Achieving Software Reliability Without Breaking the Budget

Bojan Cukic

Lane Department of CSEE
West Virginia University

University of Houston
September 2013

Software Engineering (I)maturity

- **35% of large applications are cancelled,**
- **75% of the remainder run late and are over budget,**
- ***Defect removal efficiency is only about 85%***
- **Software needs better measures of results and better quality control.**

- ***Right now various methods act like religious cults more than technical disciplines.***
  - *Capers Jones*, Feb. 3, 2012, in Data & Analysis Center for Software (DACS), LinkedIn Discussion Forum

Software Engineering (I)maturity

- **Major cost drivers for software in the U.S., rank order**
  1) The cost of finding and fixing bugs
  2) The cost of cancelled projects
  3) The cost of producing / analyzing English words
  4) The cost of security flaws and attacks
  5) The cost of requirements changes
  6) The cost of programming or coding
  7) The cost of customer support
  …
  11) The cost of innovation and new kinds of software
  12) The cost of litigation for failures and disasters
  13) The cost of training and learning
  14) The cost of avoiding security flaws
  15) The cost of assembling reusable components
- **This list is based on analysis of ~13,000 projects.**
  – *Capers Jones*, Feb. 4, 2012, in DACS

Outline – Software Engineering as Data Science

- **Fault prediction**
  - Early in the life cycle.
  - Lower the cost of V&V by directing the effort to places that most likely hide faults.
- **Effort prediction**
  - With few data points from past projects
- **Problem report triage**
- **Summary**

Software Reliability Prediction

- **Probability of failure given known operational usage.**
  - Reliability growth
    - Extrapolates reliability from test failure frequency.
    - Applicable late in the life cycle.
  - Statistical testing and sampling
    - Prohibitively large number of test cases.
  - Formal analysis
    - Applied to software models
- **All prohibitively expensive**

  -> **Predict where faults hide, optimize verification.**

Fault Prediction Research

- **Extensive research in software quality prediction.**
  - Faulty modules identified through the analysis and modeling of static code metrics.
    - Significant payoff in software engineering practice by concentrating V&V resources on problem areas.
- ***Are all the prediction methods practical?***
  - *Predominantly applied to multiple version systems*
    - *A wealth of historical information from previous versions.*
  - ***What if we are creating Version 1.0?***

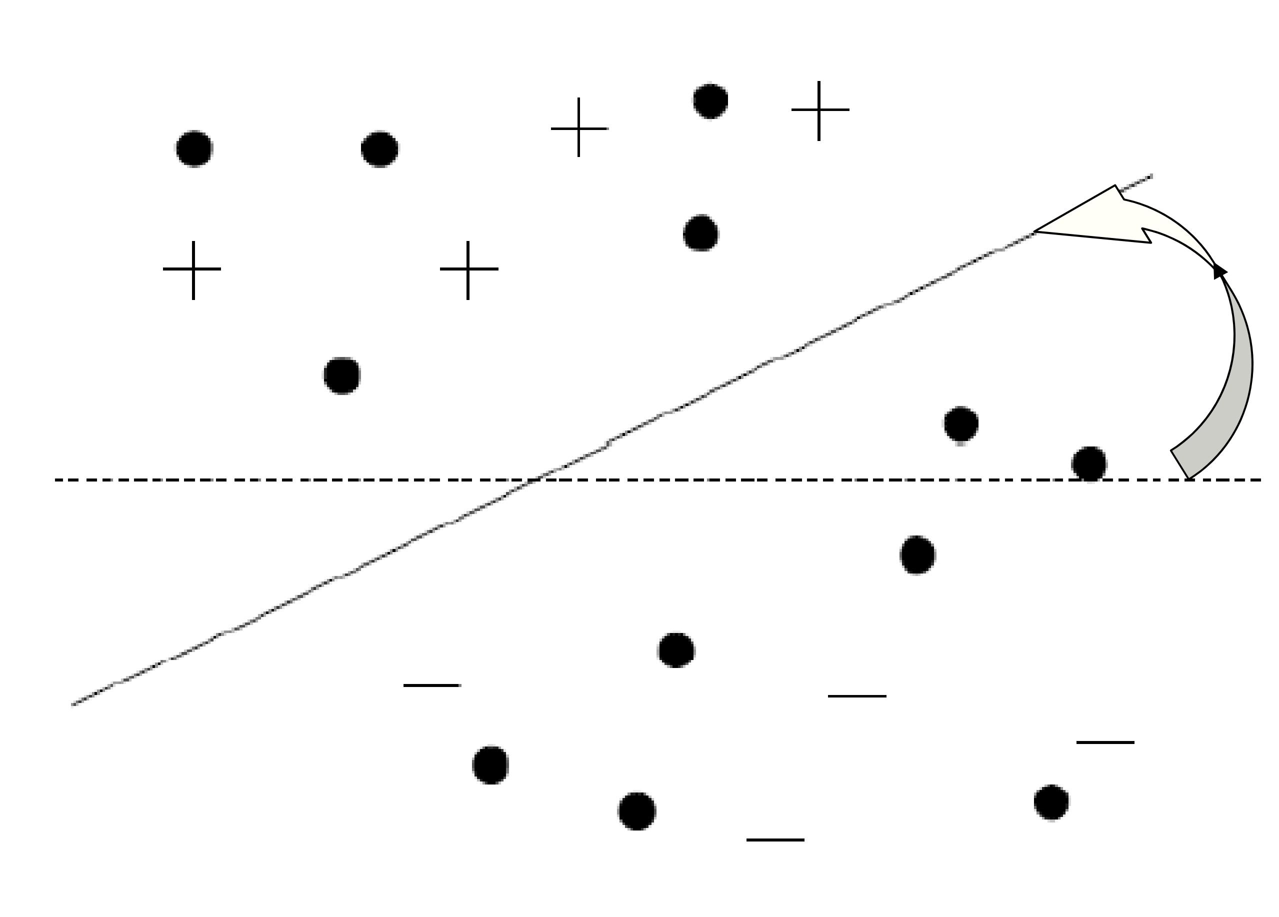
Prediction within V1.0

- **Not as rare a problem as some tend to believe.**
  - Customized products are developed regularly.
  - One of a kind applications:
    - Embedded systems, space systems, defense applications.
    - Typically high dependability domains.
  - NASA MDP data sets fall into this category.
- **Labeling modules for fault content is COSTLY!**
  - The fewer labels needed to build a model, the cheaper the prediction task.
    - The absence of problem report does not imply fault free module.
- **Standard fault prediction literature assumes *massive* amounts of labeled data available for training.**

Goals

- **How much data does one need to build a fault prediction model?**
  - What happens when most modules do not have a label?
- **Explore suitable machine learning techniques and compare results with previously published approaches.**
  - Semi –supervised learning (SSL).
  - An intermediate approach between supervised and unsupervised learning.
  - Labeled and unlabeled data used to train the model
  - No specific assumptions on label distributions.

SSL: Basic idea



Basic idea

- **Iteratively train a supervised learning algorithm from "currently labeled" modules.**
  - Predict the labels of unlabeled modules.
  - Migrate instances with *"high confidence"* predictions into the pool of labeled modules (FTcF algorithm).
  - Repeat until all modules labeled.

- **Large number of independent variables (>40).**
  - Dimensional reduction (not feature selection).
  - Multidimensional scaling as the data preprocessing technique.

