Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

How to Tell Apart the Good from the Bad: Setting Thresholds in Software Engineering

Luigi Lavazza[(1)]

Dipartimento di Scienze Teoriche e Applicate

luigi.lavazza@uninsubria.it

(1) This material was developed jointly with Prof. Sandro Morasca

Contents

Contents

Topics
The context
The problem
Proposal 1: slope-based thresholds
Proposal 2: optimistic-pessimistic approach
Proposal 3: fault-proneness H-index
Final considerations

The context

- Like in all engineering disciplines, Software Engineering practitioners need to manage the quality of software products and processes
  - monitor
  - control
  - evaluate
  - improve
  - . . .
- In this presentation, we focus on the faultiness of software modules as the quality of interest

A bit of terminology

- Faultiness
  - Presence of at least one fault in a module.
- Software module: a “piece” of software
  - A subsystem, a class, a procedure, etc.

The context

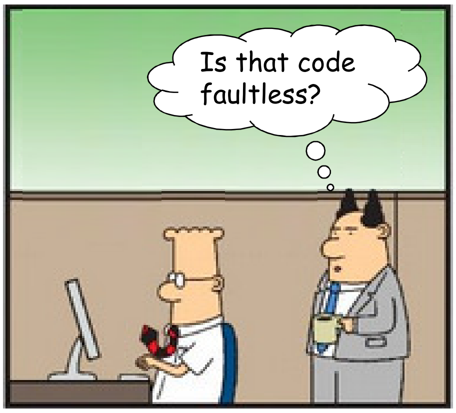
- Quantitative information helps managing quality
- Measures
  - Internal, depending only on the software itself
    - Code measures: size, complexity, coupling, etc.
  - External, depending also on elements of the external world
    - Faultiness (depending on specifications)
    - Maintainability (depending on the required changes)
    - …

Internal measures are not interesting by themselves

- The manager gets code measures, but he does not know how to interpret them.
- Note: even a *smart* manager who knows the meaning of RFC does not know what values of RFC are “good” and what values are “bad”.
  - This is typical of internal measures.

External measures are interesting

- The manager wants that only good quality code is released.
- Faultiness is what practitioners are really interested in for decision making along the software lifecycle
  - allocating V & V resources
  - controlling the production process
  - assessing the quality of the software under construction

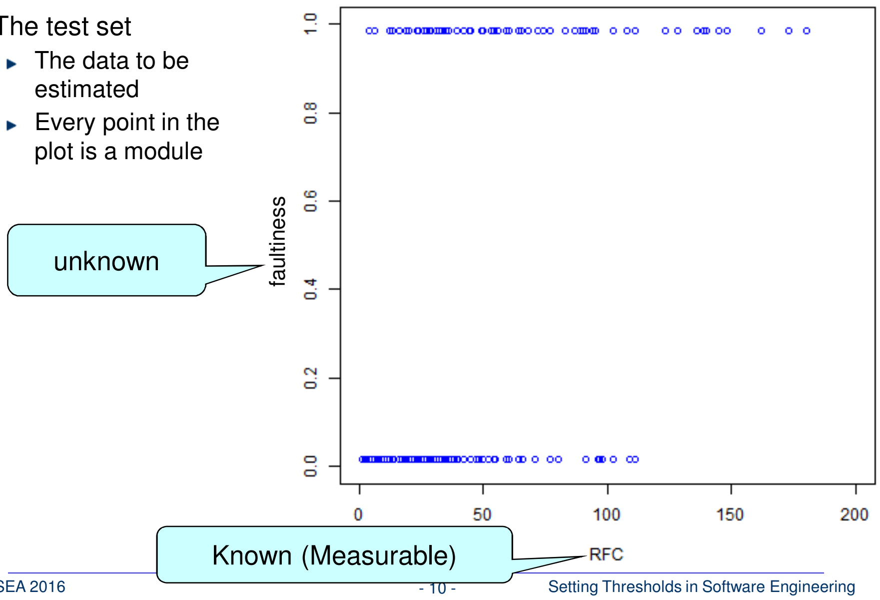


The context

- Unfortunately, faultiness cannot be measured based on the code only.
  - E.g., given a module, how can you “measure” if it is faulty or not?
- We need to **estimate** faultiness
  - We can use our knowledge about the module, i.e., the values of its internal measures
- But how?

Hypothesis one: estimates are based uniquely on internal measures

- The test set
  - The data to be estimated
  - Every point in the plot is a module

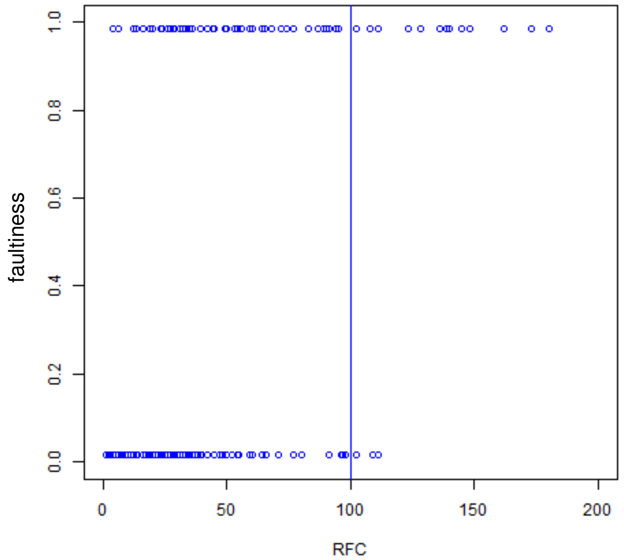


We need a threshold for estimating

- Under the current hypothesis, estimates are based uniquely on internal measures
  - E.g., RFC, response for a class
- We need a threshold T such that
  - Modules whose RFC measure is greater than T are classified faulty
  - Modules whose RFC measure is not greater than T are classified not faulty
- Problem: how do we define threshold T?
  - let's consider a few possibilities …

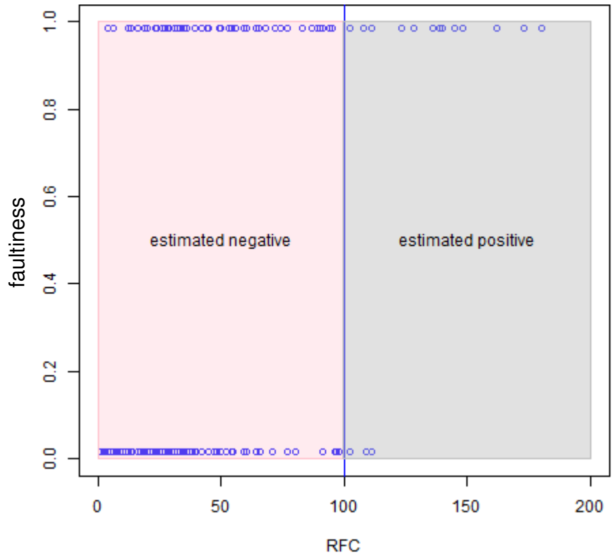
ICSEA 2016 - 11 - Setting Thresholds in Software Engineering

Midpoint Threshold



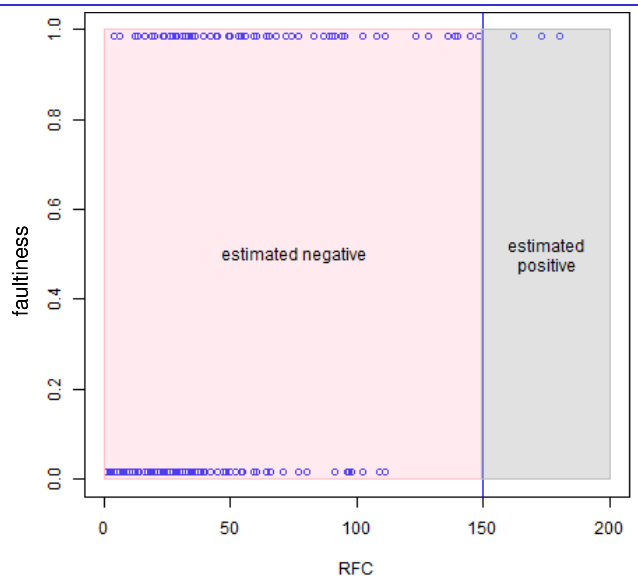
ICSEA 2016 - 12 - Setting Thresholds in Software Engineering

