Automated prediction of delays in software development projects

Hoa Khanh Dam

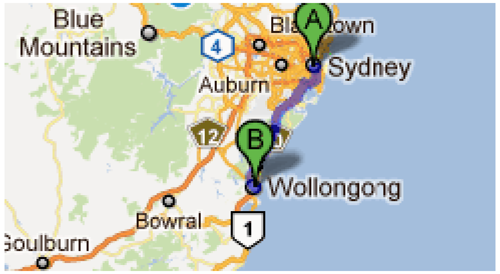
UNIVERSITY OF WOLLONGONG

Who am I?

UNIVERSITY OF WOLLONGONG

- Dr Hoa Dam, Senior Lecturer in Software Engineering at University of Wollongong.

- Where is Wollongong?



Who am I? (cont.)

- Research interests and expertise:
  - Data-driven Software Engineering (e.g. applications of data mining and machine learning into software engineering)
  - Change management & inconsistency management
  - Model-driven development and evolution
  - Agent-oriented software engineering
  - Service-oriented engineering
  - Business process management

Email: hoa@uow.edu.au

Web: http://www.uow.edu.au/~hoa

What am I talking about today?

- Predictive models
- Building models for predicting software delays
  - Local classification models
  - Relational classification models

- Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran and Aditya Ghose, *Characterization and prediction of issue-related risks in software projects,* Proceedings of 12th Working Conference on Mining Software Repositories (MSR), co-located with ICSE 2015, pages 280 - 291, IEEE (ACM SIGSOFT Distinguished Paper Award)

- Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran and Aditya Ghose, *Predicting delays in software projects using networked classification,* Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Preprint copies of these papers available on my website.

Predictive models – a (very!) brief introduction

- Example: the weather problem – condition for playing a game.

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	Normal	False	Yes
…	…	…	…	…

```
If outlook = sunny and humidity = high then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes
```

Predictive models – a (very!) brief introduction (cont.)

- How do we (automatically) build a model for predicting if a game is played?
- Supervised learning:
  - a machine learning task of inferring a **function** from labelled **training data**
    - ***f(outlook, temperature, humidity, windy)*** returns true or false.
    - Features: input variable, e.g. outlook, temperature, etc.
    - Target/dependent variable, e.g. play = yes or no
    - Training set consists of training examples
  - Regression vs. classification

Predictive models – a (very!) brief introduction (cont.)

- Some basic learning models (learners)
  - **Naïve Bayes**: probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	No		Yes	No		Yes	No		Yes	No	Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Training set

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Source: **Data Mining: Practical Machine Learning Tools and Techniques,** *3rd Edition by Ian H. Witten, Eibe Frank, Mark A. Hall*

Predictive models – a (very!) brief introduction (cont.)

- Some basic learning models (learners)
  - **Naïve Bayes**: probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Evidence E = instance

Event H = class value for instance

$$Pr[H|E] = \frac{Pr[E_1|H]Pr[E_2|H]\ldots Pr[E_n|H]Pr[H]}{Pr[E]}$$

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Likelihood of the two classes

For "yes" = 2/9 × 3/9 × 3/9 × 3/9 × 9/14 = 0.0053

For "no" = 3/5 × 1/5 × 4/5 × 3/5 × 5/14 = 0.0206

Conversion into a probability by normalization:

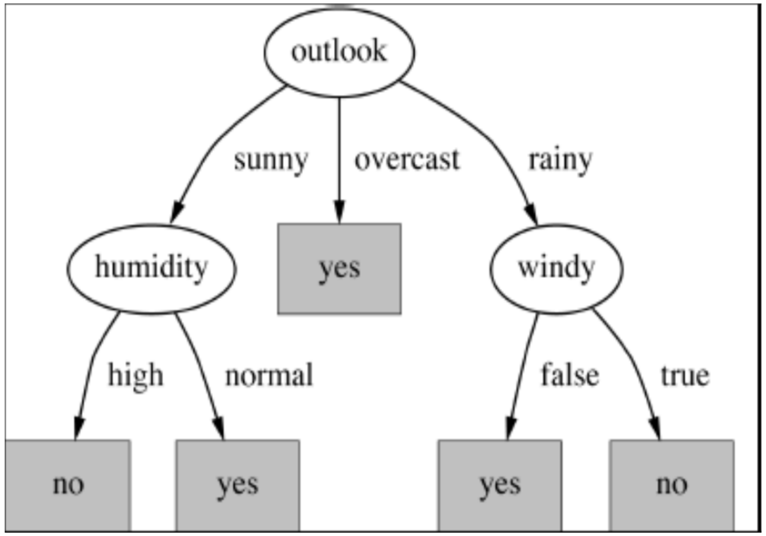
P("yes") = 0.0053 / (0.0053 + 0.0206) = 0.205

P("no") = 0.0206 / (0.0053 + 0.0206) = 0.795

Source: **Data Mining: Practical Machine Learning Tools and Techniques,** *3rd Edition by Ian H. Witten, Eibe Frank, Mark A. Hall*

Predictive models – a (very!) brief introduction (cont.)

- Some basic learning models (learners)
  - **Decision Trees**



Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Source: **Data Mining: Practical Machine Learning Tools and Techniques,** *3rd Edition by Ian H. Witten, Eibe Frank, Mark A. Hall*

Predictive models – a (very!) brief introduction (cont.)

- Some advanced learning models (learners)
  - Random Forests (RF)
    - An ensemble learning method
    - A significant improvement of the decision tree approach
    - Generating many classification trees, each of which is built with random subset of variables at each node split, and aggregates into the individual results using voting
  - Neural Networks
  - Logistic Regression

These models were used in our papers but are not covered today.

What am I talking about today?

