IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

CheckNames

Vérification de la construction de logiciels par l'usage des noms

A. Beugnard, J. Mallet
(P4S/Lab-STICC)

25 février 2022

Les architectes, les développeurs :

- Produisent des interfaces. . .
- Écrivent des programmes. . .
- Écrivent des tests. . .
- Configurent. . . définissent des constantes, des adresses, des ressources (texte, audio, images, vidéos, etc.)
- Déploient tout ce qu'on a fait dans l'environnement de production. . . et donc changent les configurations

Comment s'assurer que rien n'est oublié ?
Comment automatiser ?

Ils utilisent des *programmes* de *build* (make, ant, gradle, etc.)

- La construction de logiciels s'appuie sur de nombreux langages
  - Prog. : C, C++, Java, Elixir, Scala, ...
  - *Build* : make, ant, maven, gradle,...
  - Config : XML, json, txt, ...
- Le seul point commun ? ils *définissent* et *utilisent* des noms
  - de fonction, de variable, de constante, ...
  - de machine, de fichier, d'utilisateur, ...
  - d'adresse, d'écran, de widgets (IHM), ...
  - etc.

Chaque langage peut faire ses propres vérifications

... mais rien n'existe entre langages.

- Le *build* est au cœur du développement
  - Relie les métiers SysAdmin, DevOps, DevSecOps, Dev
  - Relie les artéfacts (sources, config, ressources)
- Les *builds* sont source de nombreuses erreurs [ATdSdM07, MAN+11, SSE+14, KKA14, IZ17, RHLS17, VSZ+17, MMPJ18]

Vision renforcée par un article de 2020 qui introduit la notion de *Dependency Bugs* [FNFSW20].

Constat

Il n'y a pas d'outils de vérification d'usage des noms.

But du projet

Montrer qu'une approche d'analyse des noms translangage permet de réduire les risques d'erreur lors de la construction des applications.

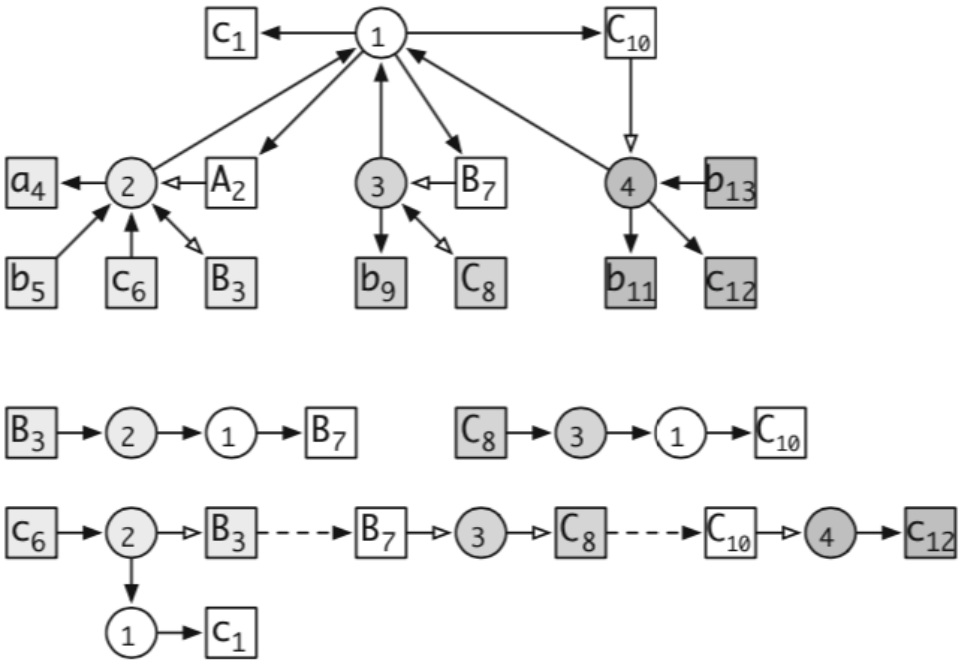
- Fournir une aide (analyse, vérification, diagnostique) globale, systématique et multi-langage de la sûreté du *build* pour les DevSecOp, Dev, DevOp.
- Réduire le coût de mise au point des *builds*
- Réduire le coût d'exécution des *builds*

Approche

- Un modèle abstrait de nommage (définitions, références des noms), générique pour les sources, ressources, artéfacts ...
- Une analyse statique des noms et de leurs usages, depuis toutes les sources de noms, permettant de réduire le nombre d'erreurs de construction et d'exécution.
- Bénéfices
  - Base formelle
  - Diagnostique d'usage des noms (erreurs, alertes, . . .)
  - Automatique
  - Extensible / adaptable
- Limite
  - Pas de calcul sur les noms eux-mêmes