Midpoint Threshold



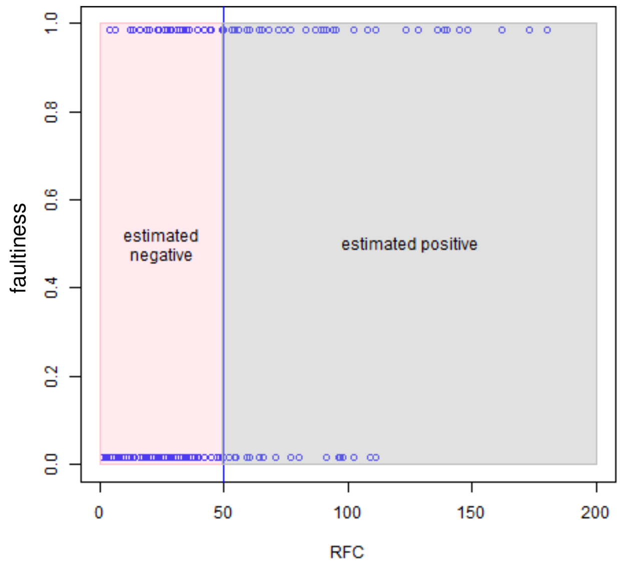
ICSEA 2016 - 13 - Setting Thresholds in Software Engineering

Upper Fourth Threshold: too Optimistic?

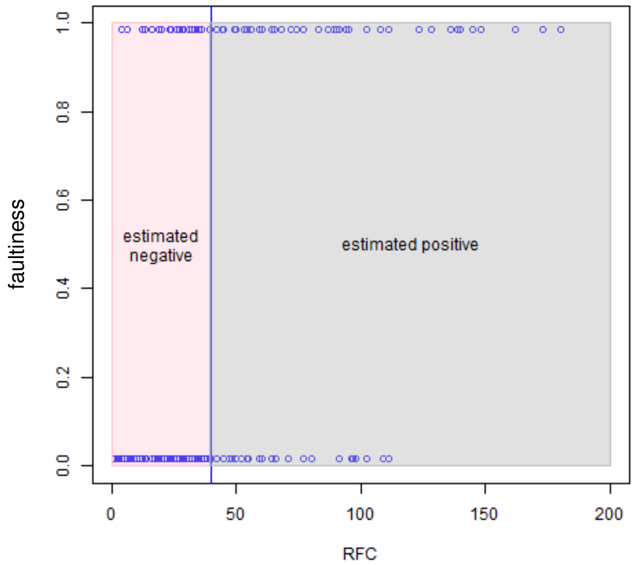


ICSEA 2016 - 14 - Setting Thresholds in Software Engineering

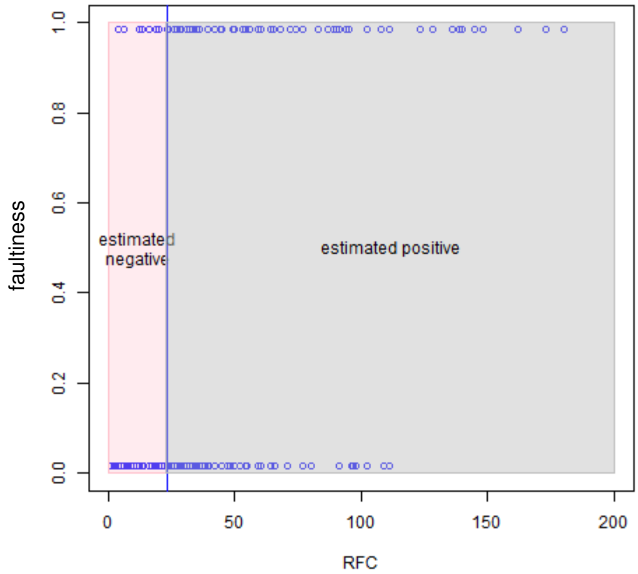
Lower Fourth Threshold: too Pessimistic?



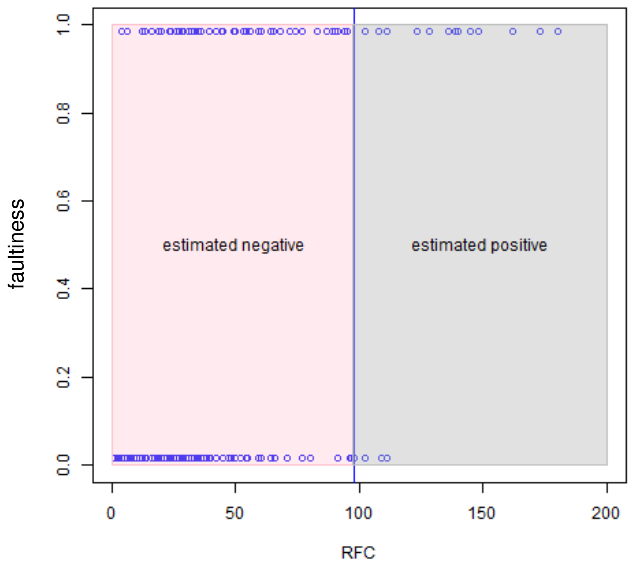
Mean Threshold



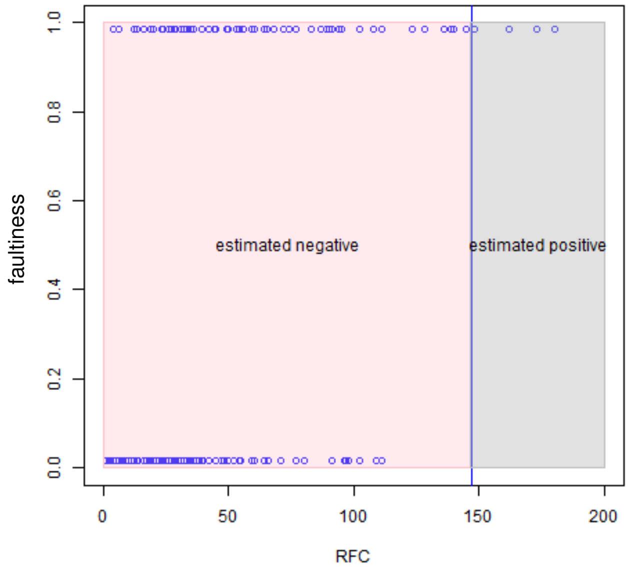
Median Threshold



Mean + Standard Deviation Threshold [Erni and Lewerentz]

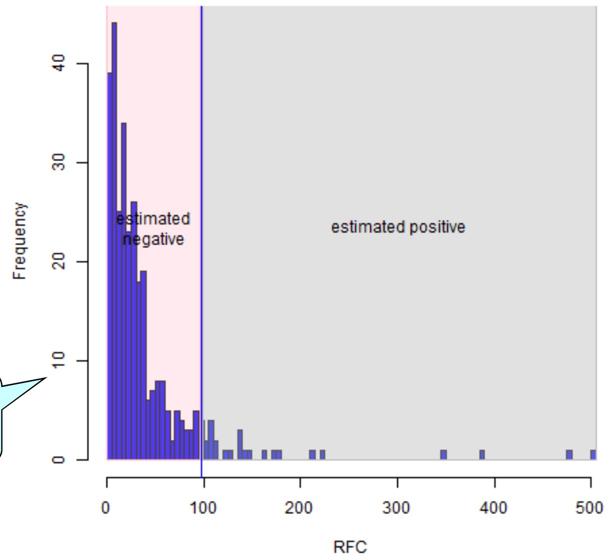


(Mean + Standard Deviation Threshold)*1.5 [Lanza and Marinescu]



ICSEA 2016 - 19 - Setting Thresholds in Software Engineering

Hypothesis one: estimates are based uniquely on internal measures



What's Faultiness got to do with it?