- Predictive models √
- Building models for predicting software delays
  - Local classification models
  - Relational classification models

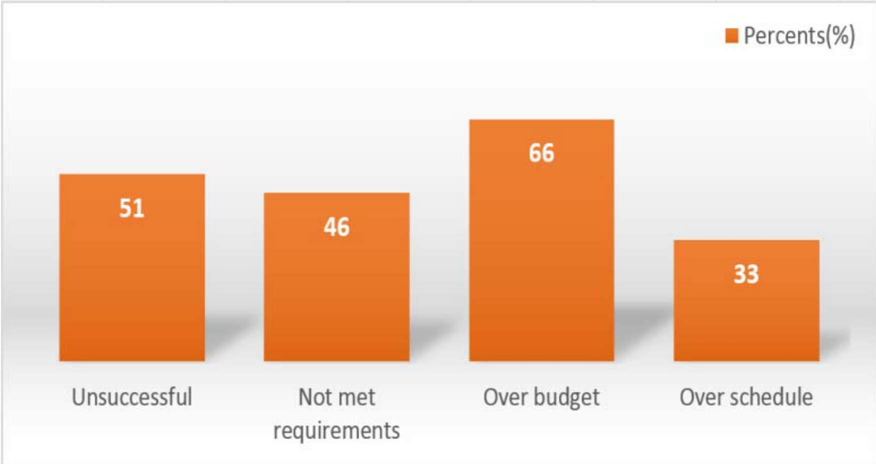
- Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran and Aditya Ghose, *Characterization and prediction of issue-related risks in software projects,* Proceedings of 12th Working Conference on Mining Software Repositories (MSR), co-located with ICSE 2015, pages 280 - 291, IEEE (ACM SIGSOFT Distinguished Paper Award)

- Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran and Aditya Ghose, *Predicting delays in software projects using networked classification,* Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Preprint copies of these papers available on my website.

Motivation

- Study on 5,400 IT projects in 2012 by McKinsey and the University of Oxford



- Standish Group's CHAOS report
  - 82% software projects missed schedules

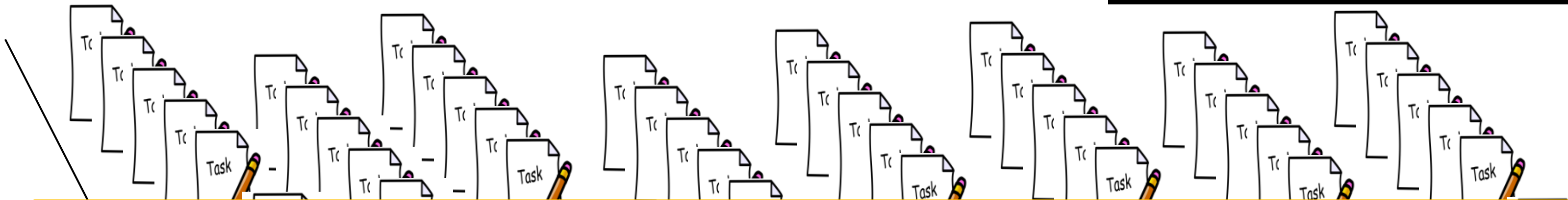
⇒Predicting risks causing delay is critical in software project management.

Motivation (cont.)

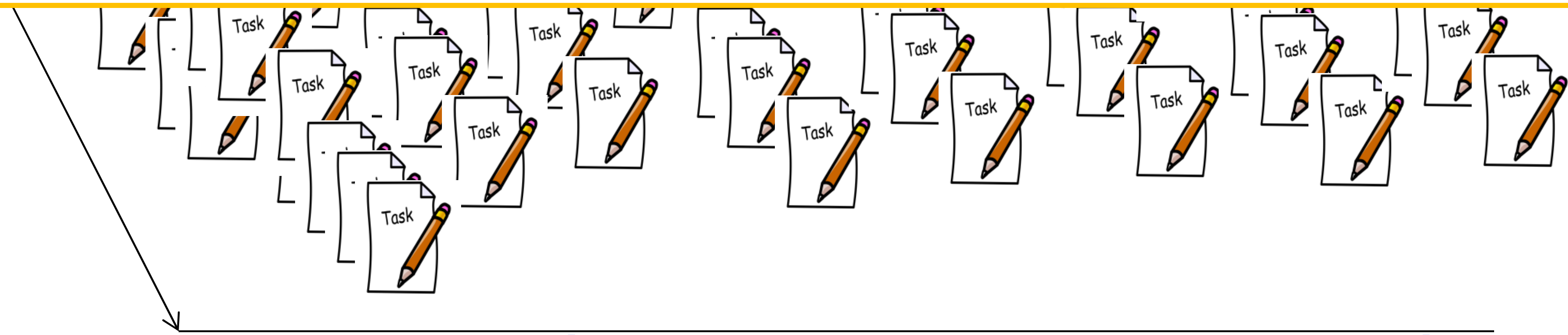
- Current practices in software risk management mostly rely on:
  - high-level, generic guidance (e.g. Boehm's "top 10 list of software risk items" or SEI's risk management framework); or
  - subjective expert judgements

⇒There is a gap in providing *data-driven*, *actionable* support for project managers in risk management.

What exactly is the problem we were trying to solve?



Which of these tasks would be a delay risk (i.e. causing a release delay)?



Milestone (e.g. release 2.1 on June 2)

How did we (attempt to) solve it?

