RECOMMENDING WHEN DESIGN TECHNICAL DEBT SHOULD BE SELF-ADMITTED

Computer Engineering

Cedric Noiseux

December 5, 2017

Table of Contents I

Table of Contents II

Concepts and Definitions

Technical Debt (TD)

Definition

TDs are not quite right code which we postpone making it right [1].
They are temporary and suboptimal solutions or workarounds.

Reasons

Quickly fix an issue
Early conception stages
Lack of comprehension, skill or experience

Concepts and Definitions

Technical Debt (TD)

Types

Design	Object-Oriented design principles
Requirement	Functional and non functional requirement trade-offs
Code	Quality of source code
Test	Quality of testing activities
Documentation	Misleading Information
Defect	Known and possible defects

Impacts

Slower conception and execution

Lower maintainability and quality

Higher production cost

Concepts and Definitions

Self-Admitted Technical Debt (SATD)

Definition

SATDs are technical debts which are consciously introduced and explicitly documented by developers.

Characteristics

Diffusion	31% of files contain SATD, 51 instances per system
Evolution	Persistent and increase with time
Actors	Experienced developers introduce them

Elements of the Problematic

Self-Admitted Technical Debt (SATD)

Issues

Weak Performance Results

Ineffective Strategies

- Code Refactoring
- Continuous Tracking
- Proactiveness

Flawed or Incomplete Approaches

- Pattern Matching
- Machine Learning with Natural Language Processing

Objectives

Main Research Question

How can we identify and detect technical debts in a software project using source code or features and known self-admitted technical debts with a machine learning approach?

TEDIOUS	CNN
Metrics and Warnings	Source Code
Machine Learners	Neural Network

Objectives

Application Objectives

Recommendation System

To encourage self-admitting technical debts

To ease tracking

To fix issues faster

Complement to existing smell detectors

Or use as an alternative

Tracking **Managing** **Improving**

Objectives

Design Objectives

Defining and Extracting Features

TEDIOUS : Metrics and Warnings

CNN : Source Code

Identifying SATDs

Preprocessing Features

TEDIOUS : Multi-Collinearity, Feature Selection, Normalization, Rebalancing

CNN : Tokenization, Word Embedding

Building and Applying Machine Learners

TEDIOUS : Machine Learning Models

CNN : Neural Network

Table of Contents I

Study Definition

Studied Projects

Project	Number of F	C	M	Com	Nb of Com ∈ M	Nb of SATD ∉ M	∈ M	SATD M %
Ant	1,113	1,575	11,052	20,325	13,359	1	57	0.5%
ArgoUML	1,922	2,579	14,346	64,393	17,722	203	425	2%
Columba	1,549	1,884	7,035	33,415	10,305	8	418	5%
Hibernate	2,129	2,529	17,405	15,901	9,073	21	377	2%
jEdit	394	889	4,785	15,468	10,894	6	77	2%
jFreeChart	1,017	1,091	10,343	22,827	15,412	4	1,881	18%
jMeter	1,048	1,328	8,680	19,721	12,672	95	424	5%
jRuby	970	2,063	14,163	10,599	7,809	16	275	2%
Squirrel	2,325	4,123	16,648	25,216	15,574	35	173	1%

F : Files C : Classes M : Methods Com : Comments

Study Definition

Research Questions

RQ1

How does TEDIOUS work for recommending SATD within project?

RQ2

How does TEDIOUS work for recommending SATD within project?