Algorithm

A variant of self-training approach and Yaworski's algorithm.

Pre-processing Step:MDS
1: Input: X, Y_l, d_m
2: $d = tune.MDS(X_l, Y_l, d_m)$
3: $Z = MDS(X, d)$
4: Output: Z
SSL Learning Step: FTcF
1: Input: Z, Y_l
2: Initialization: $D_l = (Z_l, Y_l), \mathsf{u} = u$
3: loop until $\lvert u \rvert \to 0$:
4: Fit $\hat{Y}_u = \phi_{D_l}(Z_u)$
5: Take u' confident cases from Z_u
6: Updating: $Z_l = Z_{l+u'}$, $Z_u = Z_{u-u'}$, $Y_l = Y_l + \hat{Y}_{u'}$, and $D_l = (Z_l, Y_l)$
7: End loop
8:Output: $\hat{Y}_{\mathsf{u}}$

An unlabeled module may change the label in each iteration…

Base learner φ : Random forest

- robust to noise

Fault Prediction Data Sets

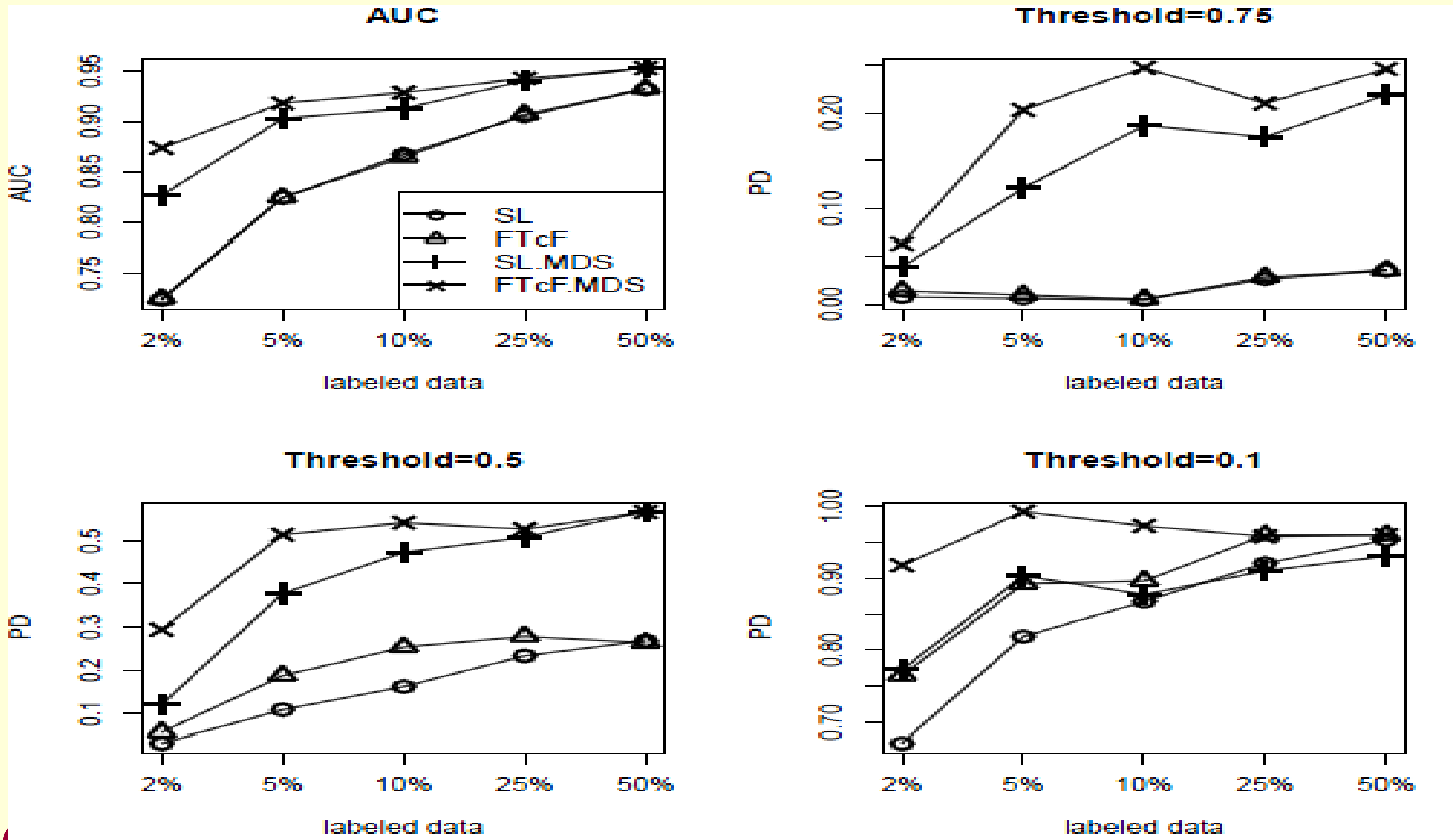
Data	Size#	% faulty	project description	language
KC1	2109	13.9%	Storage management for ground data	C++.
PC3	1563	10.43%	Flight software for earth orbiting satellite	C
PC4	1458	12.24%	Flight software for earth orbiting satellite	C
PC1	1109	6.59%	Flight software from an earth orbiting satellite	C

- Large NASA MDP projects (> 1,000 modules)

Experimentation

- **Compare the performance of four fault prediction approaches, all using RF as the base learner:**
  - Supervised learning (SL)
  - Supervised learning with dimensionality reduction (SL.MDS)
  - Semi-supervised learning (SSL)
  - Semi-supervised learning w dimensionality reduction (SSL.MDS)
- **Assume *2% - 50% of modules are labeled*.**
  - Randomly selected, 10 times.
- **Performance evaluation: Area under ROC, PD**
  - **PD =** $\frac{|\widehat{Y}_U \geq \tau|}{|Y_U = 1|}$ $\tau = \{0.1, 0.5, 0.75\}$

Results on PC4



Comparing Techniques: AUC

Table 2: AUC for the four data sets

Data	size of L	SL	FTcF	SL.MDS	FTcF.MDS
PC1	2%	0.6733	0.6677	0.8379	**0.8536**
	5%	0.7122	0.7087	0.8719	**0.8889**
	10%	0.7721	0.7806	0.9166	**0.9253**
	25%	0.8484	0.8464	0.9353	**0.9356**
	50%	0.8687	0.8728	0.9425	**0.9434**
PC3	2%	0.7053	0.7096	0.7550	**0.7841**
	5%	0.7386	0.7355	0.8494	**0.8860**
	10%	0.7512	0.7573	0.8829	**0.9024**
	25%	0.7922	0.7981	0.9103	**0.9183**
	50%	0.8199	0.8246	**0.9267**	0.9260
PC4	2%	0.7235	0.7246	0.8264	**0.8737**
	5%	0.8242	0.8243	0.9029	**0.9183**
	10%	0.8672	0.8644	0.9129	**0.9285**
	25%	0.9054	0.9069	0.9403	**0.9430**
	50%	0.9321	0.9327	**0.9538**	0.9535
KC1	2%	0.7374	0.7295	0.6793	**0.7382**
	5%	0.7404	0.7476	0.7437	**0.7477**
	10%	0.7635	0.7693	0.7728	**0.7831**
	25%	0.7794	0.7897	**0.7938**	0.7850
	50%	0.8043	0.8108	**0.8134**	0.8030