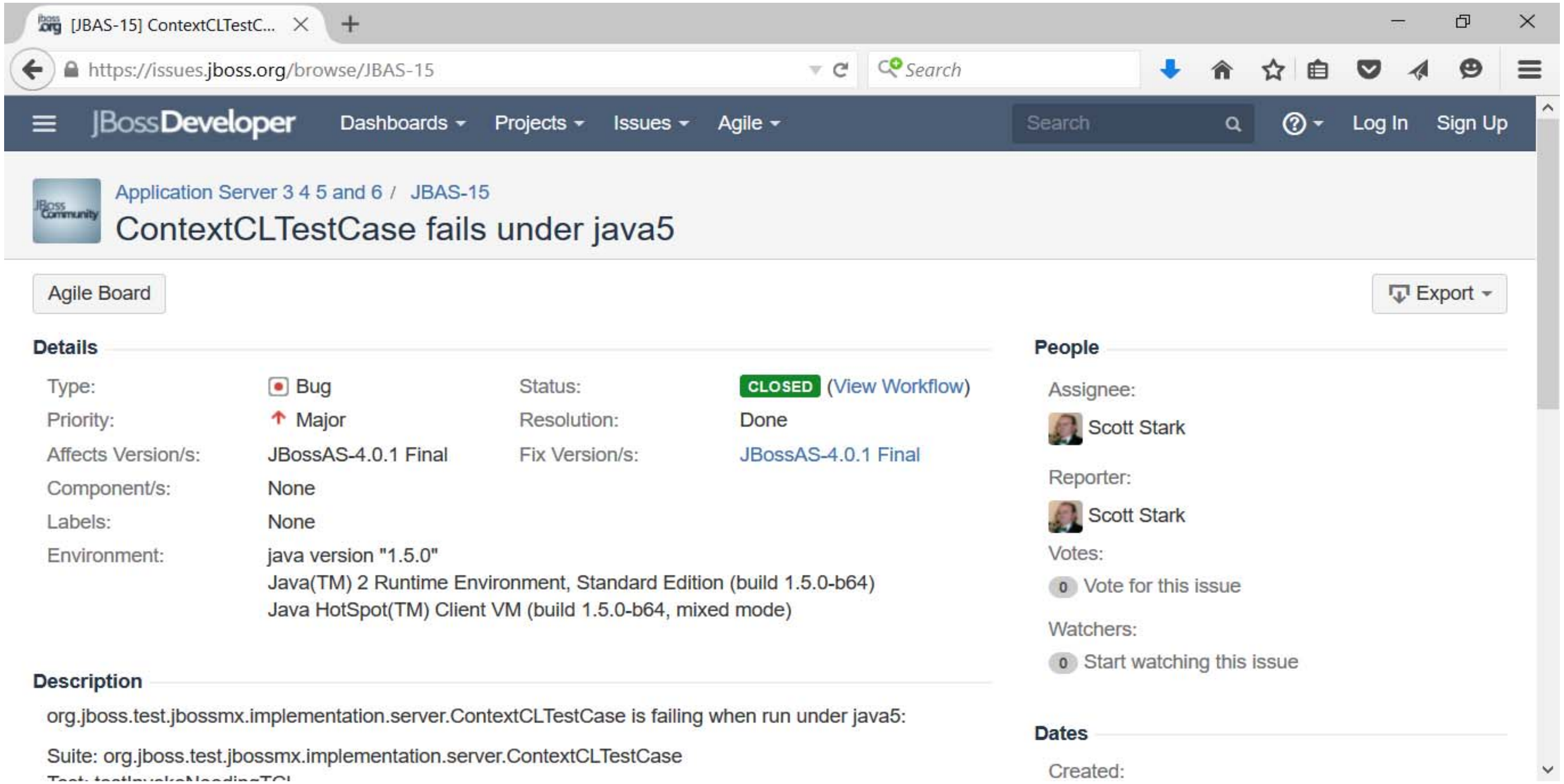
- Step 1: Characterizing the software tasks (**issues**) that constitute a risk of delay
  - Feature extraction: from 40,830 past issues in Moodle, JBoss, Apache, Duraspace, and Spring.

TABLE I: Dataset description

Project	Delayed issues		Non-delayed issues		Total
Apache	111	[2.27%]	4,771	[97.72%]	4,882
Duraspace	314	[9.75%]	2,908	[90.25%]	3,222
Jboss	2,242	[15.08%]	12,627	[84.92%]	14,869
Moodle	214	[2.24%]	9,325	[97.75%]	9,539
Spring	80	[0.96%]	8,238	[99.03%]	8,318
Total	2,961	[7.25%]	37,869	[92.74%]	40,830

How did we solve it? (cont.) Feature extraction

Extract 16 risk factors (features) from 40,830 issues collected from five open source projects: Apache, Duraspace, JBoss, Moodle, and Spring



https://issues.jboss.org/browse/JBAS-15

How did we solve it? (cont.) Feature extraction

Extract 16 risk factors (features) from 40,830 issues collected from five open source projects: Apache, Duraspace, JBoss, Moodle, and Spring

1. Discussion time
2. Waiting time
3. Type
4. Number of times that an issue is reopened
5. Priority
6. Changing of priority
7. Number of comments
8. Number of fix versions
9. Number of affect versions
10. Number of issue links
11. Number of issues that are blocked by this issue
12. Number of issues that block this issue
13. Changing of description
14. Reporter reputation
15. Developers' workload
16. Percentage of delayed issues that a developer involved with

How did we solve it? (cont.)

- Step 2: selected features are used to train five classifiers: Random Forests, Neural Networks, Decision Tree (C4.5), Naïve Bayes, and NBTree.
  - Predicting the **risk impact** (i.e. degree of delay): *non-delayed*, *major delayed*, and *minor delayed* (multi-class classification)
  - Predicting the **likelihood** of a risk occurring, i.e. the chance of an issue causing delay
  - Predicting **risk exposure**

$$\bar{RE}_i = C_1 P(i, Non) + C_2 P(i, Min) + C_3 P(i, Maj)$$

How do we evaluate it?

