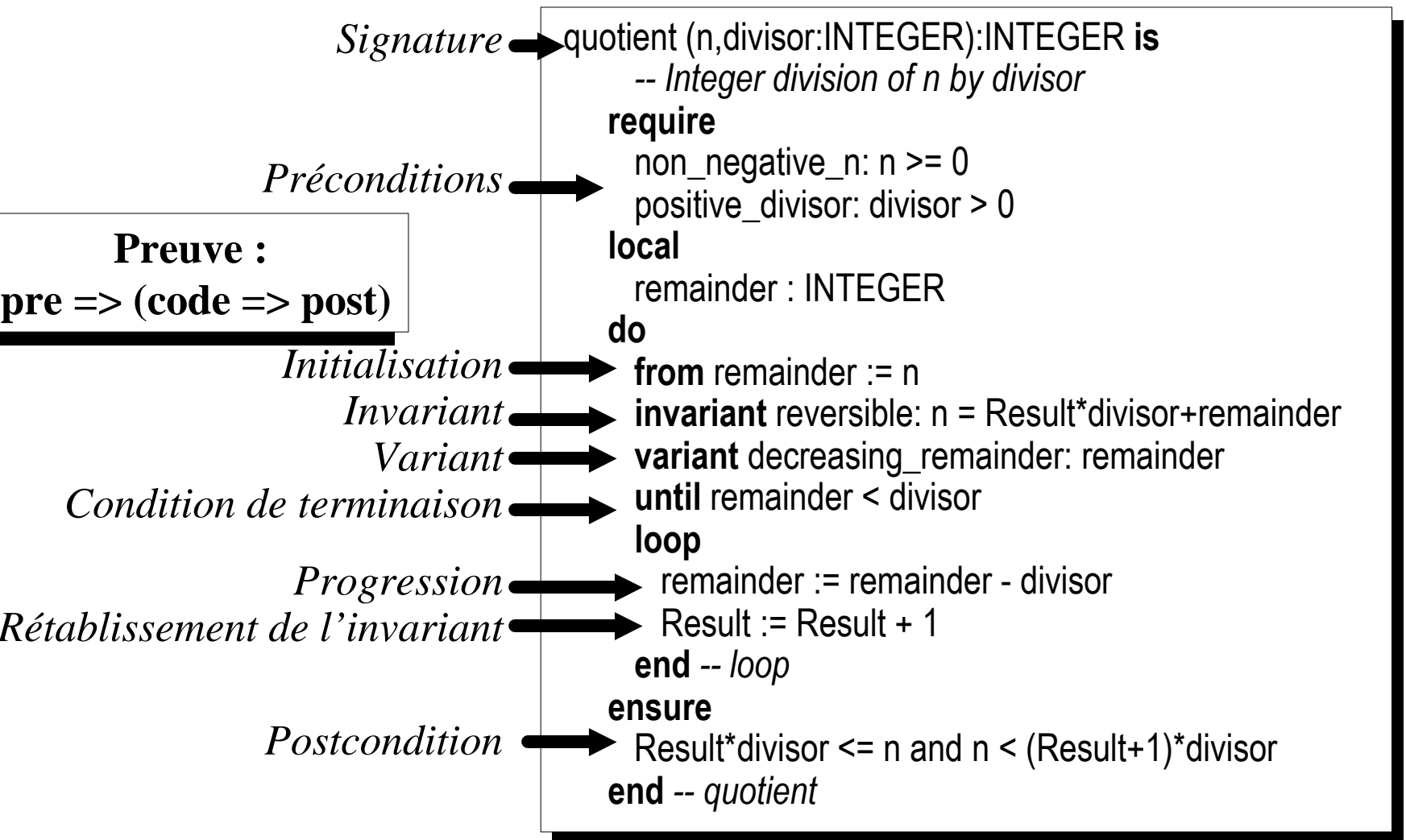
- ici, si machine 32 bits,  $2^{31} = 10^{10}$  cas de tests
- **5 lignes de code => 10 milliards de tests !**