ICSEA 2016 - 20 - Setting Thresholds in Software Engineering

Hypothesis one: estimates are based uniquely on internal measures

- Do we get good results (i.e., accurate estimates) with this strategy?
- Not really.
  - We shall see some experimental results at the end of the presentation.

Bad results could be expected.
If you try to estimate fault-proneness based on a measure that is known to be related to fault-proneness, but without taking into consideration how it is correlated to fault-proneness, your guess could easily be wrong!

ICSEA 2016 - 21 - Setting Thresholds in Software Engineering

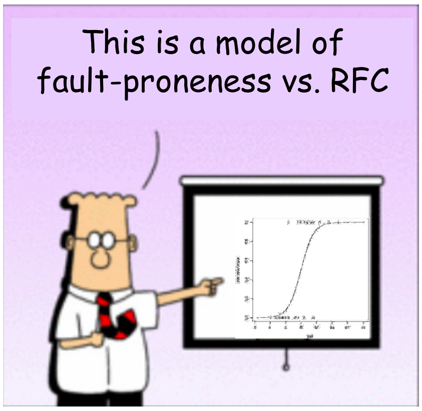
Hypothesis two: Use Internal Measures and Faultiness Data

- A faultiness estimation model can be built on top of
  - a fault-proneness estimation model
  - a fault-proneness threshold

A common practice in many fields.
E.g., widely used in mechanical maintenance, or in medicine.

ICSEA 2016 - 22 - Setting Thresholds in Software Engineering

Quality models available!



- Models relating internal measures (CBO, WMC, RFC, etc.) to external quality (e.g., fault-proneness) are (often) available.
- These models "transform" internal measures with no practical meaning into meaningful indications.

Models of fault-proneness

- Independent variable(s):
  - One or more internal measures
    - E.g., RFC, CBO, ...
- Dependent variable:
  - The quality of interest
    - In our case, fault-proneness
- Why fault-proneness instead of faultiness?
  - A model estimates the **probability of faultiness**, i.e., **fault-proneness**

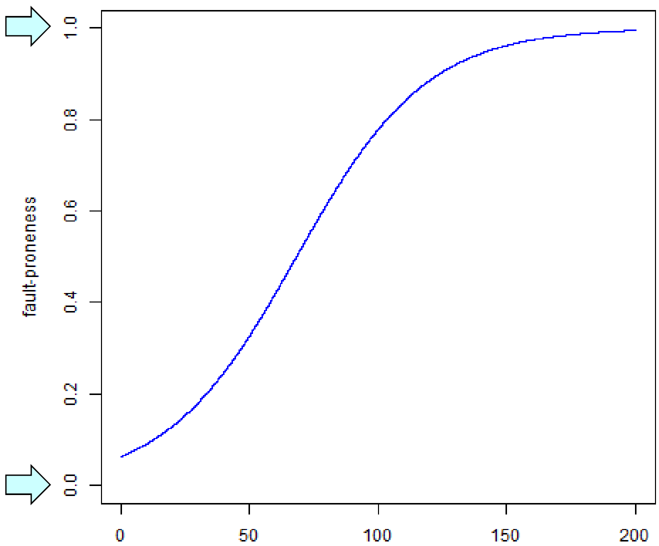
Binary Logistic Regression (BLR) models of fault-proneness

$$fp(x) = \frac{e^{logit(x)}}{1 + e^{logit(x)}}$$

- logit(X) is a linear function
  - univariate case: $c_0 + c_1X$
    - X is the internal measure
  - multivariate case: $c_0 + c_1X_1 + c_2X_2 + \ldots$
    - X_1, X_2 … are internal measures

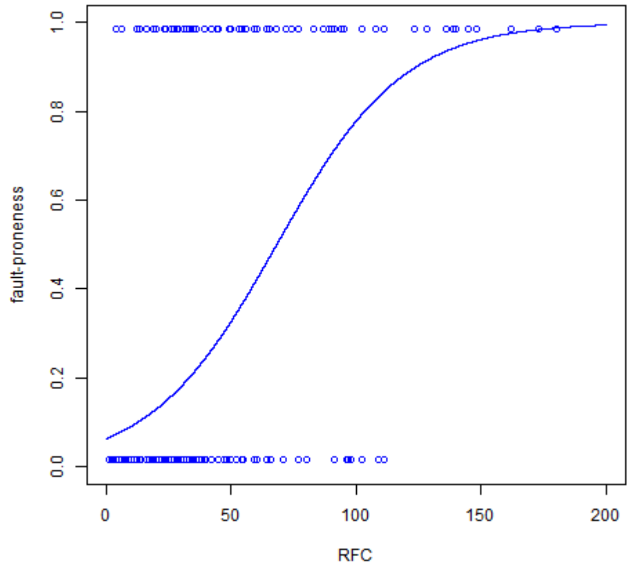
ICSEA 2016 - 25 - Setting Thresholds in Software Engineering

BLR model



ICSEA 2016 - 26 - Setting Thresholds in Software Engineering

BLR model



In this presentation we consider only models with increasing fp. Decreasing models are possible.

Other models

- Other types of models can be used, like, the Probit Binary Regression (PBR)

$$fp(z(\underline{X})) = \Phi(z(\underline{X})) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z(\underline{X})} e^{-t^2/2}\, dt$$

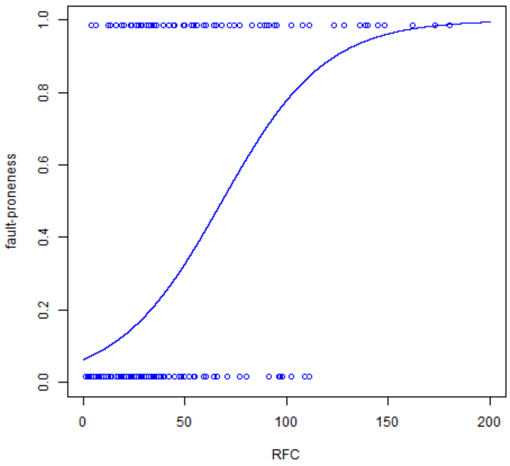
- The resulting model is S-shaped, much like the BLR model.
- In this presentation we shall use only BLR models.
  - What we shall see here can be usually extended easily to other types of models.

How to build models

- Assuming that we have proper data
  - E.g., a spreadsheet with
    - A row for each module
    - A column for each internal measure
    - A column for faultiness
- We need a statistical tool to compute the model.
- I suggest R
  - Open-source and free
  - Supported by a huge community
  - There are books and documentation available
  - Provides a wealth of statistical tools
    - To make sure that the models found are statistically significant
    - To test their “goodness”
      - Hosmer test, likelihood ratio test, ...

ICSEA 2016 - 29 - Setting Thresholds in Software Engineering

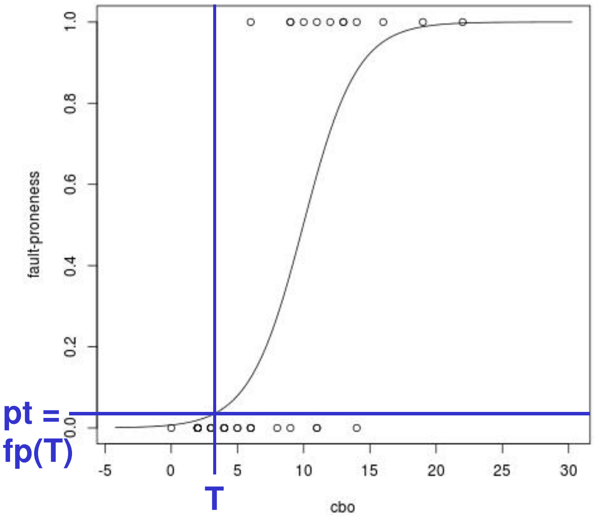
How to use a model?