- Training set vs. Test set
- Performance measures:
  - Precision, recall, F-measure (**today**)
  - Area Under the ROC curve (AUC), Macro-averaged Mean Cost-Error (MMCE), and Macro-averaged Mean Absolute Error (**see details in the papers**)

Performance measures

- The confusion matrix is usually used to store the correct and incorrect decisions made by a a trained classifier.
- For example, if a task is classified as delayed when it was truly delayed, the classification is a **true positive** (tp).
- If the task is classified as delayed when actually it was not delayed, then the classification is a **false positive** (fp).
- If the task is classified as non-delayed when it in fact caused a delay, then the classification is a **false negative** (fn).
- Finally, if the task is classified as non-delayed and it in fact did not cause a delay, then the classification is **true negative** (tn).

Precision, Recall and F-measure

- Precision: the ratio of correctly predicted delayed task over all the tasks predicted as delayed task.

$$pr = \frac{tp}{tp + fp}$$

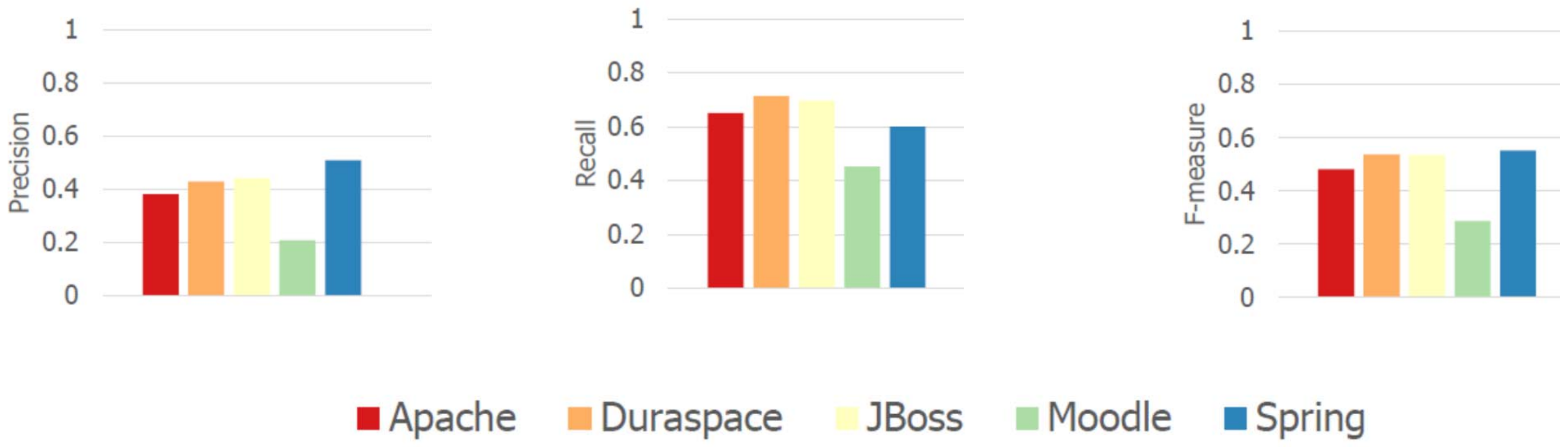
- Recall: The ratio of correctly predicted delayed task over all of the actually delayed tasks.

$$re = \frac{tp}{tp + fn}$$

- F-measure: measures the weighted harmonic mean of precision and recall.

$$F - measure = \frac{2 * pr * re}{pr + re}$$

Results



How can we improve this result?
Which data have we not used yet?

Improved approach using network data

- The above approach employ traditional machine learning classification techniques to perform classification on each task **independently** using its attributes or features.

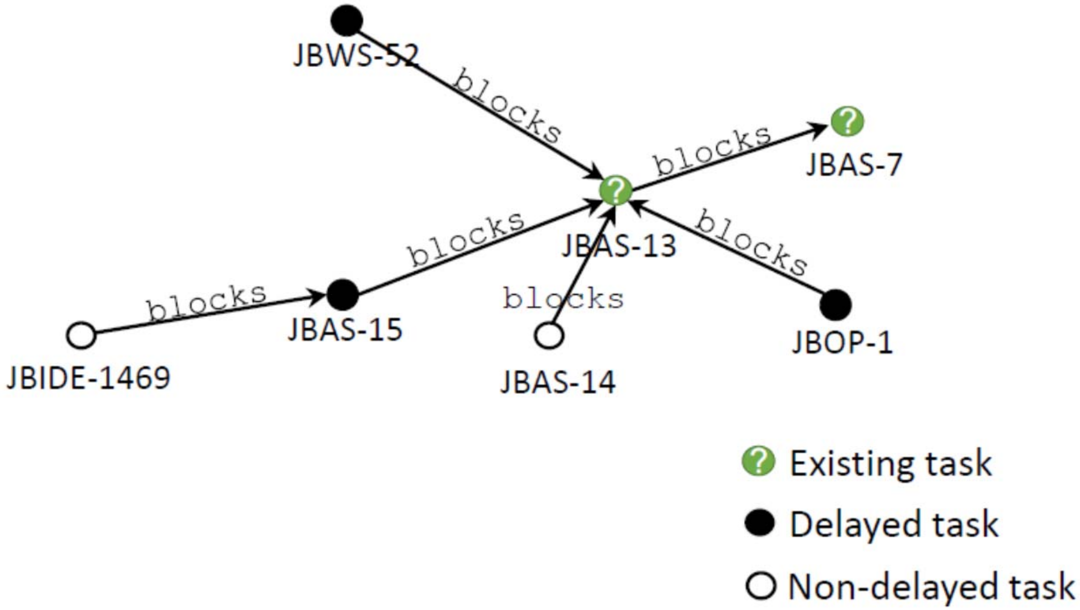
- Such approaches do not take into account the role of the underlying network of **inter-relationships** between software tasks.
  - Task dependencies in software projects are **predominant** – approximately 57% of the tasks in the five open source projects selected were related to at least one other task.
  - These task dependencies form the **networked data**

Networked data

- Networked data are seen in many different forms in our daily life,
  - E.g. hyperlinked Web pages, social networks, communication networks, biological networks and financial transaction networks.
  - They are used in various applications:
    - E.g. classifying Web pages, scientific research papers, protein interaction and gene expression data.