# Programming-in-the-Small

- Problème de la puissance d'expression des langages
  - test à zéro + goto (=boucle) => puissance ~ machine de Turing
- Réduire la puissance d'expression pour prouver
  - Automates, réseaux de Petri...
- Solution réaliste : Associer des assertions aux programmes
  - Éléments de spécification formelles servant à construire des fragments de programmes prouvables
  - Pré-conditions et Post-conditions d'opérations
    - »  $pre \Rightarrow (code \Rightarrow post)$
  - Invariants et variant de boucles
    - » correction partielle + correction totale

# Preuve de programme :

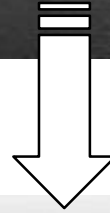
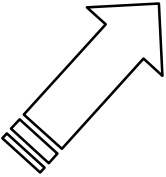
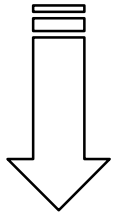
## exemple de la fonction « quotient »



# Produire des programmes prouvés

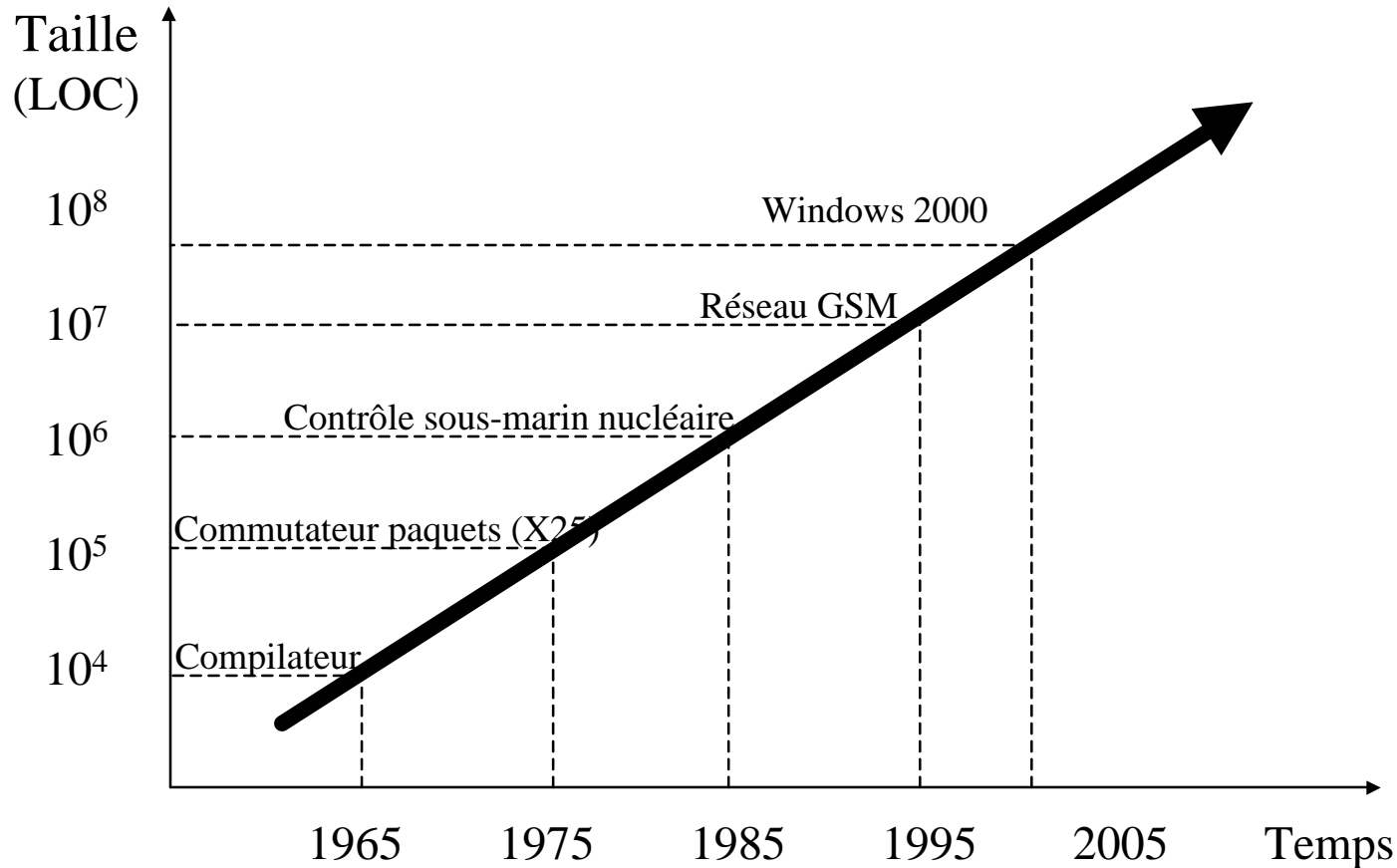
- C'est possible...
  - On peut construire des programmes de tel manière à ce qu'ils soient prouvables
  - En partie automatisable (Atelier B, etc.)
    - » Outils spécifiques
- ...mais coûteux
  - Preuve au moins aussi complexe que le code
  - Autant de chances de se tromper dans la preuve...
- En pratique :
  - Réservé à des (petits) sous-systèmes (très) critiques
  - Recours aux tests pour le reste