Comparing Techniques: PD

Table 5: PD with threshold=0.1 for the four data sets

Data	size of L	SL	FTcF	SL.MDS	FTcF.MDS
PC1	2%	0.6365	0.7108	0.8027	**0.8770**
	5%	0.6662	0.7394	0.8648	**0.9592**
	10%	0.7138	0.8092	0.8800	**0.9631**
	25%	0.8204	0.8571	0.8796	**0.9449**
	50%	0.8476	0.8667	0.9095	**0.9238**
PC3	2 %	0.6395	0.7758	0.7248	**0.8771**
	5%	0.6693	0.7497	0.8196	**0.9725**
	10 %	0.6910	0.7903	0.8366	**0.9538**
	25%	0.7851	0.8587	0.8579	**0.9240**
	50%	0.8085	0.8622	0.8780	**0.9098**
PC4	2 %	0.6710	0.7642	0.7727	**0.9182**
	5%	0.8187	0.8924	0.9035	**0.9930**
	10%	0.8677	0.8963	0.8774	**0.9732**
	25%	0.9211	0.9606	0.9106	**0.9585**
	50%	0.9538	0.9604	0.9311	**0.9604**
KC1	2%	0.5489	0.7249	0.5938	**0.7969**
	5%	0.6067	0.6876	0.7130	**0.8190**
	10%	0.6773	0.7461	0.7421	**0.8043**
	25%	0.6773	0.7473	0.7260	**0.7623**
	50%	0.7368	0.7695	0.7505	**0.7618**

Statistical Analysis

H_0: There is no difference between the 4 algorithms across all data sets

H_a: Prediction performance of at least one algorithm is significantly better than the others across all data sets

P-value from ANOVA measures evidence against **H_0**

Table 2: P-value of ANOVA test on varied size of labeled data for all performance measures

size of L	AUC	PD(0.75)	PD(0.5)	PD(0.1)
2%	**0.03795**	**0.00054**	**0.00052**	**0.00100**
5%	0.05688	**0.000105**	**1.09E-06**	**0.01011**
10%	0.08185	**5.72E-07**	**3.97E-06**	**0.02952**
25%	0.33810	**1.44E-05**	**0.00033**	0.53151
50%	0.49175	**0.00062**	**0.00433**	0.85391

Which approaches differ significantly?
Use post-hoc Tukey's "honestly significant difference (HSD)"

Table 4: Significance comparison of PD(0.1)

	SL	FTcF	SL.MDS
FTcF	none	–	–
SL.MDS	none	none	–
FTcF.MDS	2%, 5%, 10%	2%	2%

Benchmarking

- **Lessman (TSE 2008) and Menzies (TSE 2007) offer benchmark performance for NASA MDP data sets**
  - Lessman et al. on 66% of the data, Menzies trains on 90%,

Table 11: Comparison of results with [18] using AUC

Data sets	Size of L	SL	FTcF.MDS	Lessmann [18]
PC1	2%	0.67	0.85	
	5%	0.71	0.89	
	10%	0.77	**0.93**	
	25%	0.85	**0.94**	
	50%	0.87	**0.94**	0.9
PC3	2%	0.71	0.78	
	5%	0.74	**0.88**	
	10%	0.75	**0.90**	
	25%	0.79	**0.91**	
	50%	0.82	**0.93**	0.82
PC4	2%	0.72	0.87	
	5%	0.82	0.92	
	10%	0.87	0.93	
	25%	0.91	0.94	
	50%	0.93	0.95	0.97
KC1	2%	0.74	0.74	
	5%	0.74	0.74	
	10%	0.76	**0.78**	
	25%	**0.78**	**0.79**	
	50%	**0.80**	**0.80**	0.78

Table 10: Comparison of results with [19]

Data sets	Size of L	SL	FTcF.MDS	Menzies [19]
PC1 (PF=0.17)	2%	0.45	0.73	
	5%	0.46	**0.80**	
	10%	**0.53**	**0.85**	
	25%	**0.66**	**0.88**	
	50%	**0.74**	**0.91**	0.48
PC3 (PF=0.35)	2%	0.66	0.77	
	5%	0.73	**0.90**	
	10%	0.74	**0.92**	
	25%	**0.81**	**0.94**	
	50%	**0.85**	**0.95**	0.8
PC4 (F=0.29)	2%	0.62	0.89	
	5%	0.79	0.81	
	10%	0.86	0.97	
	25%	0.94	0.98	
	50%	**0.98**	**0.99**	0.98

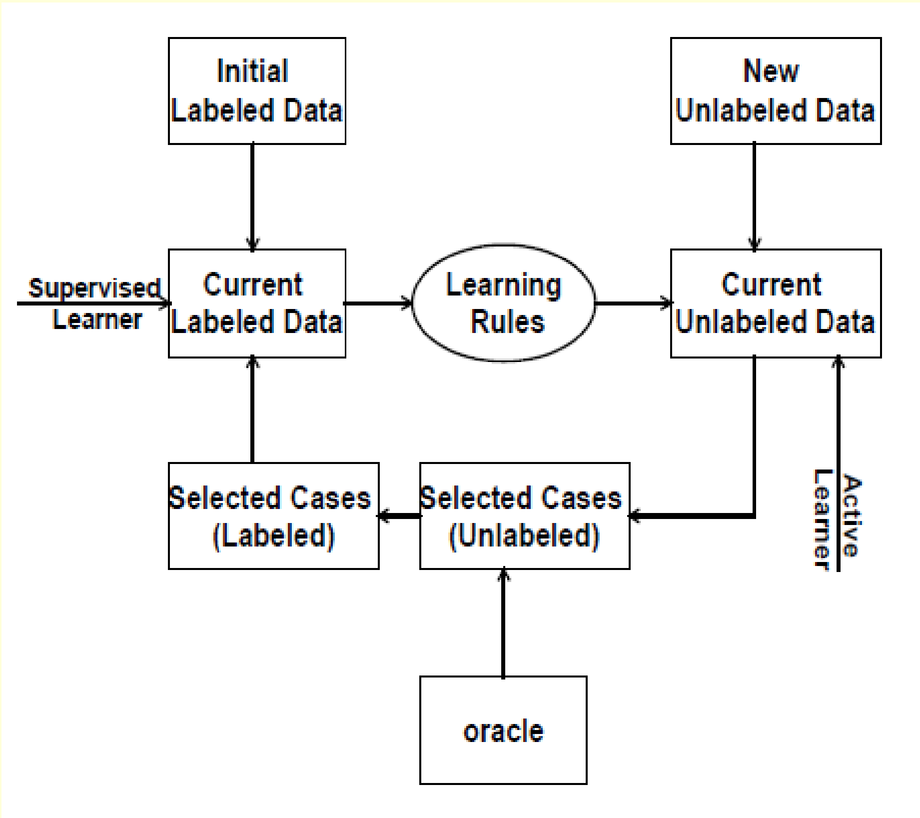
What if predicting on V2.0?

- **The lack of training data not an issue.**
- **Eclipse data set**

Release	packages/files	% with defects	metrics
2.0	377 / 6729	50.4% / 14.5%	41 / 32
2.1	434 / 7888	44.7% / 10.8%	41 / 32
3.0	661 / 10593	47.4% /14.8%	41 / 32

- **Active instead of supervised learning**
  – Characteristics of faults change between the successive versions.