RQ3

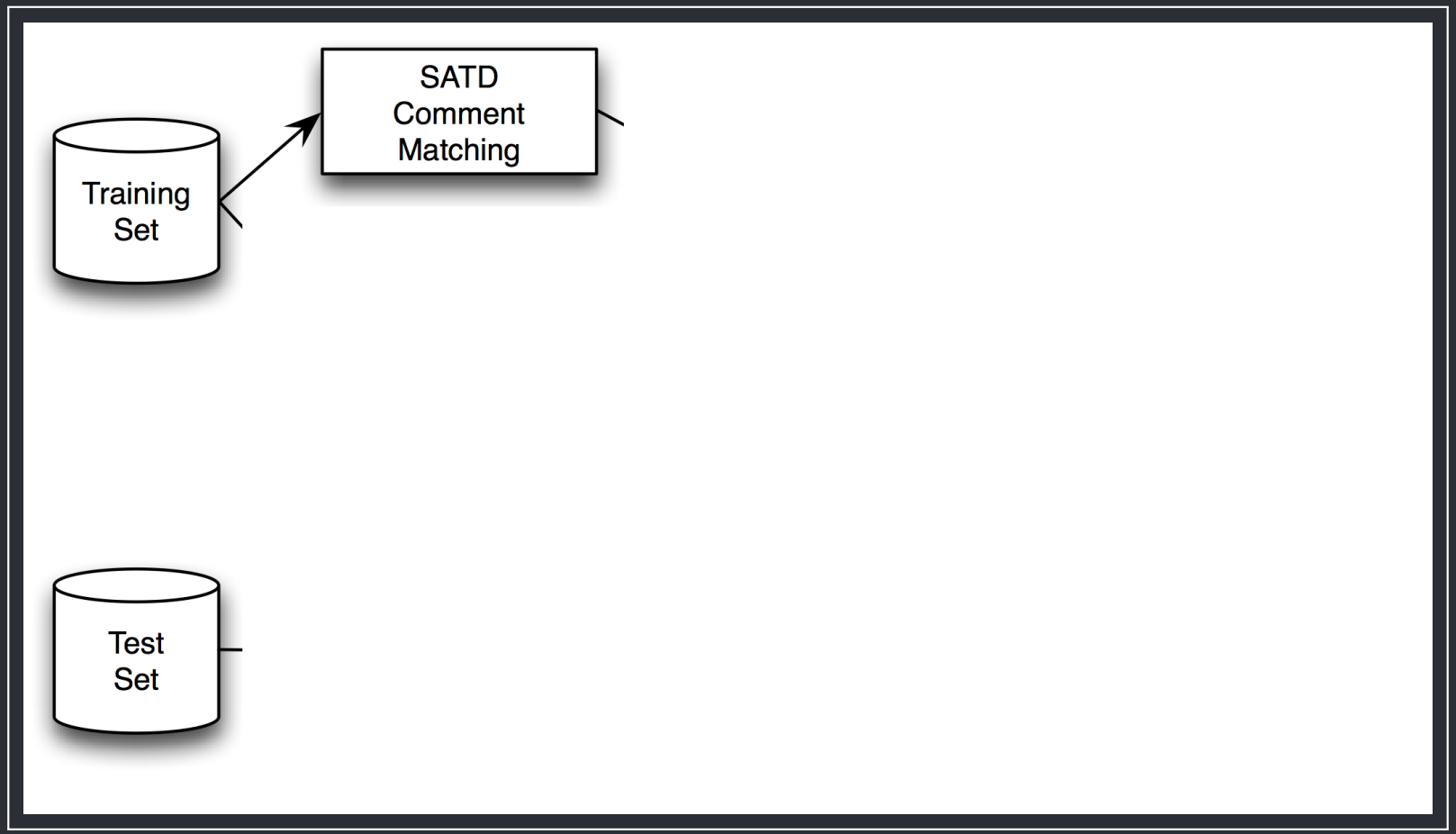
How would a method-level smell detector compare with TEDIOUS?