- How do we use the knowledge that an internal measure is related to the probability of faultiness?

ICSEA 2016 - 30 - Setting Thresholds in Software Engineering

Threshold on a Fault-proneness Curve

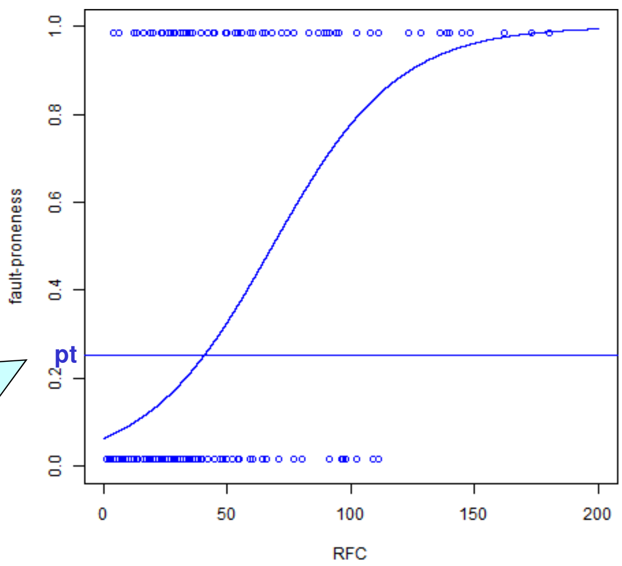


We need a threshold pt, which indicates the maximum acceptable value for fault-proneness

pt=fp(T)

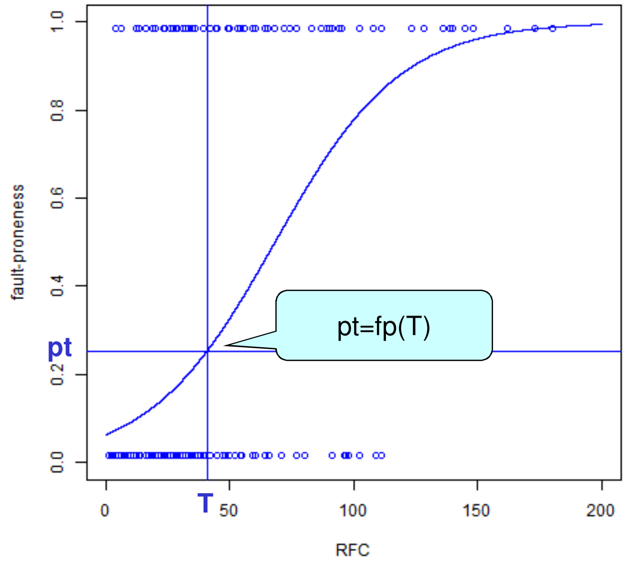
When CBO of a module M grows greater than T the manager should start some activity to improve M, because its probability of being faulty is beyond the maximum acceptable risk

Threshold on Fault-proneness



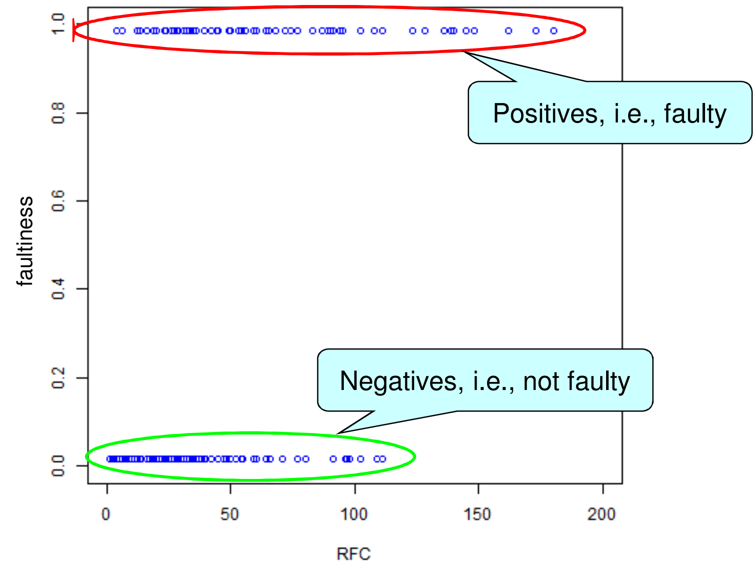
We need a threshold pt, which indicates the maximum acceptable value for fault-proneness