# Programming-in-the-Large : Gérer la complexité due à la taille





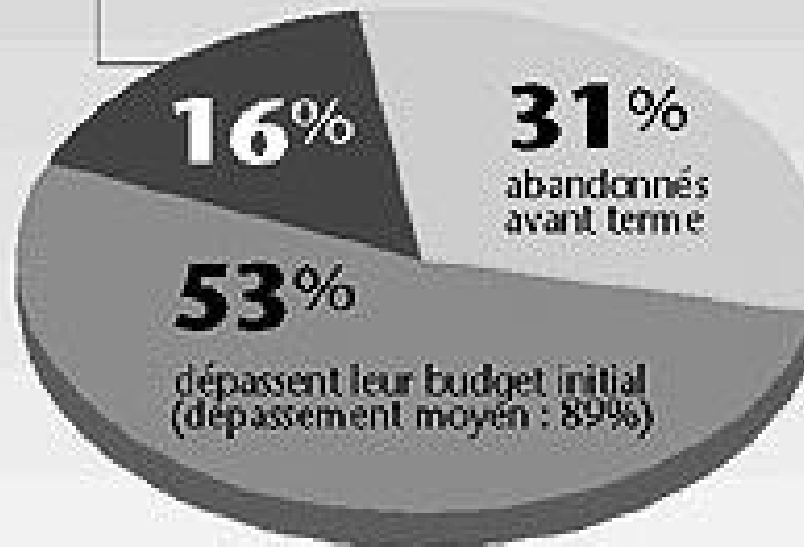
# Augmentation exponentielle de la taille du logiciel



# Sort des projets informatiques

## Le sort des projets informatiques

Ne remplissent pas toutes les fonctions prévues  
(42% des fonctions en moyenne)



Etude portant sur 8 380 projets américains réalisée par le Standish Group en 1995