- A similar class of networked data (i.e. networked tasks) can also provide valuable information for predicting delays in software projects.

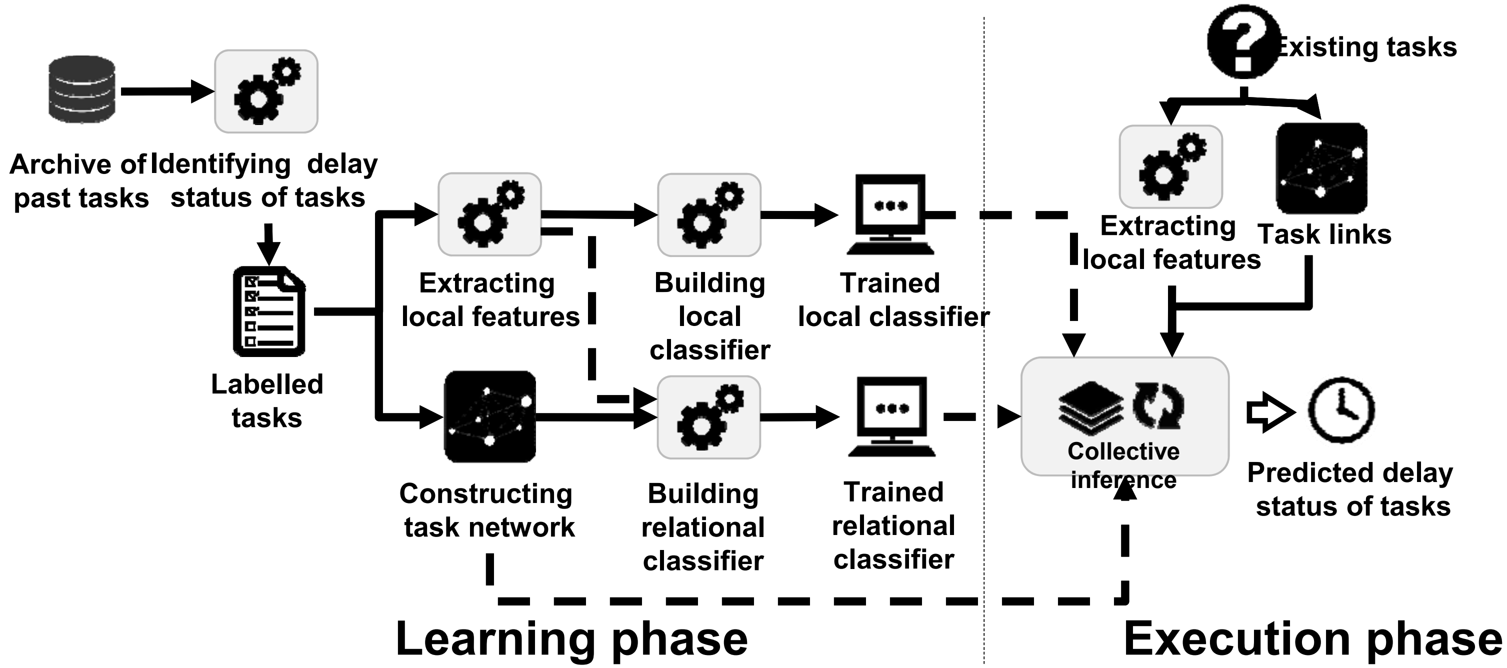
Example: tasks in JBoss



For example, if a task blocks another task and the former is delayed, then the latter is also at risk of getting delayed.

This example demonstrates a common **delay propagation** phenomenon in (software) projects, which has not been considered by previous approaches.

Overview of the approach



Task network construction

Definition 1 (Task network). *A task network is a directed graph $G = (V, E)$ where:*

- *each vertex $v \in V$ representing a software task in the form of $\langle ID, c, attrs \rangle$ where ID is a unique identifier of the task, c is the risk class, i.e. label (e.g. non-delayed, minor delayed or major delayed), which the task belongs to, and $attrs$ is a set of the task's attribute-value pairs $(attr_i, val_i)$ (i.e. local features).*

- *each edge $e \in E$ representing a link between tasks u and v in the form of $\langle \langle u, v \rangle, types, weights \rangle$ where $types$ is set of the link's type and $weigths$ is set of link's weight.*