Threshold on Independent Variable

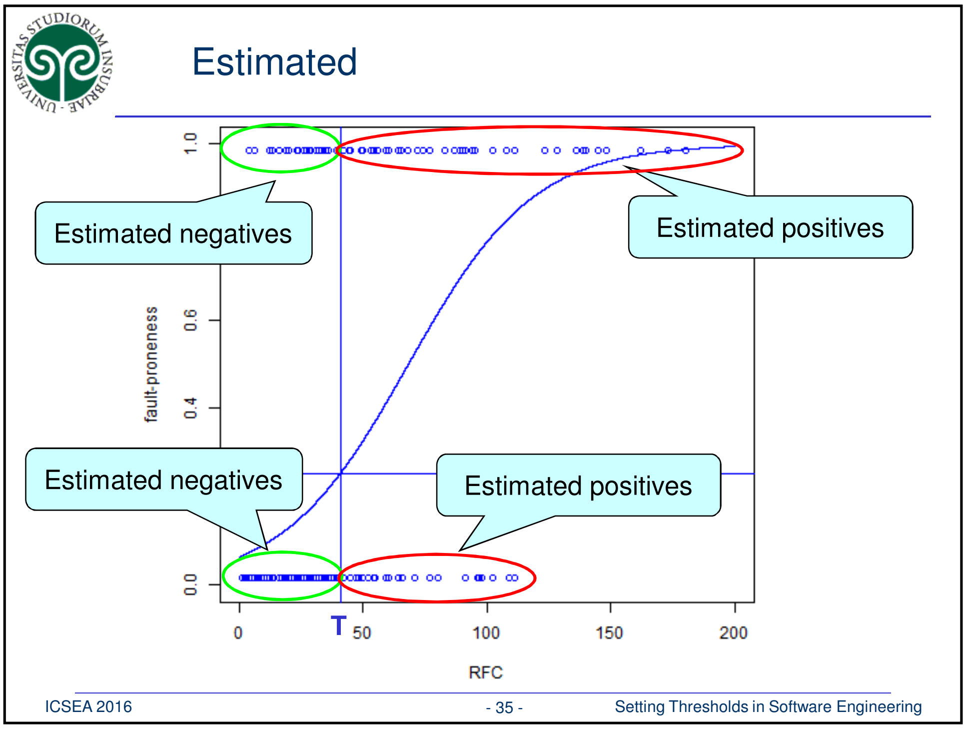


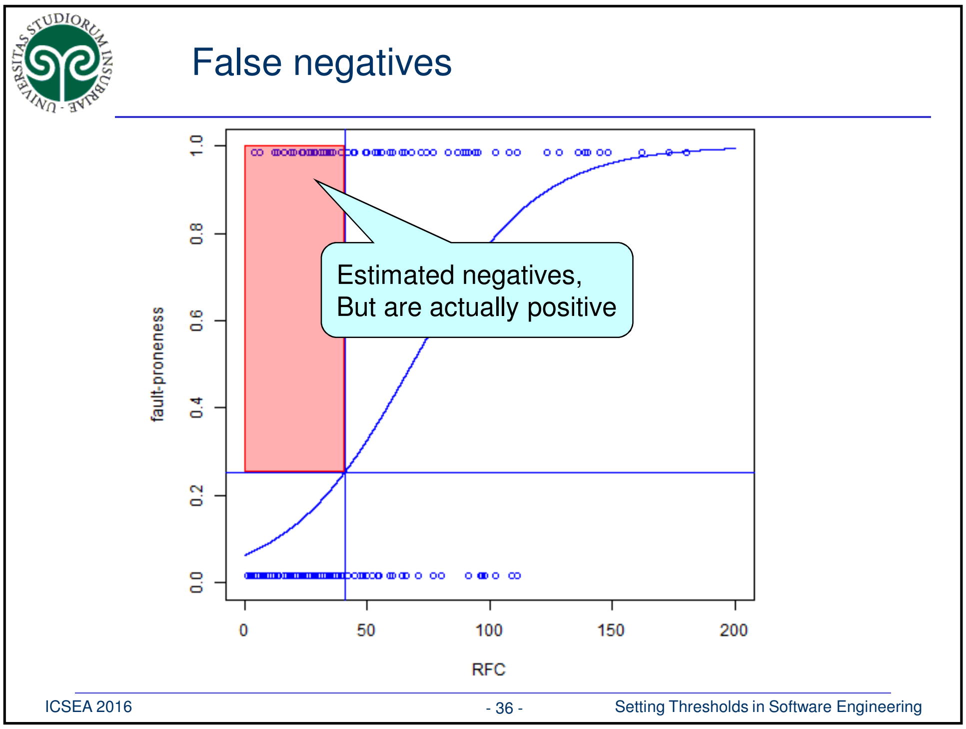
ICSEA 2016 - 33 - Setting Thresholds in Software Engineering

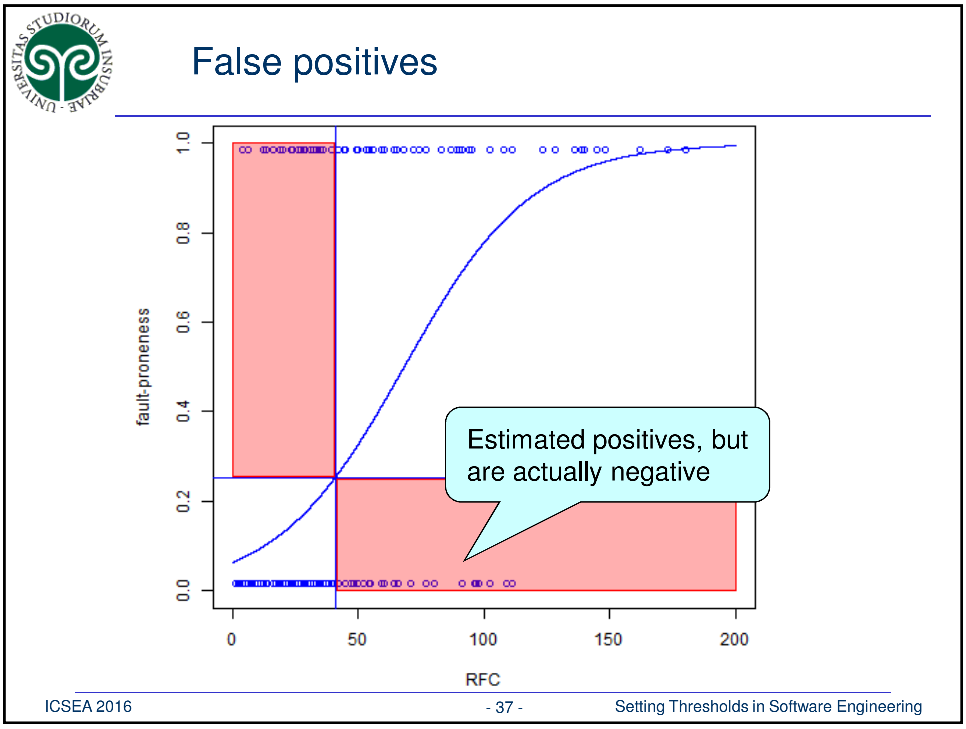
Positives and negatives

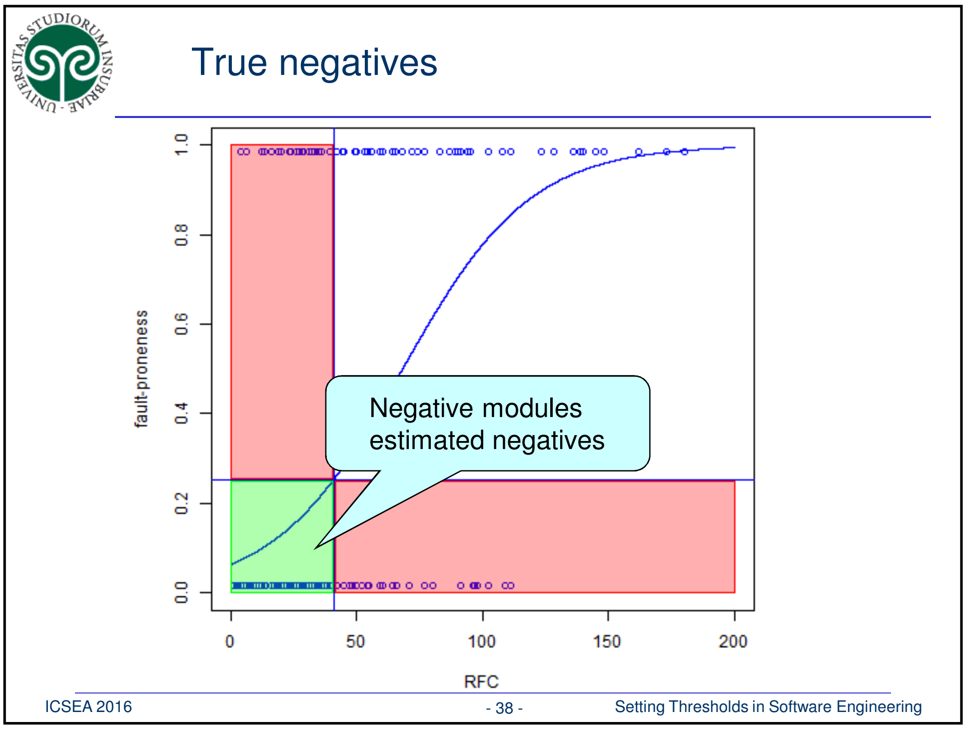


ICSEA 2016 - 34 - Setting Thresholds in Software Engineering

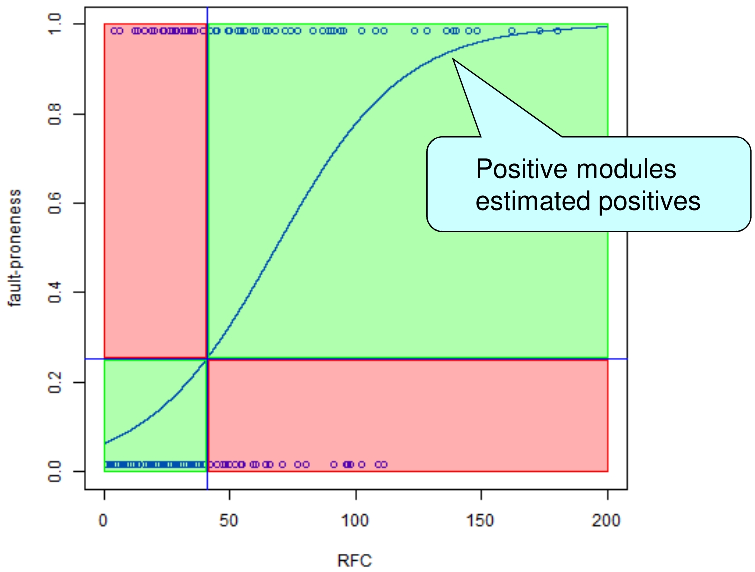

Estimated
Estimated negatives
Estimated positives
Estimated negatives
Estimated positives
fault-proneness
0.0
0.4
0.6
1.0
0
T
50
100
150
200
RFC
ICSEA 2016
- 35 -
Setting Thresholds in Software Engineering


False negatives
Estimated negatives,
But are actually positive
fault-proneness
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
RFC
ICSEA 2016
- 36 -
Setting Thresholds in Software Engineering


False positives
Estimated positives, but are actually negative
fault-proneness
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
RFC
ICSEA 2016
- 37 -
Setting Thresholds in Software Engineering


True negatives
Negative modules estimated negatives
fault-proneness
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
RFC
ICSEA 2016
- 38 -
Setting Thresholds in Software Engineering

True positives



How good is a model?