# Gestion de la complexité due à la taille :

## diviser pour résoudre

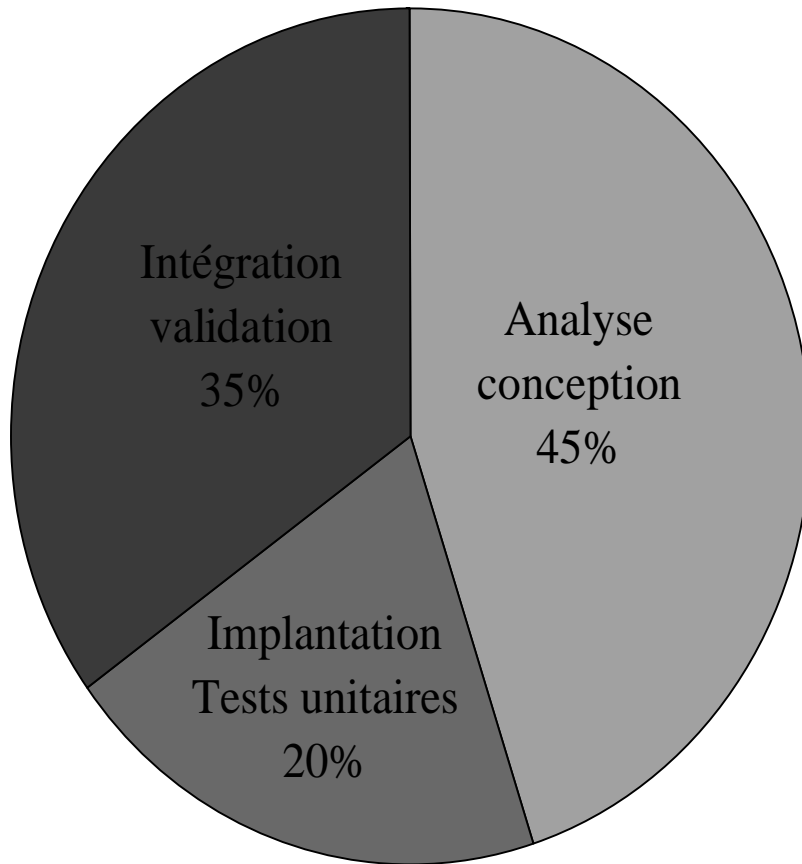
### ■ Aspects techniques

- en s'appuyant techniquement sur la modularité
  - » Encapsulation et masquage d'information, faible couplage

### ■ Aspects organisationnels

- S'organiser au delà de la réalisation : méthodes
  - » Analyse, conception
  - » Validation et vérification
- problèmes de gestion de ressources (humaines, etc.) et de synchronisation entre tâches
- Ingénierie de la conduite de projet = compromis entre
  - » respect des délais
  - » respect des coûts
  - » réponse aux besoins/assurance qualité

# Coûts du développement



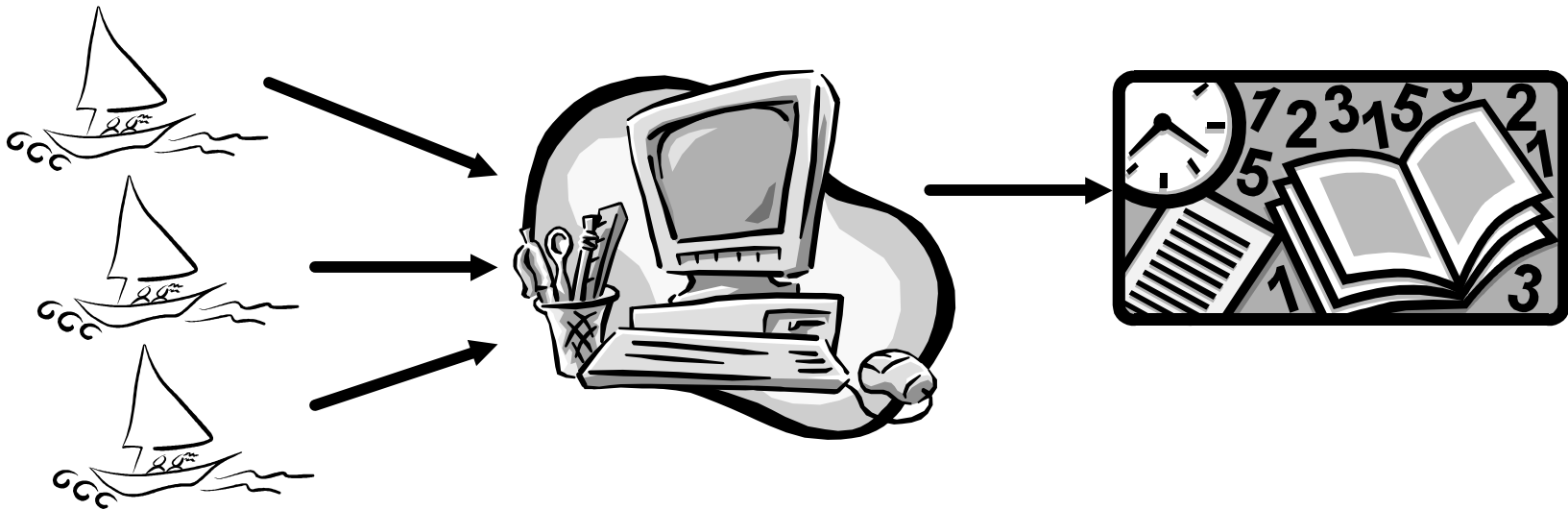
Et la proportion  
Analyse/Conception  
+ Intégration/Validation  
tend à augmenter avec  
la taille du logiciel