- L'analyse sémantique des liens lors de la compilation et de l'édition de liens. [Interne à un langage.]
- Une *Theory of Name Resolution* existe [NTVW15, vANT+16] pour plusieurs langages (mais indépendamment les uns des autres)

```
def c1 = 4
module A2 {
  import B3
  def a4 = b5 + c6
}
module B7 {
  import C8
  def b9 = 0
}
module C10 {
  def b11 = 1
  def c12 = b13
}
```



1. Extraire les noms, leur(s) valeur(s) associée(s) et leurs utilisations
2. Construire un arbre de nommage global
3. Vérifier et diagnostiquer
4. Capitaliser les propriétés à vérifier selon les contextes d'utilisation et les moments de vérification

- Identification d'applications
- Apport ? Difficile sans toute l'application
- Développement d'un prototype
- Application à un cas simple
- Discussion avec des architectes

Constat

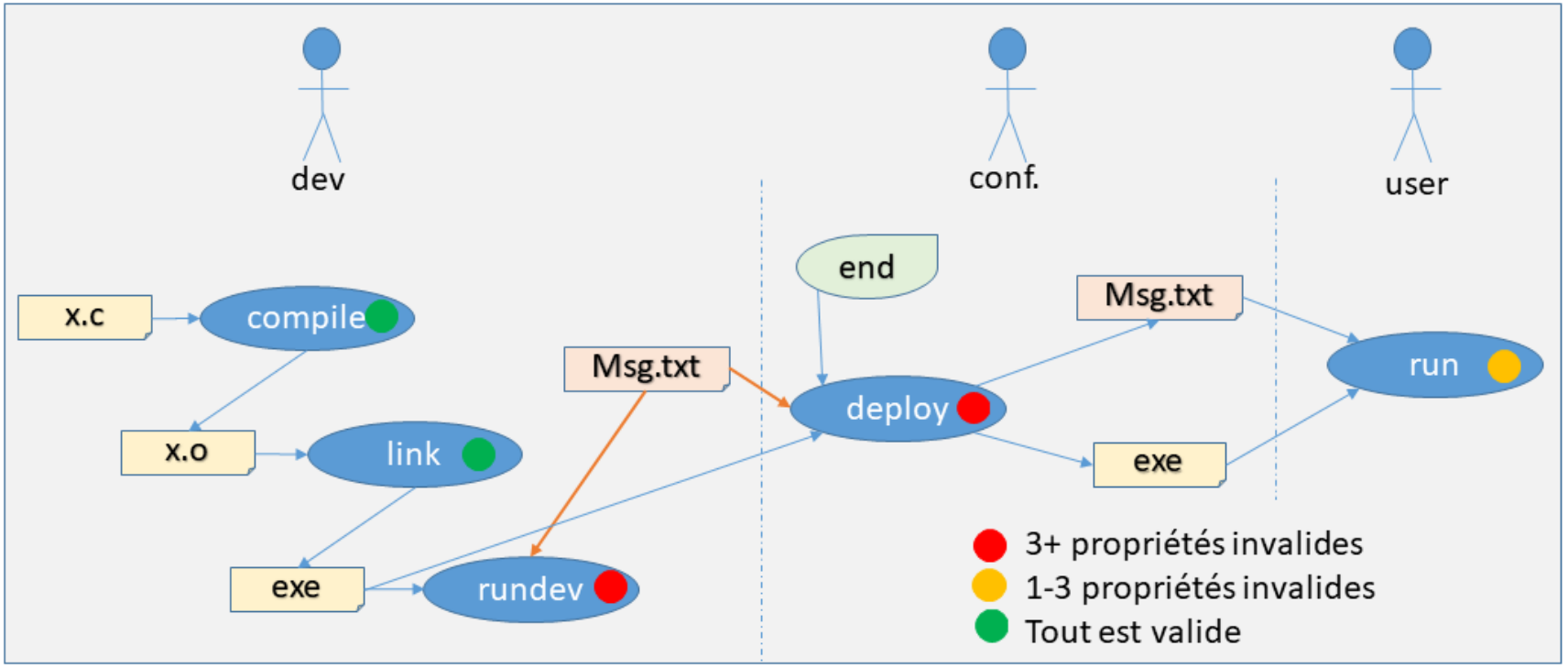
La mise au point des programmes de *build* est complexe et coûteuse.