Methodology

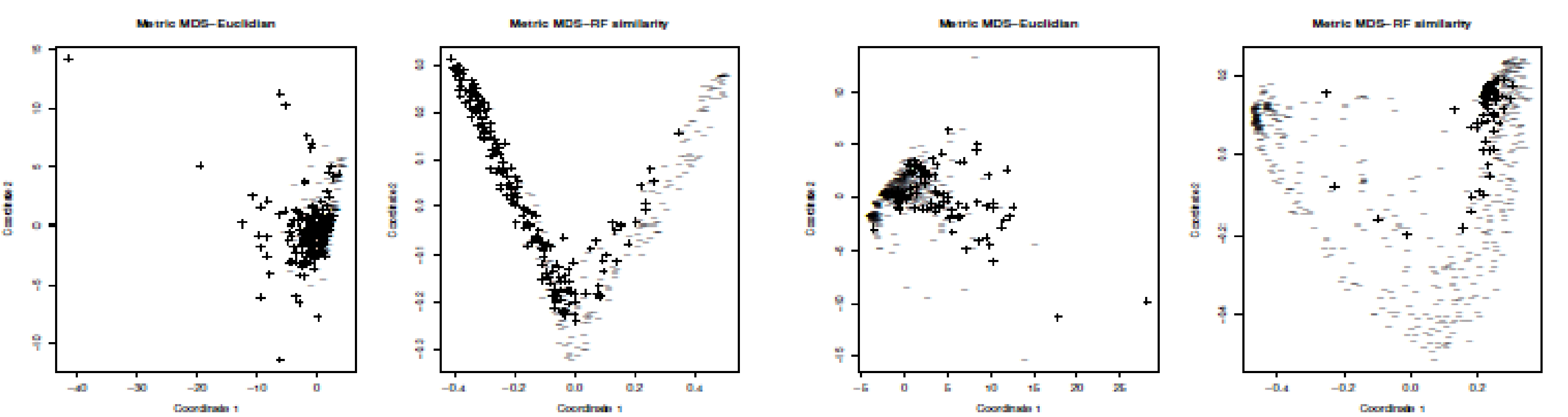


In each iteration, 1% of the modules is "labeled" by the "oracle".

"Oracle" → Software V&V Engineer

Dimensionality Reduction

- **Too many highly correlated software metrics!**
- **Multi-dimensional scaling (MDS)**
  - A nonlinear optimization.
  - Finds embeddings s.t. similarities are preserved.
  - Similarity measure matters – random forest similarity



a) Packages level b) Files level

Experiments

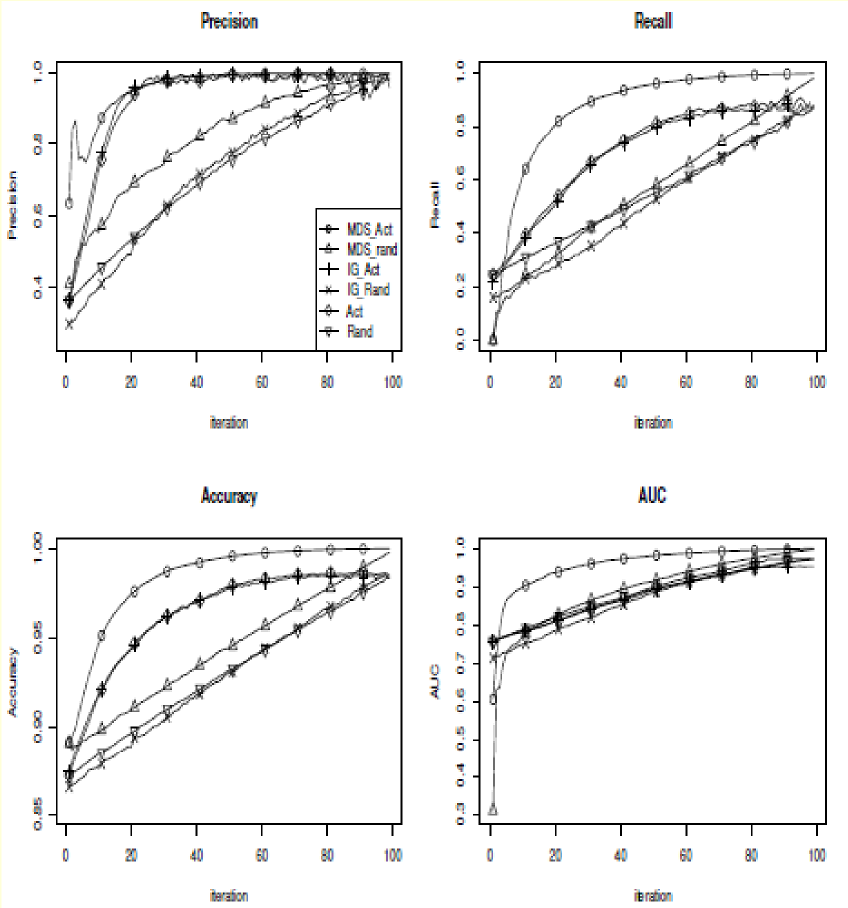


Figure 7: Defect prediction in release 2.1 from 2.0 (files)

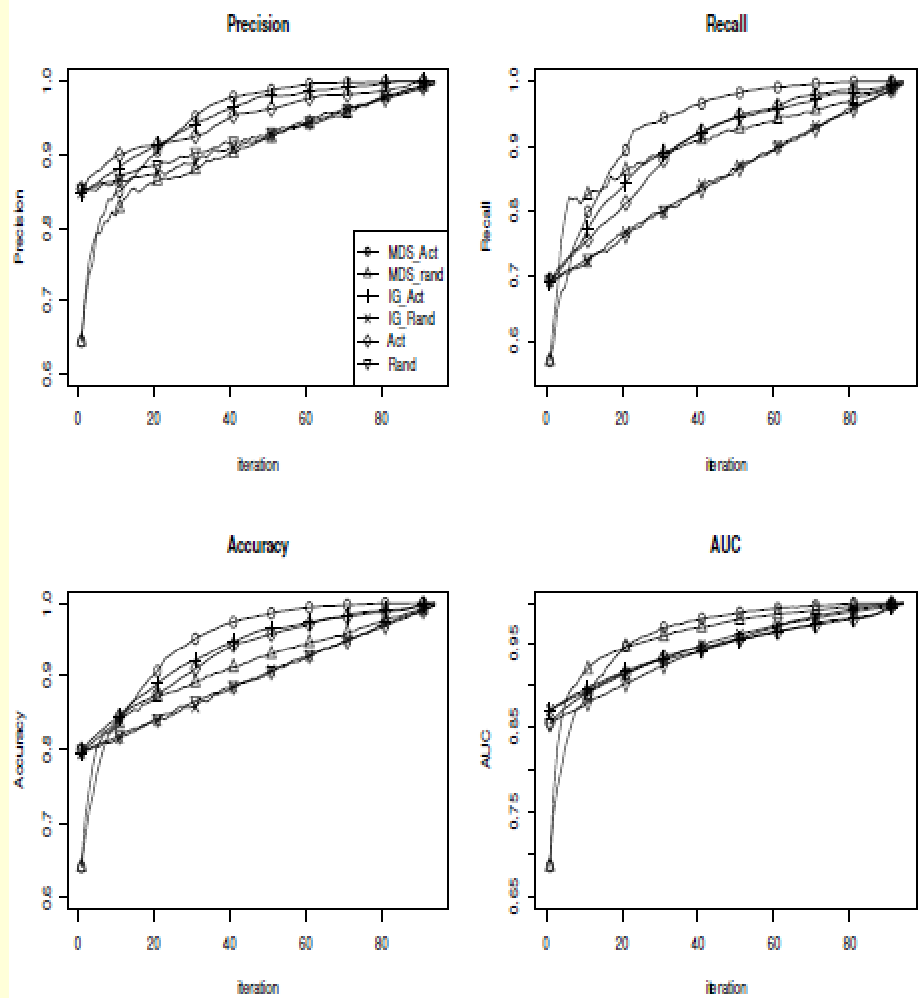


Figure 6: Defect prediction in release 3.0 from 2.0 and 2.1 (packages)