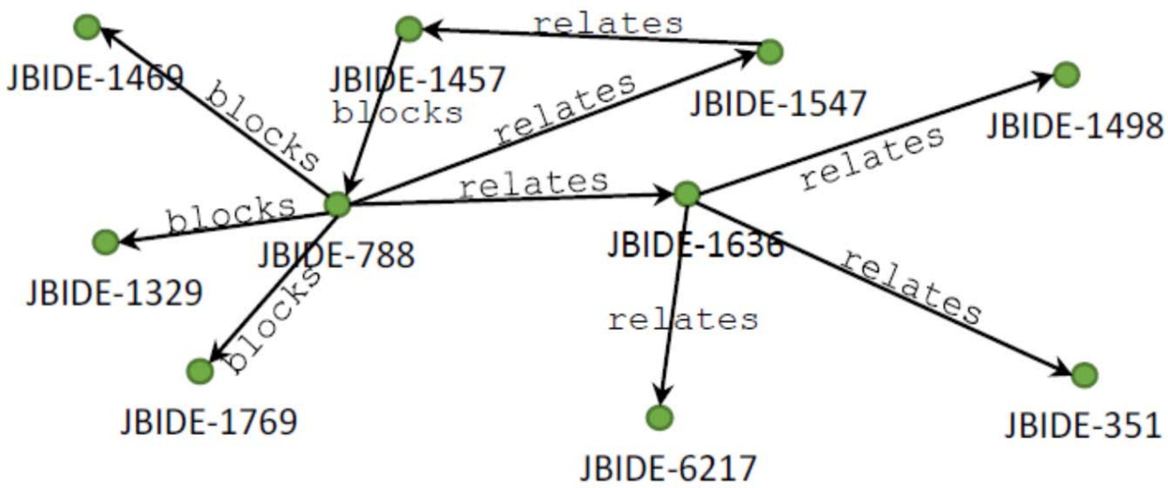
Explicit relationships

- There are a number of dependencies among tasks which are explicitly specified in the task records.
  - These typically determine the order in which tasks need to be performed.

- There are generally four different types of relationships of the preceding tasks to the succeeding tasks:
  - finish to start (predecessor must finish before successor can start),
  - start to start (predecessor must start before successor can start),
  - finish to finish (predecessor must finish before successor can finish),
  - and start to finish (predecessor must start before successor can finish).

Explicit relationships (cont.)

- For example, blocking is a common type of relationships that is explicitly recorded in issue/bug tracking systems.
  - Blocking tasks are software tasks that prevent other tasks from being resolved, which could fall into the finish to start or finish to finish category.



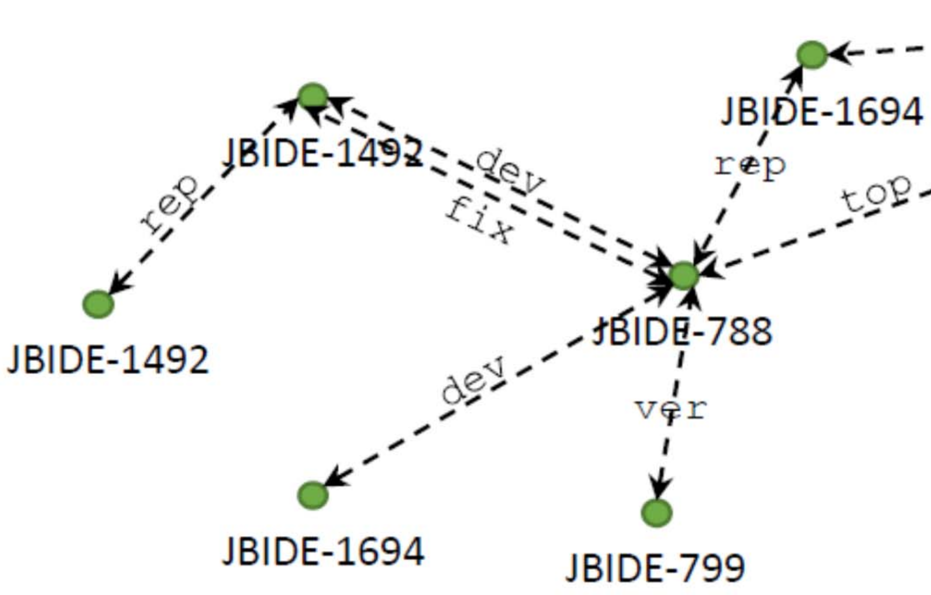
Implicit relationships

- While explicit relationships are specified directly in the task reports, implicit relationship need to be inferred from other task information.

- There are different task information that can be extracted to identify a (implicit) relationship between tasks.

- We classified them into three groups as described below.
  - **Resource-based relationship**: this type of relationships exists between tasks that share the same (human) resource. The resource here could be the developers assigned to perform the tasks or the same person who created and reported the tasks.
  - **Attribute-based relationship**: tasks can be related if some of their attributes share the same values. For example, there is a relationship between tasks performed on the same component since they may affect the same or related parts of code.

- **Content-based relationship**: tasks can be similar in terms of how they are conducted and/or what they affect.
  - We use Latent Dirichlet Allocation (LDA) to build a topic model representing the content of a software task. We then establish relationships between on the basis that related tasks share a significant number of common topics.



dev = same developer
rep = same reporter
com = same component
fix = same fix version
ver = same affect version
top = same topic

Relational classifiers

- Relational classifiers make use of information about related tasks to estimate the label probability. For simplicity, we use only direct relations for class probability estimation:

$$P(c \mid G) = P(c \mid N_i)$$

where N_i is a set of the immediate neighbors of task v_i (i.e. those that are directly related to v_i) in the task network G, such that $P(c \mid N_i)$ is independent of $G \setminus N_i$.

Relational classifiers (cont.)