- We need to evaluate how good is a model, i.e., how accurate are its estimates.
- Informally, we want
  - Many true positives and true negatives
  - As few as possible false negatives and false positives.

Estimated/Actual Faultiness Contingency Tables

- We need to check how close estimated faultiness is to actual faultiness

		Actual		
		Non-faulty	Faulty	Total
Estimated	Non-faulty	TN	FN	EN
	Faulty	FP	TP	EP
	Total	AN	AP	n

Accuracy indicators

- Precision: proportion of estimated positives that are actually positive

$$Precision = \frac{TP}{EP}$$

- Recall: proportion of actual positives that are estimated positives

$$Recall = \frac{TP}{AP}$$

- F-measure: harmonic mean of Precision and Recall

$$Fmesure = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

Contents

The real problem

- How should we choose pt (hence, T)?

Some possible thresholds

- fp = 0.5 (Fifty)
  - a theoretical threshold, used for no prior knowledge, same value no matter the application or discipline
- fp = $\frac{AP}{n}$ (All)
  - This is the proportion of faulty modules in the entire data set
    - Useful to evaluate the accuracy of the proposed thresholds
      - It is the value you get with a constant logit

ICSEA 2016 - 45 - Setting Thresholds in Software Engineering

Why not using Fifty or All thresholds?

- Fifty does not use any knowledge about the actual modules.
  - If AP/n is 0.1 and you use 0.5 thresholds, you are going to have a lot of false positives
  - If AP/n is 0.9 and you use 0.5 thresholds, you are going to have a lot of false negatives
- AP/n could be a reasonable choice. Unfortunately, AP is not known at estimation time.

ICSEA 2016 - 46 - Setting Thresholds in Software Engineering

Contents

Topics
The context
The problem
Proposal 1: slope-based thresholds
Proposal 2: optimistic-pessimistic approach
Proposal 3: fault-proneness H-index
Final considerations

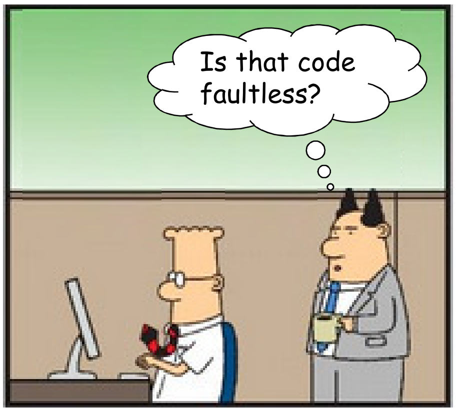
Slope-based Thresholds[1]

- A first proposal is applicable when we want to identify "**early symptoms**" of possible faultiness

[1] Sandro Morasca and Luigi Lavazza, "Slope-based Fault-proneness Thresholds for Software Engineering Measures ", *EASE 2016*

The problem

- The manager wants that only good quality code is released.
- He wants to get some evidence that lets him take action **as soon as** the quality of a module under development becomes **“not good enough”**.



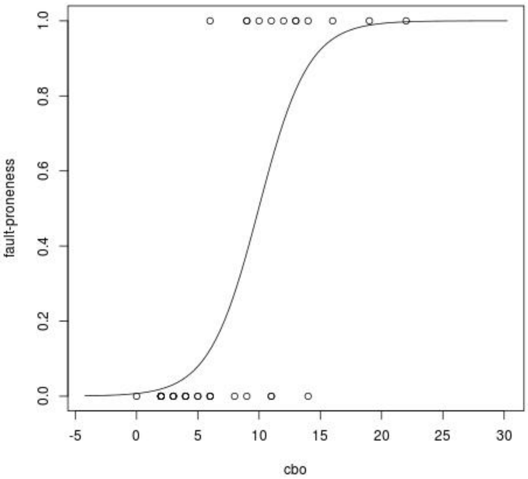
ICSEA 2016 - 49 - Setting Thresholds in Software Engineering

We have a model

- The model is built as shown before, based on data from previous developments (e.g., of previous releases).

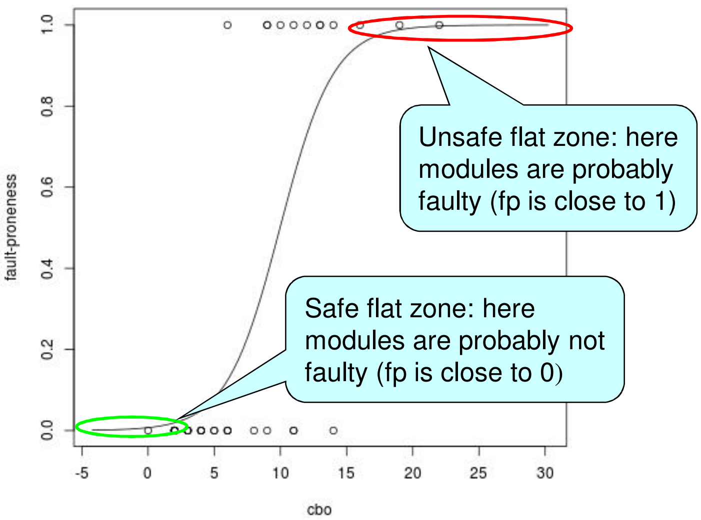
ICSEA 2016 - 50 - Setting Thresholds in Software Engineering

Yes, but … how to use the model?



ICSEA 2016 - 51 - Setting Thresholds in Software Engineering

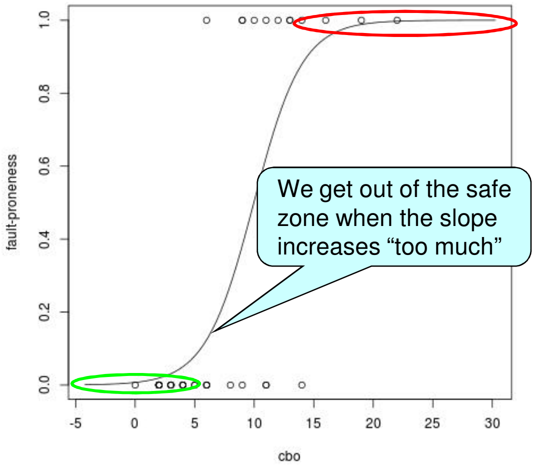
What does the model tell us?



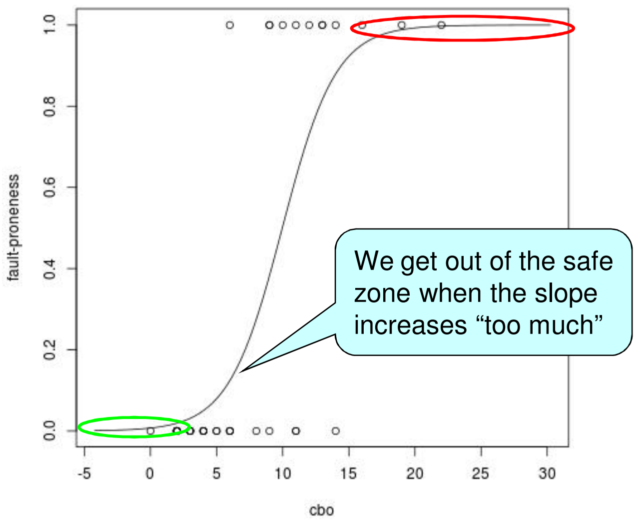
ICSEA 2016 - 52 - Setting Thresholds in Software Engineering