Statistical Significance

Table 10: Post-hoc test for performance differences between the six active learning approaches at their 30^{th} iteration ($1: MDS\_Act, 2: MDS\_rand, 3: IG\_Act, 4: IG\_rand, 5: Act, 6: Rand$). "✓" stands for statistically significant difference between two approaches. "x" stands for no significant difference detected between the two approaches.

Prediction from	Predicting for	Methods Compared	Package level				File level			
			Precision	Recall	Accuracy	AUC	Precision	Recall	Accuracy	AUC
release 2.0	release 2.1	1–2	✓	✓	✓	✓	✓	✓	✓	✓
		1–3	✓	✓	✓	✓	x	✓	✓	✓
		1–4	✓	✓	✓	✓	✓	✓	✓	✓
		1–5	✓	✓	✓	✓	x	✓	✓	✓
		1–6	✓	✓	✓	✓	✓	✓	✓	✓
		3–4	✓	✓	✓	x	✓	✓	✓	✓
		3–5	x	x	x	✓	x	x	x	x
		3–6	✓	x	✓	x	✓	✓	✓	x
		5–6	✓	✓	✓	x	✓	✓	✓	x
release 2.0 & 2.1	release 3.0	1–2	✓	✓	✓	✓	✓	✓	✓	✓
		1–3	x	✓	✓	✓	x	✓	✓	✓
		1–4	✓	✓	✓	✓	✓	✓	✓	✓
		1–5	✓	✓	✓	✓	x	✓	✓	✓
		1–6	✓	✓	✓	✓	✓	✓	✓	✓
		3–4	✓	✓	✓	x	✓	✓	✓	x
		3–5	x	x	✓	x	x	x	x	x
		3–6	✓	✓	✓	✓	✓	✓	✓	x
		5–6	✓	✓	✓	✓	✓	✓	✓	x
release 2.1	release 3.0	1–2	✓	✓	✓	x	✓	✓	✓	✓
		1–3	x	✓	✓	✓	x	✓	✓	✓
		1–4	✓	✓	✓	✓	✓	✓	✓	✓
		1–5	✓	✓	✓	✓	x	✓	✓	✓
		1–6	✓	✓	✓	✓	✓	✓	✓	✓
		3–4	✓	✓	✓	x	✓	✓	✓	x
		3–5	✓	x	x	✓	x	x	x	✓
		3–6	✓	✓	✓	x	✓	✓	✓	✓
		5–6	✓	✓	✓	✓	✓	✓	✓	✓

Summary

- **Fault prediction from few data points is feasible**
  - A few extra points in large projects help the prediction too.
- **Unlabeled data naturally occurs in fault prediction.**
  - Embrace it!
- **While not predicting reliability, these techniques optimize V&V expenditure.**

Outline – Software Engineering as Data Science

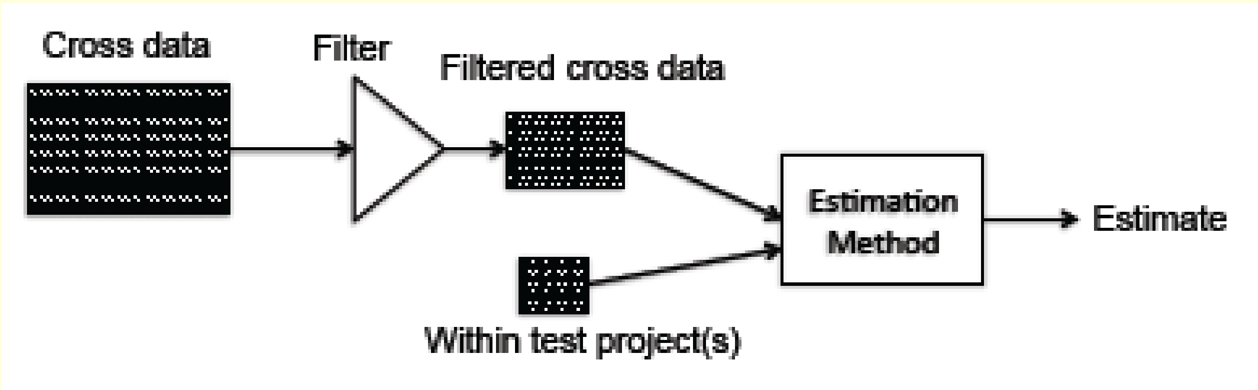
- **Fault prediction**
  - Early in the life cycle.
  - Lower the cost of V&V by directing the effort to places that most likely hide faults.
- **Effort prediction**
  - With few data points from past projects.
- **Problem report triage**
  - Minimize human involvement.
- **Summary**

Software Effort Estimation (SEE)

- **Supervised learning predominant in the literature**
  - Independent variables
    - E.g. ***metrics*** defining completed software projects.
  - Dependent variables
    - E.g. ***labels*** *(effort values)* from past projects.
- **Collecting metrics is relatively easy, but**
  - *The collection of labels is very costly* [1].
  - In some cases actual effort data may not even exist.
- **Data starved problems!**