The Approach

Train and Test Sets

The Approach

SATD Matching



The Approach

SATD Matching

Identification of Self-Admitted Technical Debt

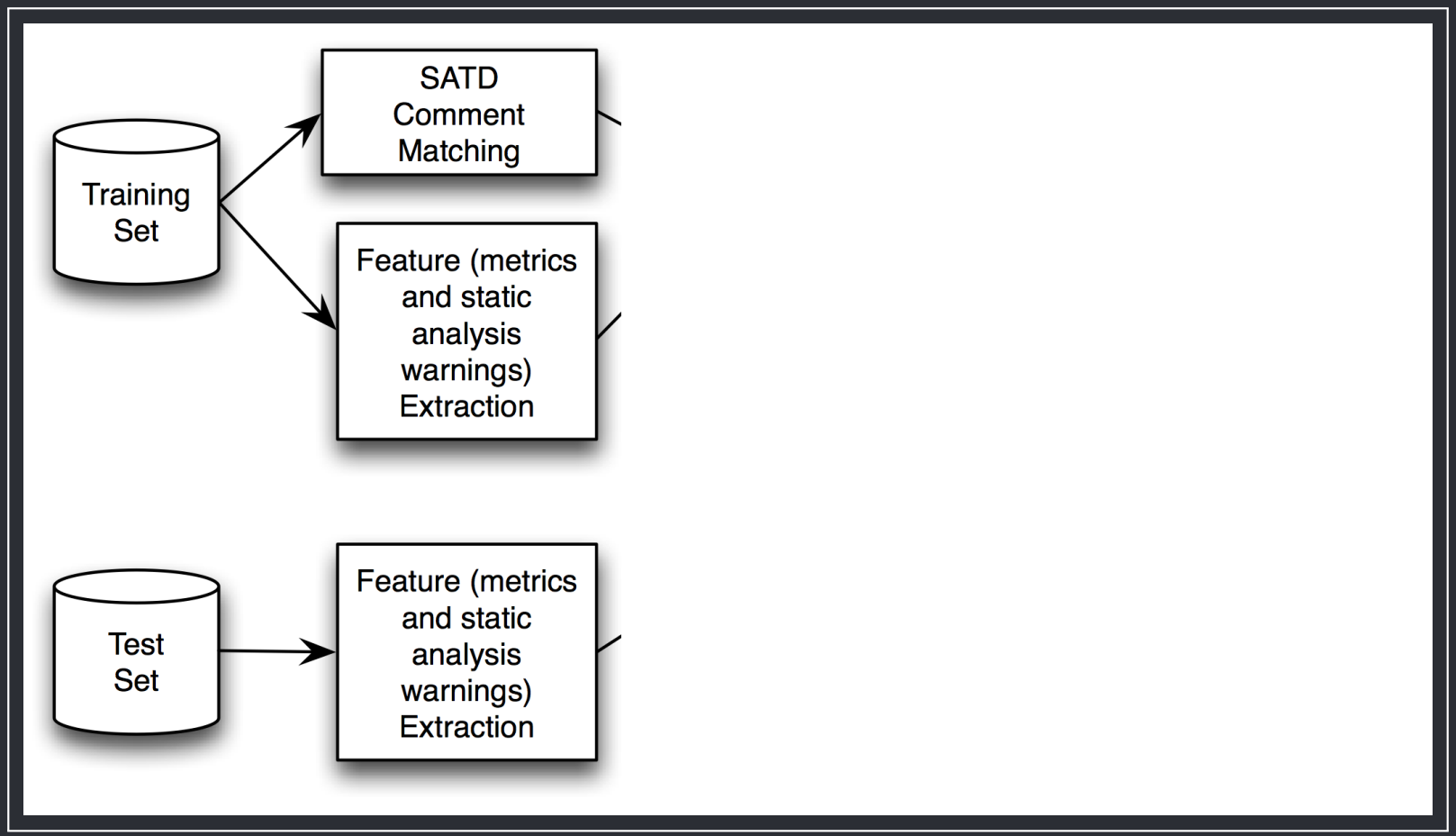
Method-Level

Pattern Matching

Tagging

The Approach

Features Extraction



The Approach

Features Extraction

```
<comment type="block" format="javadoc" pos:line="106" pos:column="5">/**
 * Get account using the email address to identify it.
 *
 * @param address
 *            email address
 * @return account item
 */</comment>
<function><type><specifier pos:line="113" pos:column="5">public</specifier> <name pos:line="113" pos:column="12">AccountItem</name></type> <name pos:
line="113" pos:column="24">getAccount</name><parameter_list pos:line="113" pos:column="34">(<param><decl><type><name pos:line="113" pos:column="35">
String</name></type> <name pos:line="113" pos:column="42">address</name></decl></param>)</parameter_list> <block pos:line="113" pos:column="51">{
```