# Programming-in-the-Duration

- Gestion de l'évolution d'un système
- D'après Swanson & Beath (Maintaining Information Systems in Organizations, 1989)
  - Durée de vie moyenne d'un système : 6.6 ans
  - 26% des systèmes sont âgés de plus de 10 ans
- Problème de la maintenance corrective et évolutive
  - Adaptation aux changements des besoins
  - Adaptation aux changements de l'environnement
    - » Support nouvelles plate-formes, bug « an 2000 »
- Notion de ligne de produits
  - Anticiper des familles de besoins

# Problème de la maintenabilité

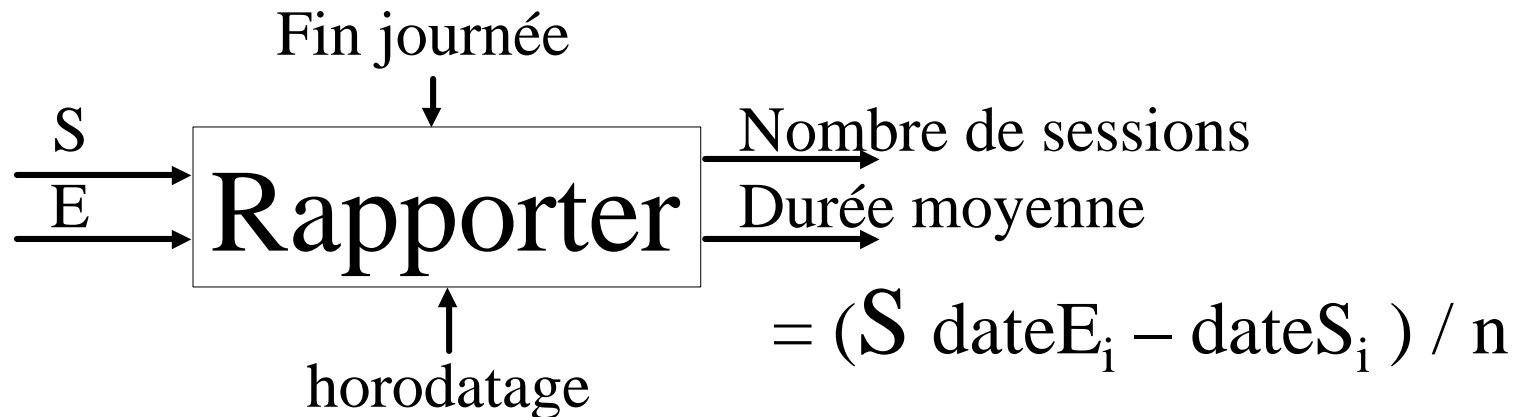
- Exemple critique dû à M. Jackson
- Barques à louer sur un lac
  - Enregistrement des départs (S) et retours (E) sur un terminal, avec horodatage automatique



- On veut chaque jour un rapport avec :
  - le nombre de sessions
  - la durée moyenne d'une session

# Approche « droit au but »

- Décomposition fonctionnelle en vogue circa 1980
- Structured Analysis and Design technique : SADT
  - Le système a une fonction...
  - Décomposée récursivement en sous-fonctions...
    - » ... Jusqu'à tomber sur des fonctions élémentaires (cf. Pascal)
    - » Reliées par flots de données
  - Approche modulaire et hiérarchique
    - » Permet de gérer toute taille de problème
- Expression SADT