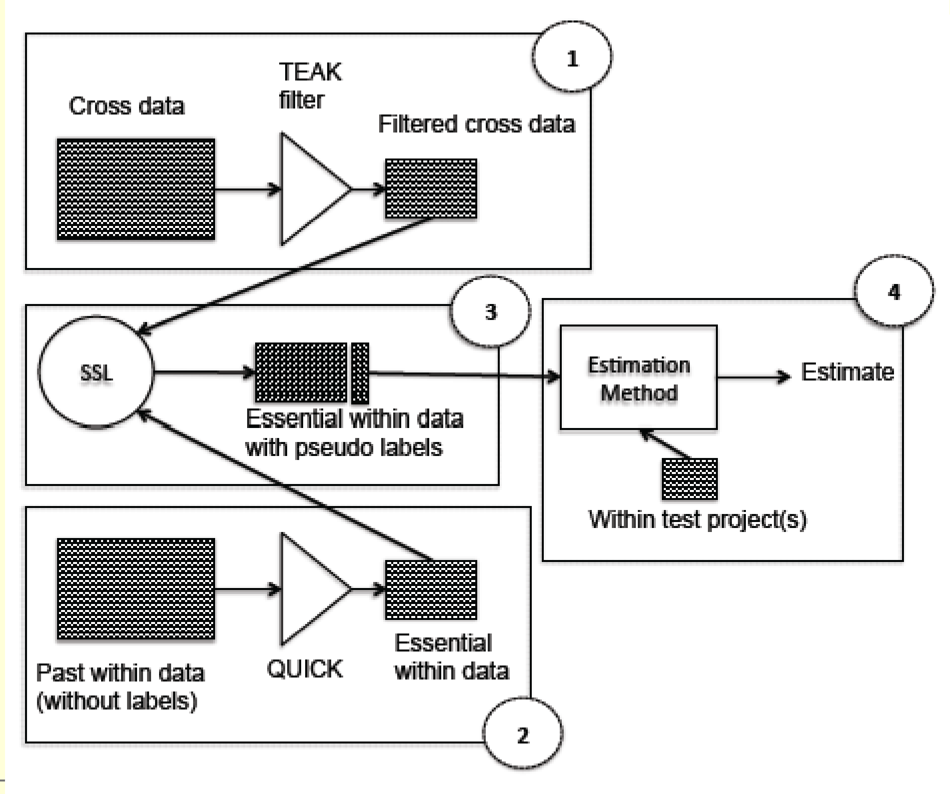
Proposition of Cross-company Data

- **When effort data from past is not available**
  - Use effort examples from others (cross-company data)
  - Use cross-company data for training
- **Is it relevant for your project?**
  - Transferring all project examples is not a good idea.
  - Select instances that appear to be projects “similar” to the one at hand.



Synergistic effort prediction

- **The goal is to enable effective prediction in cases when doing it with other methods would not be feasible.**



Performance

***Synergy,* compared to within/cross-company learning over 20 runs (hence 2x20 = 40 total comparisons) in terms of win, tie, loss**

- Cases of **losses are highlighted** with gray

Dataset	MAR			MMRE			MdMRE			Pred(25)			MBRE			MIBRE			MMER		
	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
cocomo81e	0	39	1	2	32	6	0	32	8	1	32	7	0	25	15	0	25	15	0	26	14
cocomo81o	0	27	13	0	31	9	0	31	9	0	31	9	0	26	14	0	26	14	0	27	13
cocomo81s	0	40	0	0	39	1	0	39	1	0	39	1	0	40	0	0	40	0	0	40	0
nasa93_center_1	0	38	2	0	39	1	0	39	1	0	39	1	0	38	2	0	38	2	1	39	0
nasa93_center_2	1	38	1	1	38	1	1	38	1	1	38	1	1	39	0	1	39	0	1	38	1
nasa93_center_5	0	40	0	1	38	1	0	38	2	0	38	2	6	29	5	6	29	5	7	28	5
desharnaisL1	0	38	2	0	32	8	0	32	8	0	32	8	0	28	12	0	28	12	0	28	12
desharnaisL2	0	38	2	0	37	3	0	37	3	0	37	3	0	38	2	0	38	2	0	38	2
desharnaisL3	0	31	9	0	24	16	0	24	16	0	24	16	0	31	9	0	31	9	0	40	0
finnishAppType1	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0
finnishAppType2345	0	38	2	0	38	2	0	38	2	0	38	2	0	39	1	0	39	1	1	38	1
kemererHardware1	0	40	0	0	39	1	0	39	1	0	39	1	0	40	0	0	40	0	0	40	0
kemererHardware23456	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0
maxwellAppType1	0	31	9	0	33	7	0	33	7	0	33	7	0	32	8	0	32	8	0	33	7
maxwellAppType2	0	39	1	0	39	1	0	39	1	0	39	1	0	37	3	0	37	3	0	35	5
maxwellAppType3	0	36	4	1	36	3	0	36	4	1	36	3	0	36	4	0	36	4	0	38	2
maxwellHardware2	0	35	5	0	38	2	0	38	2	0	38	2	0	34	6	0	34	6	0	34	6
maxwellHardware3	0	40	0	0	36	4	0	36	4	0	36	4	0	35	5	0	35	5	0	36	4
maxwellHardware5	0	36	4	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	39	1
maxwellSource1	0	39	1	0	37	3	0	37	3	0	37	3	0	38	2	0	38	2	0	39	1
maxwellSource2	0	40	0	0	33	7	0	33	7	0	33	7	0	27	13	0	27	13	0	26	14

Summary

- **Fully automated approach**
  - Experts not involved until the estimate is generated.
- **Cross company estimates created from publicly available data**
  - No collection cost.
- **Effort estimates can be interpreted through their similarity to local projects.**
  - Cross company learning imposes the risk that estimates cannot be easily understood when they are applied to the project.

Outline – Software Engineering as Data Science

- **Fault prediction**
  - Early in the life cycle.
  - Lower the cost of V&V by directing the effort to places that most likely hide faults.
- **Effort prediction**
  - With few data points from past projects.
- **Problem report triage**
  - Minimize human involvement.
- **Summary**

Motivation

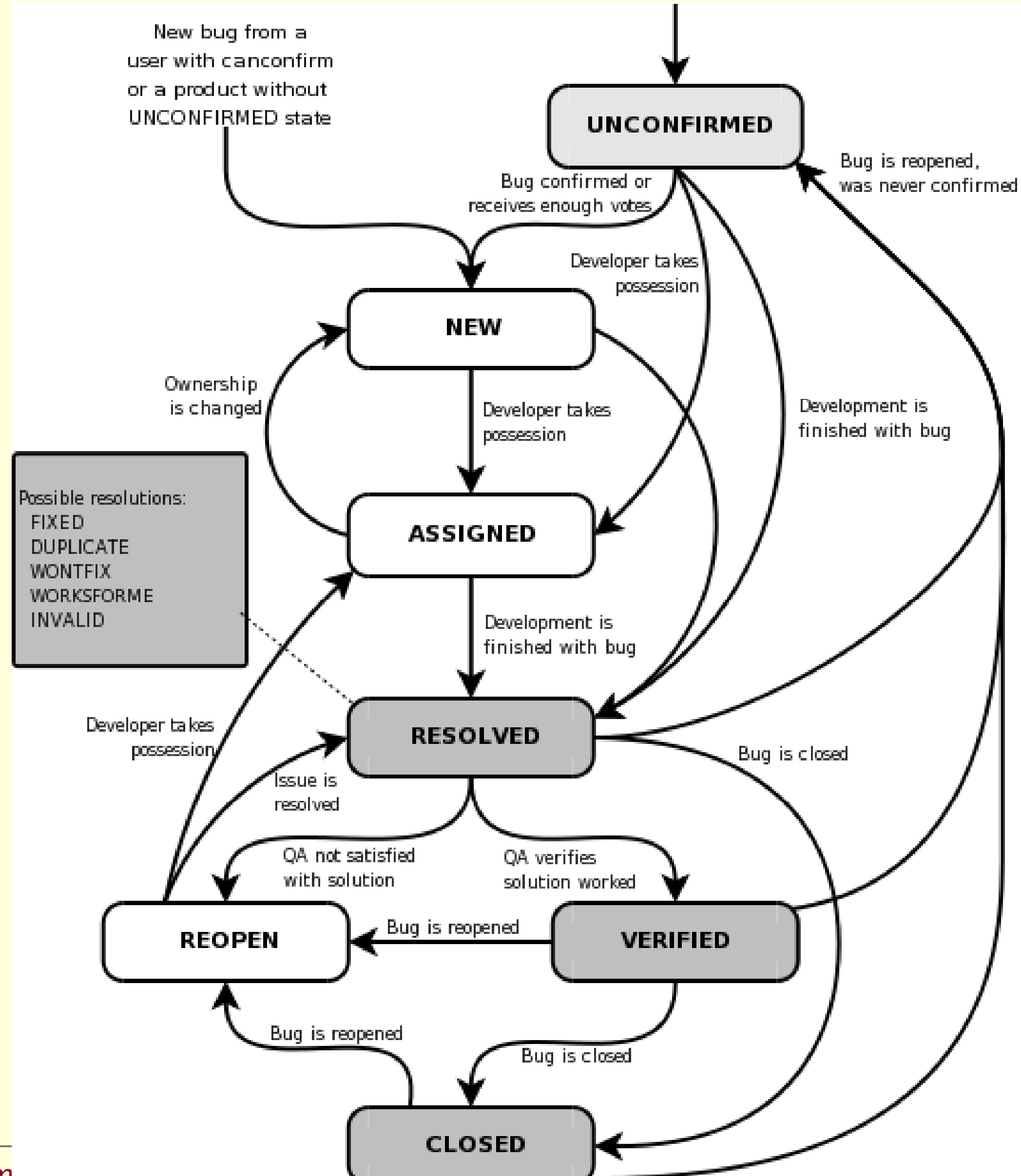
- **Automated analysis of text-based software documents is difficult.**
  - Volume
    - Open source projects average 300 - 400 newly submitted reports per day.
    - Firefox alone has over 120,000 problem reports associated with it, to date.
    - Mozilla has over 700,000 problem reports since 1998
  - Variability, diversity
    - An average problem report in Firefox contains 60-140 words
    - There are over 40,000 users submitting problem reports to the Firefox project