Source Code Metrics

XML Representation

Remove SATD Methods

Compute source code and readability metrics

The Approach

Features Extraction

List of Metrics

LOC

Nb of Statements

Nb of Comments

McCabe Cyclomatic Complexity

Nb of Expressions Nb of Passsed Param

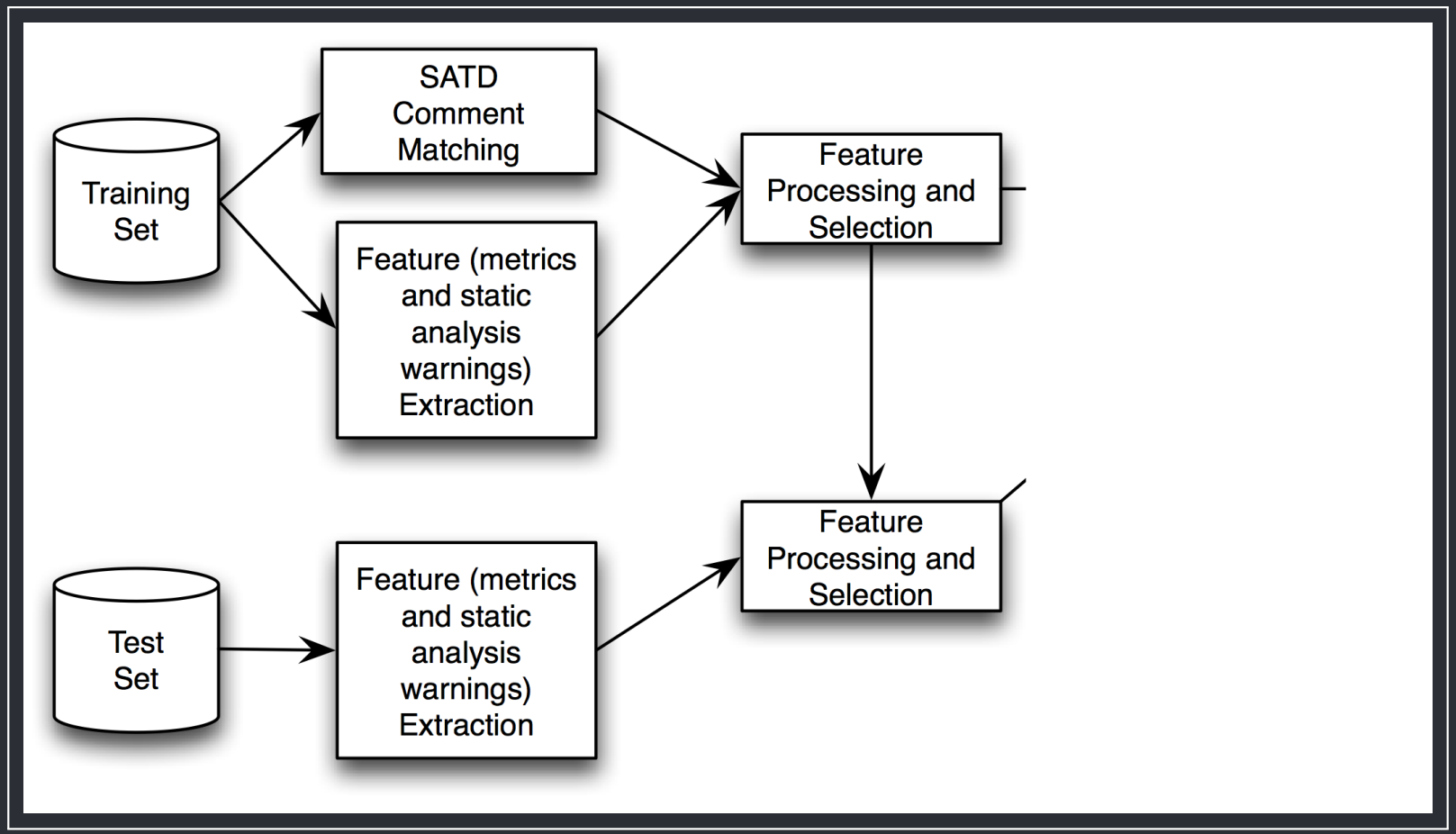
Nb of Identifiers

Nb of Call Sites

Nb of Declarations

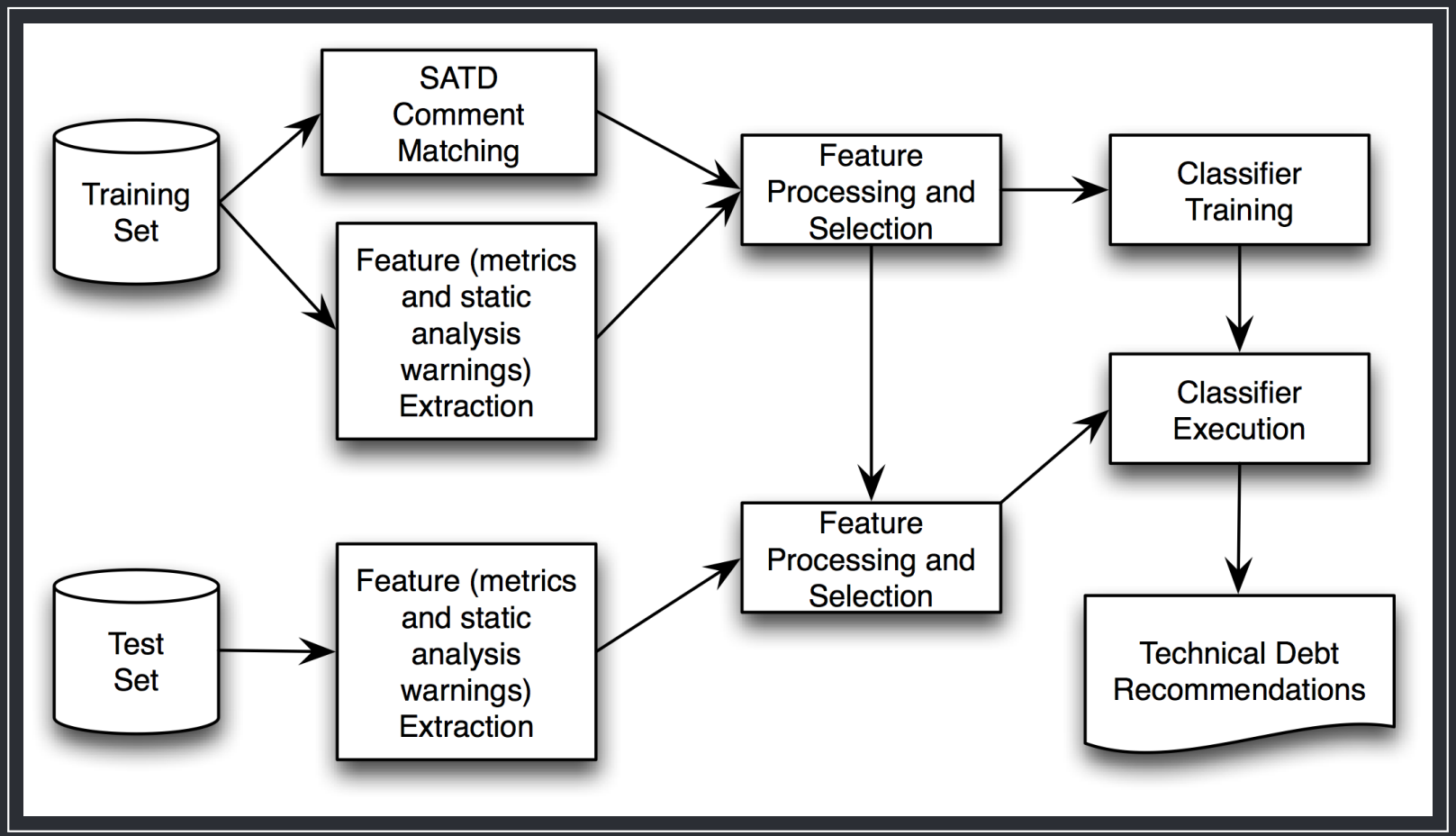
The Approach

Feature Preprocessing



The Approach

Machine Learners



Method-level Image du process (separer en differents process et expliquer) - Training and test sets - SATD matching - Features - Preprocessing - Building and applying machine learners (training and testing)
Analysis Method Within project performance Cross project performance TEDIOUS vs method level Qualitative analysis
Inspirations (Kim, Dos Santos) Description
Method-level Image du process (separer en differents process et expliquer) - Training and test sets - SATD matching - Source code - Preprocessing/Word Embedings - Building and applying machine learners (training and testing)
Research questions (main and 3 sub-questions) 9 OOS + Java More method-related than class-related Unbalance
Source code comments only Source code with comments Source code without comments Source code partially with comments
Construct Internal Conclusion Reliability External
Number of metrics/warning Number of LOC Process inherent errors

Default configurations (no real optimizations) Generalization of results Only in Java Performance evaluation can be better for CNN TEDIOUS approach CNN approach Dataset TEDIOUS results CNN results Applicability scenarios - Recommendation system - Complement to smell detectors Improvements - Optimization - Pattern matching process - More examples to train - More positive examples - More metrics/warnings and source code - Other TD types - Extend to other languages and domains

Bibliography

TEX, LATEX, and Beamer

[1] W Cunningham. *The wycash portfolio management system*. dans Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), série OOPSLA '92, New York, NY, USA: ACM, 1992, pp 29–30, DOI: 10.1145/157709.157715. En ligne: http://doi.acm.org/10.1145/157709.157715.