# Réalisation

```
■ begin
  open message stream;
  get message ;
  number := 0; totaltime := 0
  do while not end-of-stream
    if code = `S' then
      number := number + 1 ;
      totaltime := totaltime - starttime ;
    else
      totaltime := totaltime + endtime ;
    endif
    get message ;
  enddo
  print `NUMBER OF SESSIONS = `, number ;
  if number / = 0 then
    print `AVERAGE SESSION TIME= `, totaltime / number ;
  endif
  close message stream ;
end
```

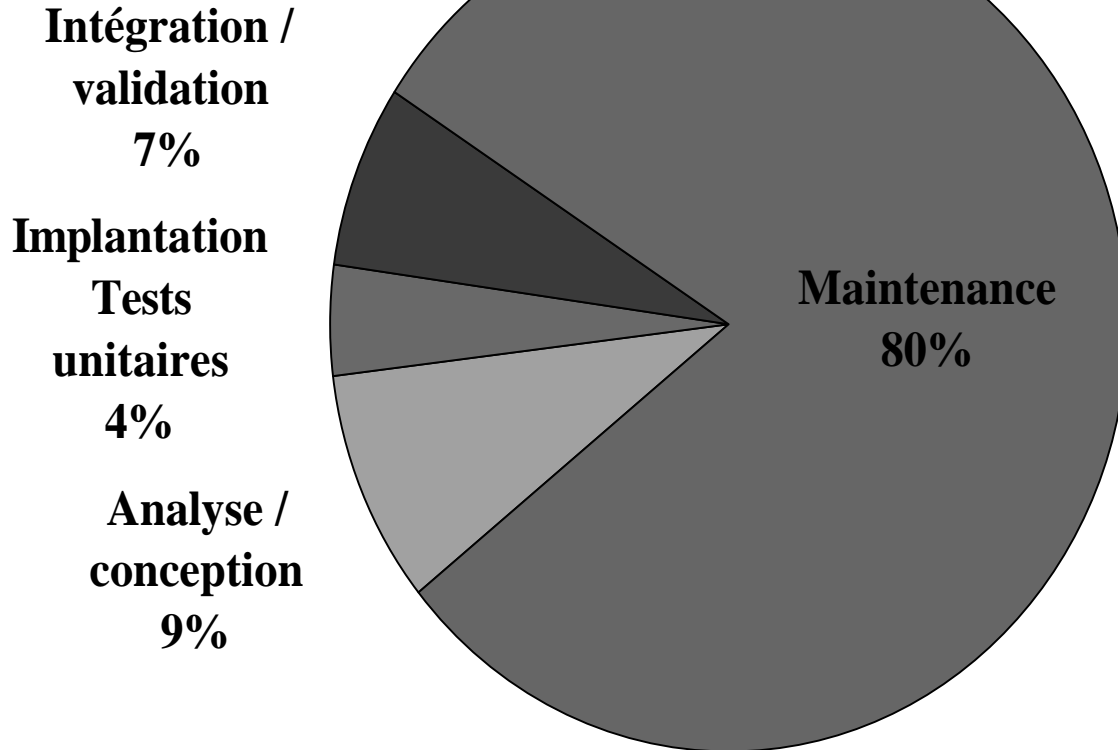


# Maintenance

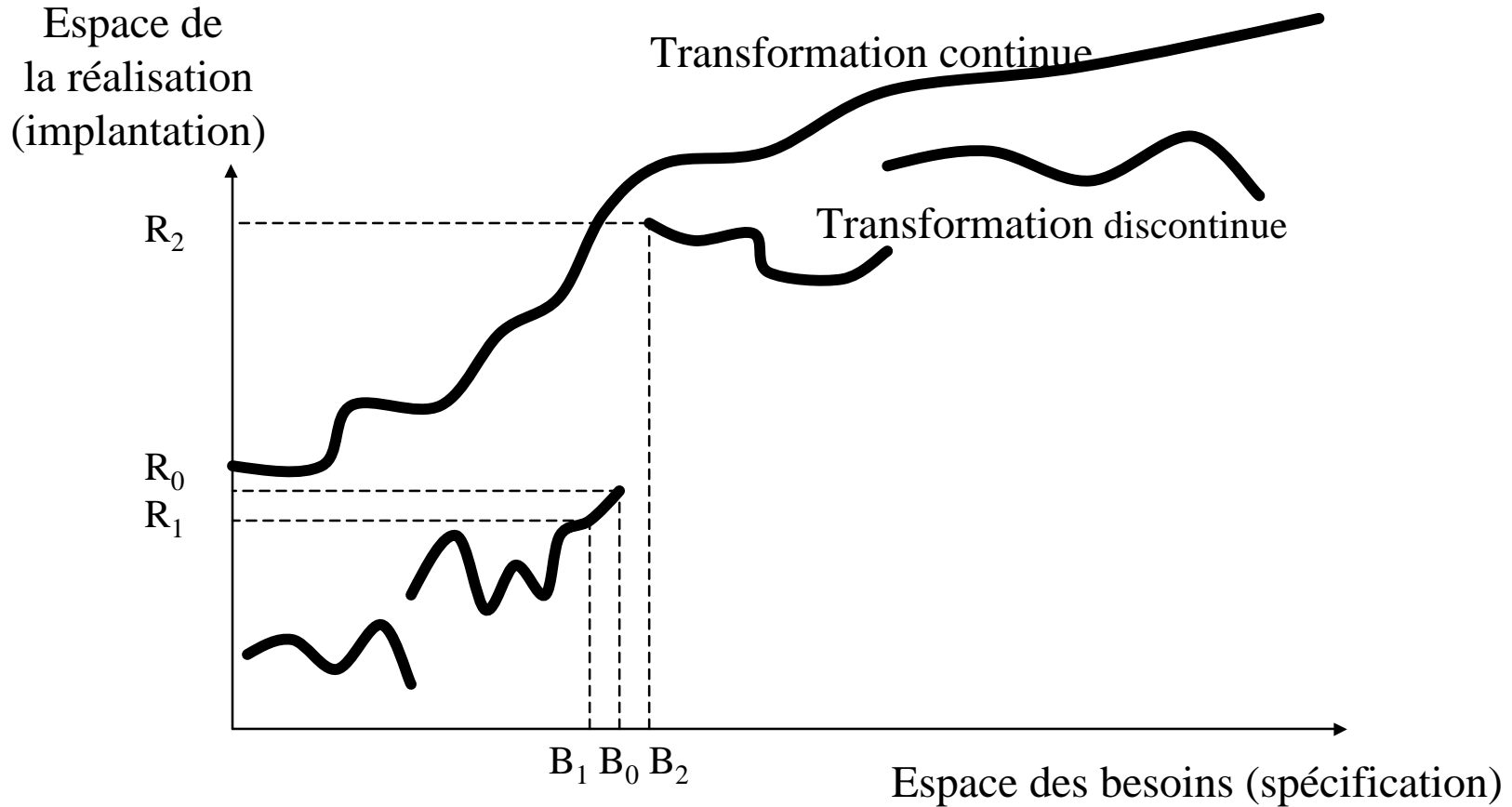
- Demandes de modifications :
  - Nombre de sessions depuis le début de la journée
  - durée de la session la plus longue
  - un état pour le matin, un autre pour le soir
  - corriger la perte de S ou de E
- Dans la réalité, c'est pire!
  - Nokia rapporte à propos de son infrastructure GSM
    - » 50% des requirements (besoins numérotés) ont changé *après* le gel du cahier des charges
    - » 60% de ceux-ci ont changé au moins 2 fois!
  - C'est le cas général plutôt que l'exception
    - » Cahier des charges figé = rêve des années 70

# Coût de la maintenance

Jusqu'à 4 fois le coût de développement initial!



# Probleme de continuité dans le développement



# Solution : approche par modélisation

## ■ Aspects techniques

- D'abord modéliser ce qui est stable dans un système
  - » Entités et événements selon Jackson (JSD)
- Ensuite ajouter les fonctionnalités
  - » C'est la partie visible, donc ce qui bouge le plus
  - » Maintenance traitée idem développement initial
- Assurer une meilleure continuité entre
  - » Domaine du problème
  - » Domaine des solutions

## ■ Aspects organisationnels

- Traçabilité
- Gestion contrôlée des changements