Issue reporting: definitions

- **Reports can be either:**
  - Primary – describing novel and unknown problems
  - Duplicates – describe previously reported problems
- **Triager:**
  - A person responsible for determining whether a report is "Primary" or "Duplicate" and assigning it to the appropriate developer
  - In open source, triagers are Mozilla staffers or volunteers
    - The development team can veto the decision of a volunteer triager.

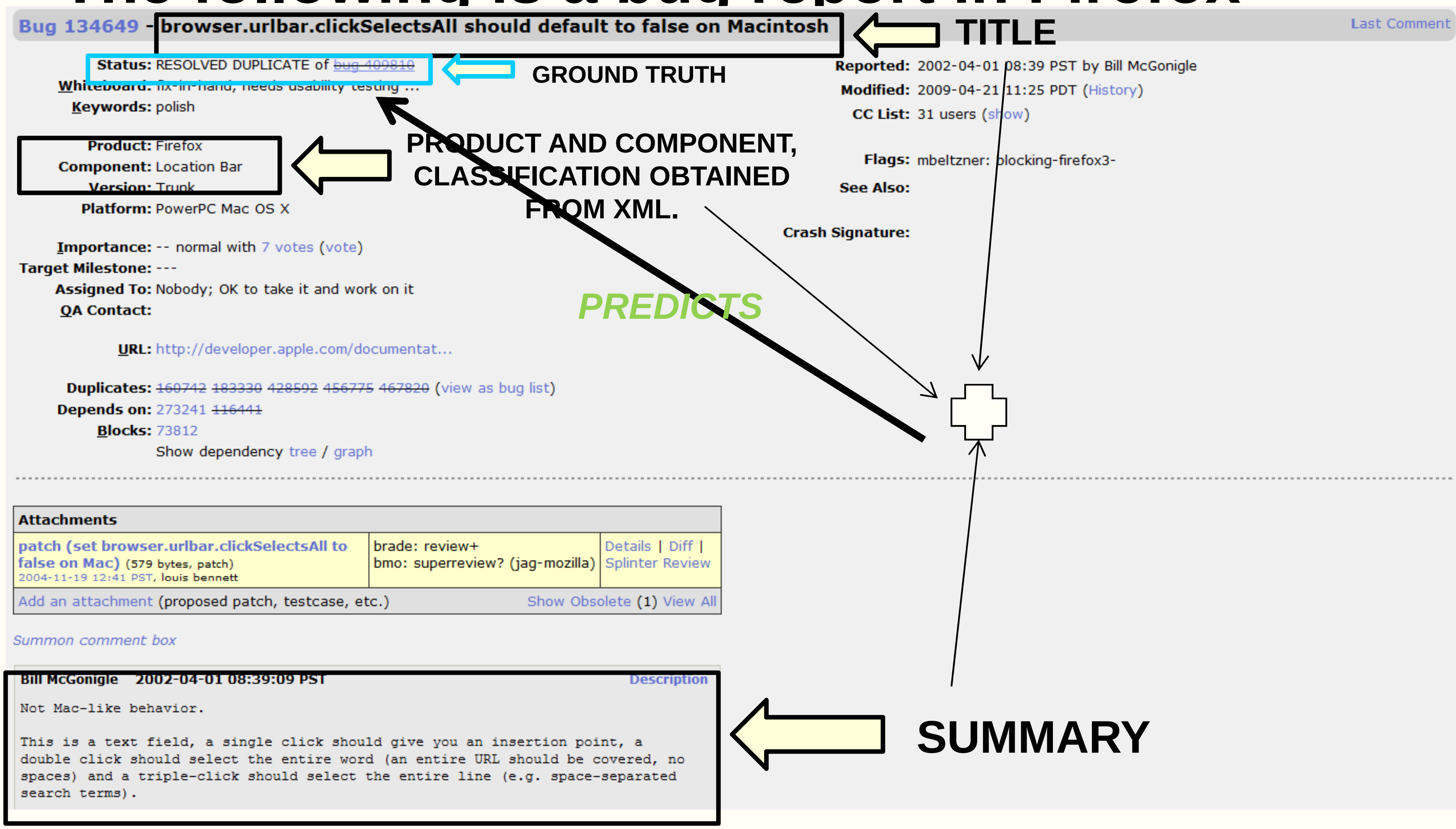
Life cycle of a bug report in Mozilla

- **CLOSED reports can be reopened and reassigned when new information appears**
- **The dynamic nature of the repository can make automated analysis work challenging**



Sample Bug Report

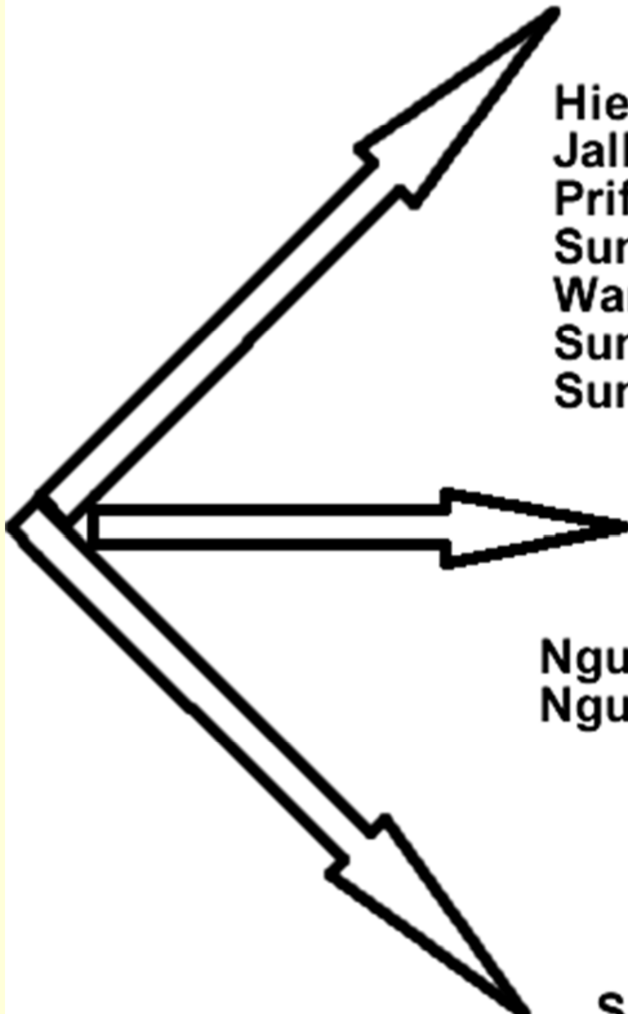
- The following is a bug report in Firefox