- Weighted-Vote Relational Neighbor (wvRN) estimates class membership probabilities based on two assumptions:
  - First, the label of a node depends only on its **immediate** neighbors.
  - Second, wvRN relies on the principle of **homophily** which assumes that neighboring class labels were likely to be the same.

$$P(c \mid v_i) = \frac{1}{Z} \sum_{v_j \in N_i} w(v_i, v_j) P(c \mid N_j)$$

where $Z = \sum_{v_j \in N_i} w(v_i, v_j)$

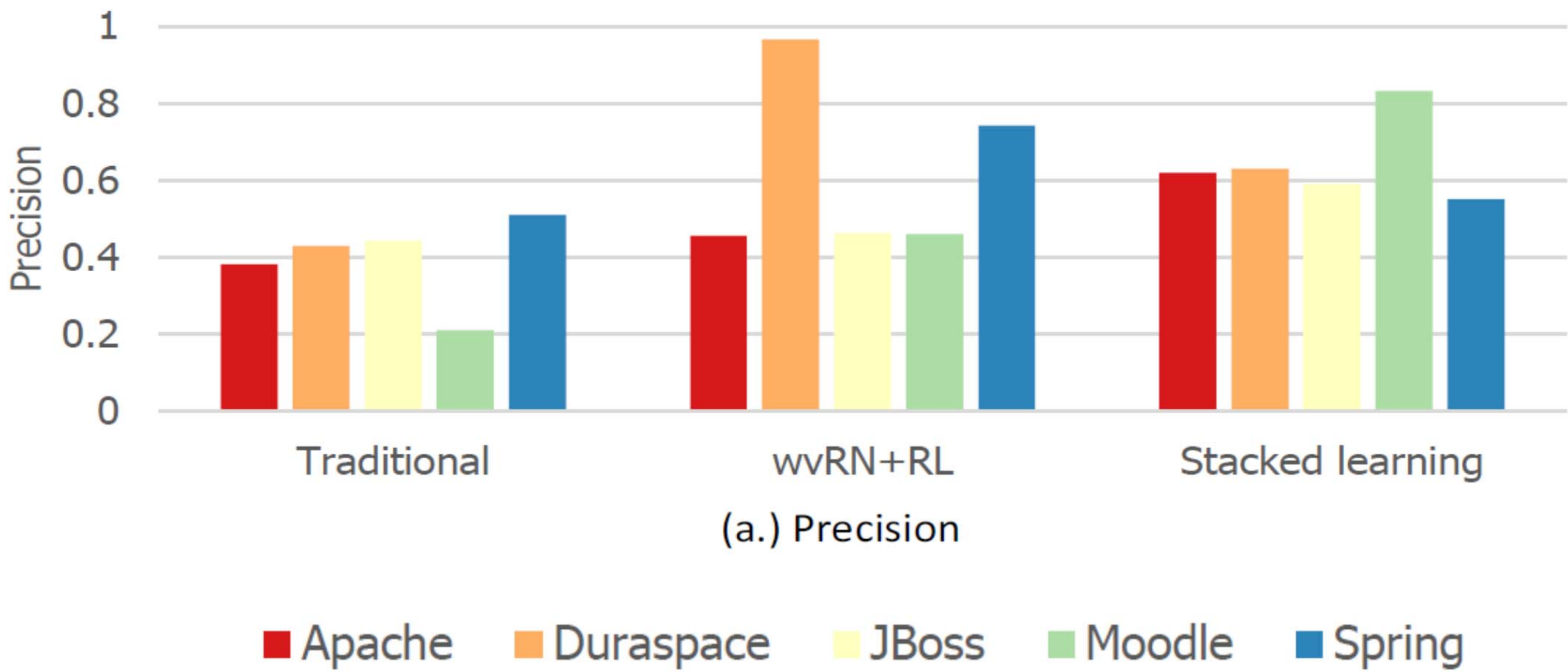
Relational classifiers (cont.)

- Stacked Graphical Learning:
  - One inherent difficulty of the weighted-voting method is the computation of the neighbour weights. Since there are multiple relations, estimating the weights are non-trivial.
  - Stacked learning offers an alternative way to incorporate relational information.
  - The idea of stacking is to learn joint models by multiple steps, taking into relational information of the previous step to improve the current step.
  - At each step, relational information together with local features are fed into a standard classifier (e.g., Random Forests). We consider relations separately and the contribution of each relation is learnt by the classifier through the relational features.
  - The classifier is then trained. Its prediction on all data points (vertices in the network) will be then used as features of the next stage.

Relational classifiers (cont.)

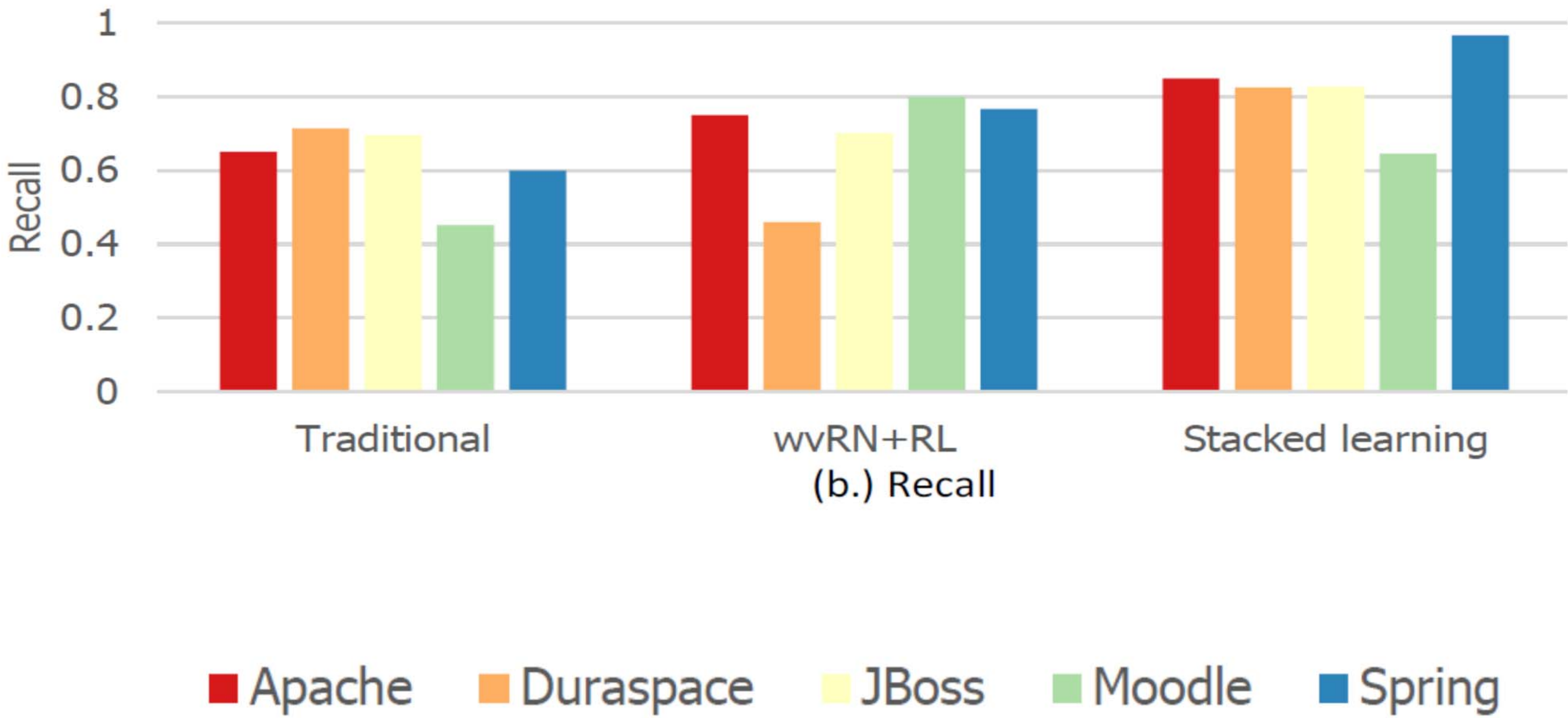
- Collective inference is the process of inferring class probabilities simultaneously for all unknown labels in the network conditioned on the seen labels.
- We employ two methods:
  - Relaxation Labeling (**RL**) and Stacked Inference (SI) – see the ASE paper for more details.