- Fichiers source en C (.c .h)
- Un fichier de ressource (un message à afficher qui sera configuré au déploiement)
- Un makefile

Les étapes

- Compilation, édition de lien
- Exécution (sans configuration)
- Déploiement et configuration
- Exécution (après configuration)

Un tableau de bord d'intégration continue, abstrait, multi-utilisateur



- Confirmation que la mise au point *des* scripts de *build* est difficile et source d'erreurs
- Réalisation d'une maquette démontrant la faisabilité de l'approche

Nous avons écrit un sujet de thèse sur ce thème qui nous semble un bon moyen de poursuivre le travail. . .

Objectif

Réduire les coûts et les délais de mise au point de l'intégration continue.

- *Various stakeholders* interact with a build system [ATdSdM07]
- People are *puzzled by the complex composition* of the build scripts [ATdSdM07]
- Historicaly, build tools have suffered from a lot of problems hampering *understandability and maintainability* [ATdSdM07]
- The specification of dependencies is a good thing, *manually* managing them is tedious and *error-prone* [ATdSdM07]
- *Build maintenanc*e could impose from 12%-36% overhead on software development [KE02]
- Role of various *non-source artifacts* [ATdSdM07]
- The *variables introduced by the configuration* system aggravate things, as problems may be tied to certain configurations only [ATdSdM07]

- Build files grow quickly and become very complex because they must support the building of the same software in multiple platforms with various configuration and environment parameters [HJ11]

Ref.	Builds (durée) *échecs*	Sources principales d'erreurs
[KKA14]	3214 (6 mois) *18%*	missing referenced files, mistakenly checking in work-in-progress, transitive dependencies
[SSE+14]	26.6M (9 mois) *37.4% (C++); 29.7% (Java)*	cant.resolve, doesnt.exist, undeclared_var_use
[SP16]	7200 () *38%*	dependencies, Java compilation, doc. generation, configuration, file not found

[VSZ+17, IZ17, RHLS17] analysent les phases causes d'erreurs (test, documentation, outils externes, etc.)

Bram Adams, Herman Tromp, Kris de Schutter, and Wolfgang de Meuter.
Design recovery and maintenance of build systems.
In *International Conference on Software Maintenance*, pages 114–123. IEEE, October 2007.

Anders Fischer-Nielsen, Zhoulai Fu, Ting Su, and Andrzej Wąsowski.
The forgotten case of the dependency bugs : On the example of the robot operating system.
In *The 42nd International Conference on Software Engineering : Software Engineering in Practice ((ICSE-SEIP)*, 2020.


L. Hochstein and Y. Jiao.
The cost of the build tax in scientific software.
In *ESEM '11*, pages 384 – 387, 2011.


Md Rakibul Islam and Minhaz F. Zibran.
Insights into continuous integration build failures.
In *14th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM, May 2017.


G. Kumfert and T. Epperly.
Software in the doe : The hidden overhead of "the build".
Technical Report UCRL-ID-147343, Lawrence Livermore National Laboratory, 2002.


Noureddine Kerzazi, Foutse Khomh, and Bram Adams.
Why do automated builds break ? an empirical study.
In *IEEE International Conference on Software Maintenance and Evolution*. IEEE, September 2014.


Shane McIntosh, Bram Adams, Thanh H.D. Nguyen, Yasutaka Kamei, and Ahmed E. Hassan.
An empirical study of build maintenance effort.
In *33rd International Conference on Software Engineering (ICSE)*. IEEE, October 2011.


Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones.
Build Systems à la Carte.
Proceedings of the ACM on Programming Languages, 2(ICFP) :1–29, July 2018.


Pierre Neron, Andrew Tolmach, Eelco Visser, and Guido Wachsmuth.
A theory of name resolution.
In *European Symposium on Programming Languages and Systems*, pages 205–231. Springer, 2015.


Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte.
An Empirical Analysis of Build Failures in the Continuous Integration Workflows of Java-Based Open-Source Software.
In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 345–355. IEEE, 2017.


Matúš Sulír and Jaroslav Porubän.
A quantitative study of java software buildability.
In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2016, pages 17–25, New York, NY, USA, 2016. ACM.


Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge.
Programmers' build errors : A case study (at google).
In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 724–734, New York, NY, USA, 2014. ACM.


Hendrik van Antwerpen, Pierre Néron, Andrew Tolmach, Eelco Visser, and Guido Wachsmuth.
A constraint language for static semantic analysis based on scope graphs.
In *PEPM'16*, pages 49 – 60, St. Petersburg, FL, USA, 2016.


Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Massimiliano Di Penta Andy Zaidman, and Sebastiano Panichella.
A tale of ci build failures : an open source and a financial organization perspective.
In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, page 183–193, Shanghai, China, 2017. IEEE Computer Society.