Characteristics of Firefox

Firefox	Total Number of Problem Reports	111,206
	Total Number of Duplicates	31,034
	Total Number of Detectable Duplicates	25,085
	Duplicates Within Nearest 15,000 Groups	24,255
	Percentage of Dataset Comprised of Duplicates	28%
	Number of Duplicate Groups	12,268
	Number of Duplicate Groups with 1 Duplicate	7,492
	Number of Primaries with No Duplicates	67,904
	Ratio of Duplicate Groups to Duplicates	2.53

Related Research



Word Frequency Methods

Hiew Firefox (<2006) - 50% recall
Jalbert Firefox (Feb 05 - Oct 05) - 51% recall
Prifti (<Jun 2010) - 53% recall
Sun Firefox (Apr 02 - Jul 07) - 53% recall
Wang Firefox (Jan 04 - Apr 04) - 67-93% recall
Sun Mozilla (Jan 10 - Dec 10) - 68% recall
Sun Eclipse (Jan 08 - Dec 08)- 75% recall

Dictionary Based Methods

Nguyen Mozilla (Jan 10 - Dec 10) - 80% recall
Nguyen Eclipse (Jan 08 - Dec 08) - 85% recall

Machine Learning Methods

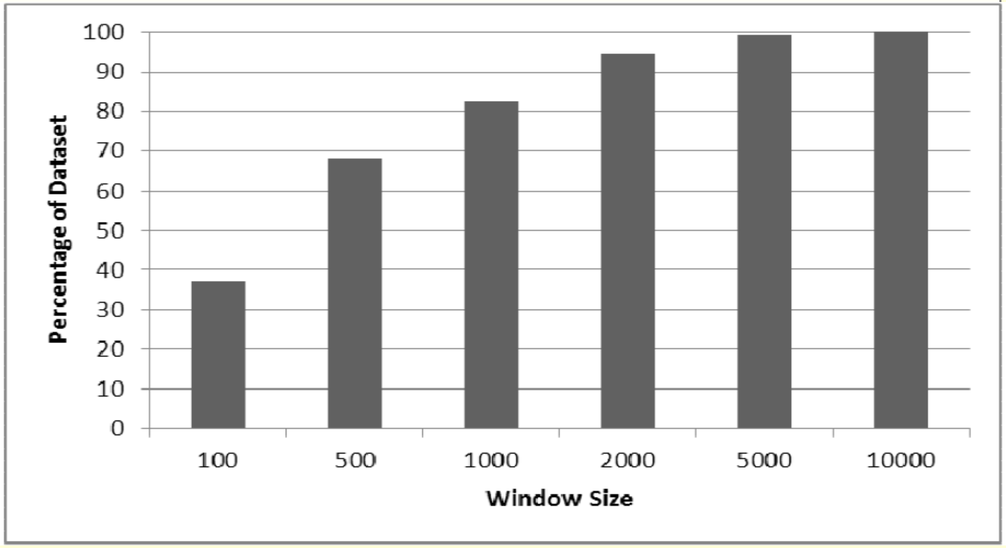
Sun Firefox (Apr 02 - Jul 07) - 70% recall

Research goals

- **Develop an effective automated (or semi automated) technique to detect similar reports.**
  - Can we develop a better word weighting scheme that places emphasis on intra group similarity?
  - Apply string matching to detect similar problem reports
- **Must be scalable, apply to small as well as to very large issue report data sets.**

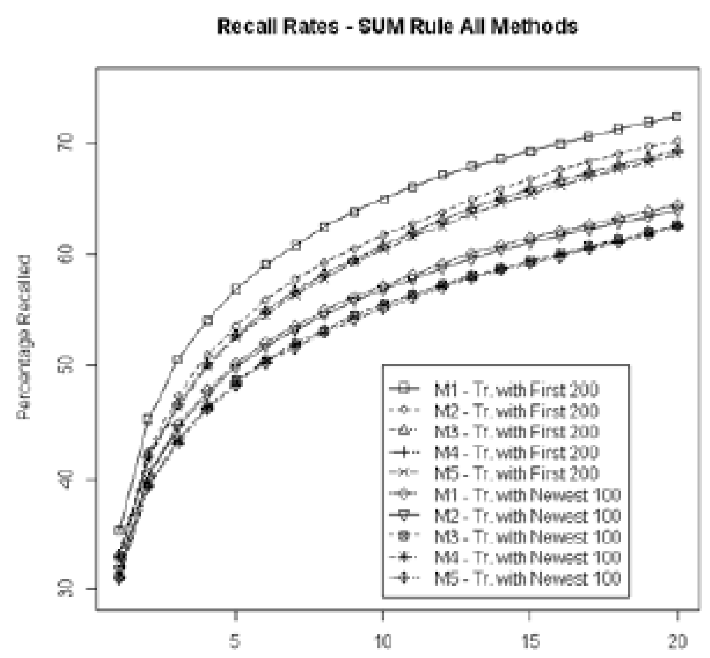
Approach

- **Use report's Title and Summary for analysis**
- **Pre-processing issue reports**
  - Tokenize, stem, remove non essential stop words
- **Combine 24 similarity measures into a multi-label classifier**
  - Cosine similarity with group centroids.
  - Longest common subsequence.
- **Time window**



Multi-label classification

- **MULAN**
  - Similarity measure match scores, reports since the last duplicate (or prime), title/summary size…
- **Classification indicates trust in the label correctness for each of the 24 measures**
- **Generate unified top 20 match list**



Summary

- **Research problem open to advancement**
  - Continual development of alternative approaches
  - Evaluation on the largest and most complicated open source repositories…
- **Upcoming work**
  - "social network" analysis of the bug reports
  - Automated detection of primary reports

Outline – Software Engineering as Data Science

- **Fault prediction**
  - Early in the life cycle.
  - Lower the cost of V&V by directing the effort to places that most likely hide faults.
- **Effort prediction**
  - With few datProblem report triage
  - a points from past projects.
  - Minimize human involvement.
- **Summary**

Summary

- **Software quality remains a research area with many challenges.**
  - Expensive consequences of faults.
  - Imperfect software requirements, derivation, construction…
- **Data analytics guide practitioners in decision making**
  - Emerging as the key analysis technique.
  - Intuitively guide verification activities.

Summary

- **Empirical evaluation remains the key to improvement**
  - Expanded list of artifacts: code, documentation, execution traces…
  - Realism in experiments.
- **Potential for significant savings in software engineering processes**
  - A major shift in software quality research.

Thank You

Questions?