Experimental results

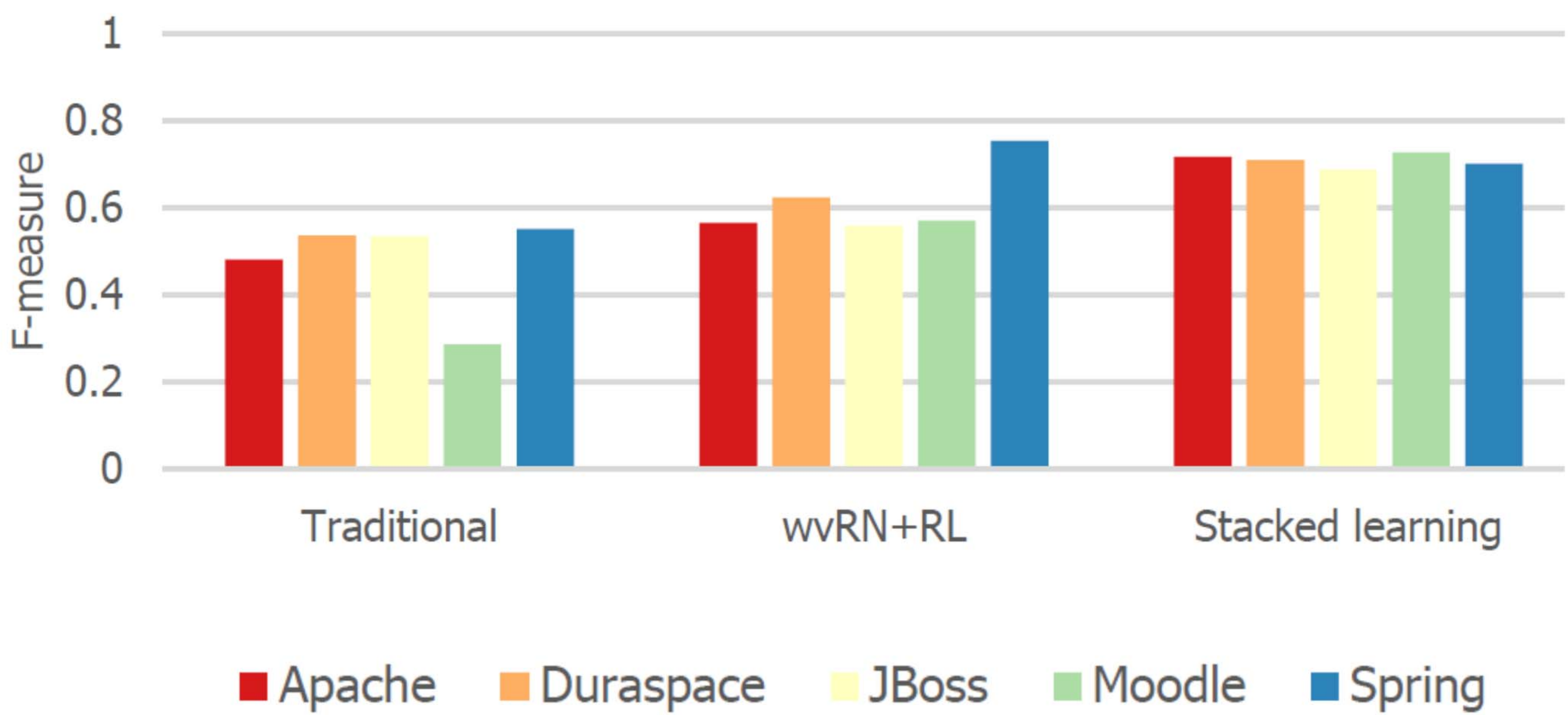


(a.) Precision

Experimental results



Experimental results



Conclusions

- Predictive models
- Building models for predicting software delays
  - Local classification models
  - Relational classification models

- More details in the papers (see http://www.uow.edu.au/~hoa)