Slope-based thresholds

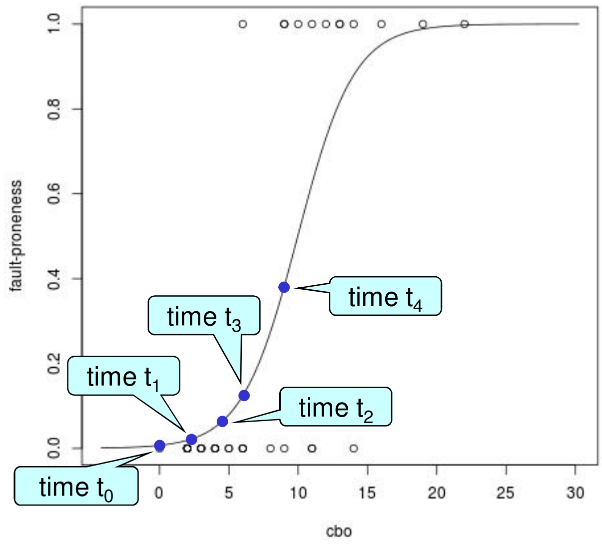
- When a module is created its CBO is zero.
- Then, while the module is being implemented, CBO increases over time
- We want to identify "early symptoms" of possible faultiness
- Idea: we need to constrain CBO to be less than a value CBO_{MAX} where small variations of CBO imply large variations of fp(CBO)



The basic idea



Where to set the threshold?



- When should the manager start warrying?
- At t_1 fault proneness is comfortably close to zero.
- At t_4, the slope is close to maximum, and fault proneness is already quite high.
- What about t_2 and t_3?

Our proposals

- Goal: identify a threshold based on "**early symptoms**" of faultiness
- Basic observations
  - fp(x) looks rather "flat" for small values of X
    - even fairly large variations in X imply small variations in fault-proneness
  - As x increases, fp(x) reaches a value past which
    - it departs very fast from the flat low-risk area
    - actually, it increases very fast
- Idea: set the threshold where the slope starts to become too steep
  Based on geometric properties of models
  - Maximum convexity
  - Fraction of maximum slope

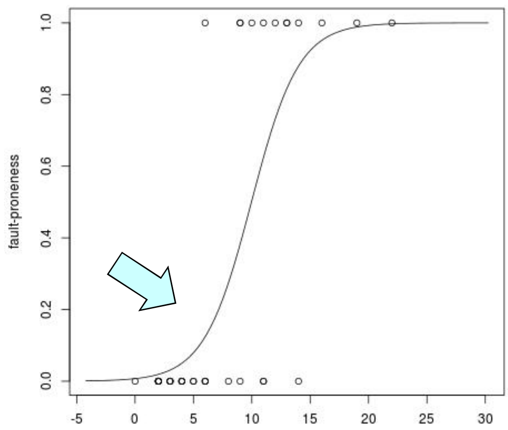
Proposal 1: Maximum Convexity (MC)

- At the beginning the slope/direction of fp(X) changes very slowly
- At the end the slope/direction of fp(X) changes very slowly too
- But, in between the slope/direction of fp(X) changes much faster
- We define the threshold as the value x_{MC} of X in which fp(X) changes slope/direction the fastest

ICSEA 2016 - 57 - Setting Thresholds in Software Engineering

Maximum convexity (MC)

- Slope is measured by fp'(X)
- Slope change is measured by fp''(X), i.e., convexity
- Since we are looking for the point where fp''(X) is maximum, x_{MC} is such that fp'''(x_{MC}) = 0
- Beware x_{MC} is not necessarily where fp(X) is steepest or even "too steep"



ICSEA 2016 - 58 - Setting Thresholds in Software Engineering

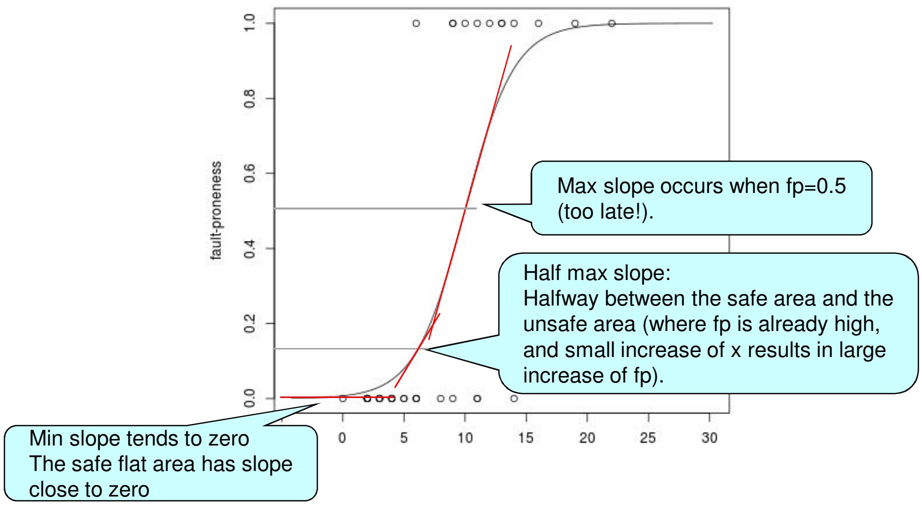
Proposal 2: Fraction of Maximum Slope

- It might be too late to wait until the curve has reached maximum slope fp'_{max}
- Define the threshold as the point x_{rMS} such that $fp'(x_{rMS})$ is a fraction r of fp'_{max}

$$fp'(x_{rMS}) = r\ fp'_{max}$$

- The value of r is set by the practitioners, based on their goals
- Via empirical studies, we found that r = 0.5 is a reasonable choice.
  - Hence, we look for $x_{MS/2}$, where the slope is half the maximum value.

Half maximum slope (MS/2)



MC threshold values

$$x_{MC} = \frac{1}{c_1}\left(\ln\left(2-\sqrt{3}\right) - c_0\right) \approx -\frac{1.55 + c_0}{c_1}$$

$$fp(x_{MC}) = \frac{1}{2} - \frac{\sqrt{3}}{6} \approx 0.2113$$

The maximum convexity is always positioned where fp=0.2113

MS/2 threshold values

$$x_{rMS} = \frac{1}{c_1}(\ln\left(\frac{1-\sqrt{1-r}}{1+\sqrt{1-r}}\right) - c_0)$$

$$fp(x_{rMS}) = \frac{1}{2} - \frac{\sqrt{1-r}}{2}$$

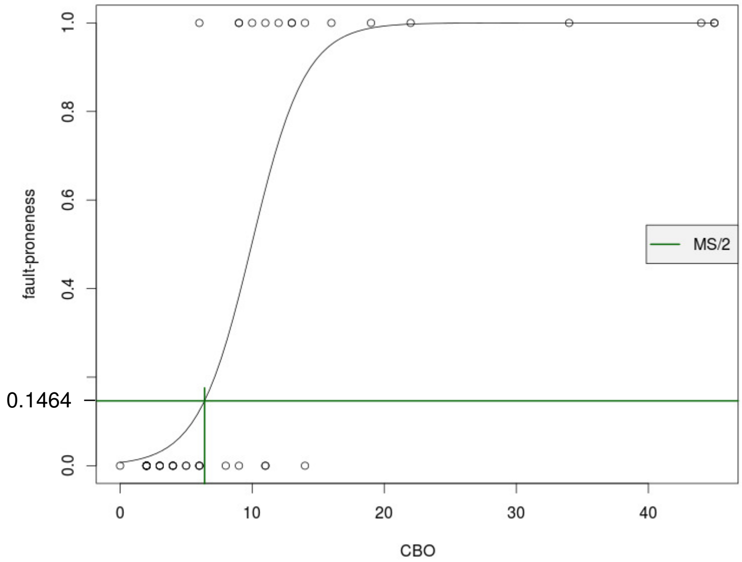
When r=0.5:

$$x_{MS/2} \approx -\frac{0.7656 + c_0}{c_1}$$

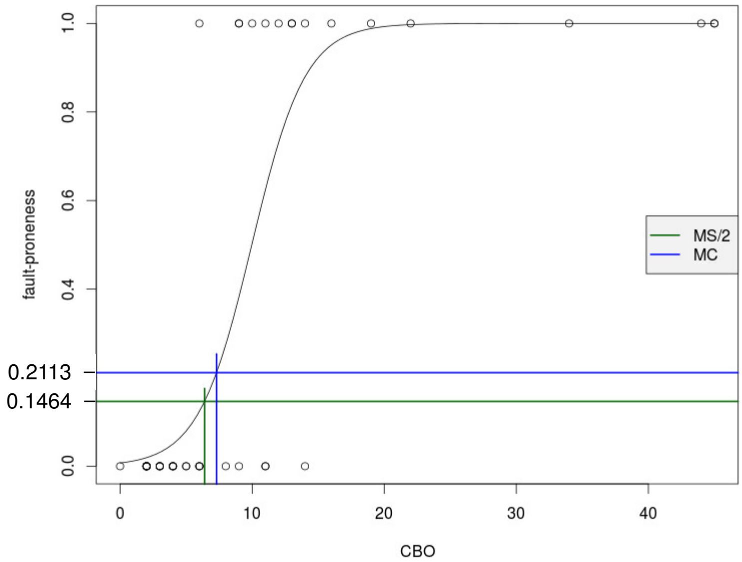
$$fp\left(x_{MS/2}\right) \approx 0.1464$$

The slope is always half the maximum when fp=0.1464

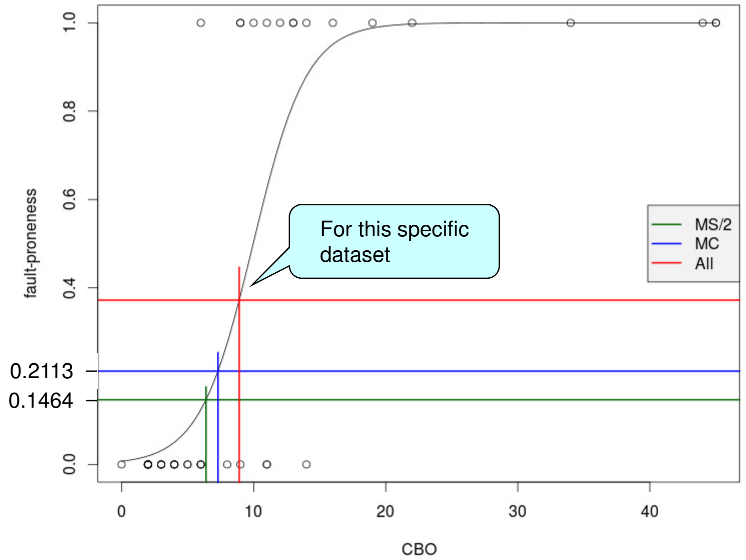
BLR Thresholds: MS/2



BLR Thresholds: MS/2, MC

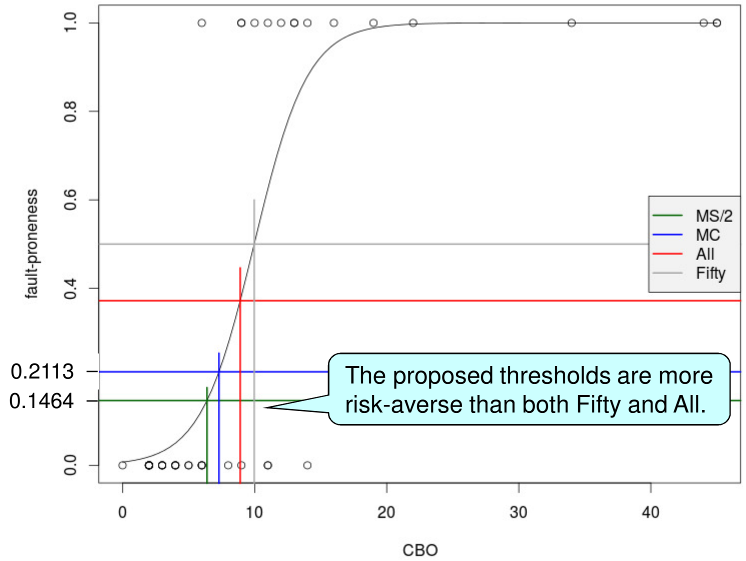


BLR Thresholds: MS/2, MC, All



ICSEA 2016 - 65 - Setting Thresholds in Software Engineering

BLR Thresholds: MS/2, MC, All, Fifty



ICSEA 2016 - 66 - Setting Thresholds in Software Engineering

What about PBR models?

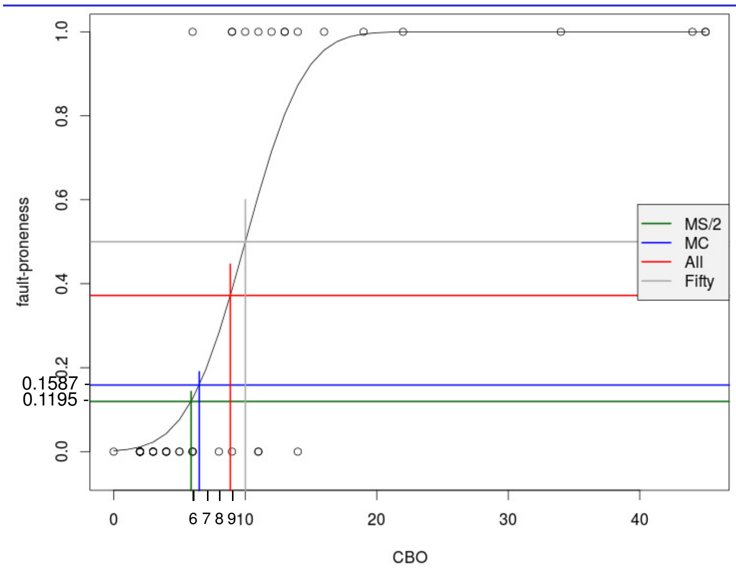
- Results of the mathematical analysis:
  - For any BLR model, maximum convexity occurs at the same values of fp.
  - For any BLR model, half maximum slope occurs at the same values of fp.
  - For any PBR model, maximum convexity occurs at the same values of fp.
  - For any PBR model, half maximum slope occurs at the same values of fp.

Fault-proneness values per type of model and type of threshold.

Model	MS/2	MC
PBR	0.1195	0.1587
BLR	0.1464	0.2113

- The values in the table above **apply to <u>all</u>** BLR and PBR models.

PBR Thresholds: MS/2, MC, All, Fifty



Empirical study

- We used real-life datasets hosted on the PROMISE repository, with data on
  - module actual faultiness
  - several independent variables
- We carried out T-time K-fold cross-validation
  - 10-time 10-fold cross-validation for larger datasets
  - 5-time 5-fold cross-validation for smaller datasets
- For each fold, we built statistically significant univariate BLR and PBR models for all internal attribute measures
- We computed overall average Precision, Recall, F-measure

Accuracy indicators

- Precision: proportion of estimated positives that are actually positive

$$Precision = \frac{TP}{EP}$$

- Recall: proportion of actual positives that are estimated positives

$$Recall = \frac{TP}{AP}$$

Recall indicates how risk-averse is the estimate.
Recall=1 means that all actual positives are estimated positive.

- F-measure: harmonic mean of Precision and Recall

$$FM = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Berek Dataset: Average F-measures with BLR

var	All	0.5	MC	MS/2
WMC	**0.80**	0.79	0.74	0.65
CBO	0.82	0.77	**0.83**	0.81
RFC	**0.88**	**0.88**	**0.88**	**0.88**
CA	**0.81**	0.77	0.78	0.75
CE	0.73	0.69	**0.81**	0.77
LOC	**0.91**	**0.91**	**0.91**	0.88
MOA	**0.69**	**0.69**	0.52	0.54
CAM	0.69	0.69	**0.75**	**0.75**
AMC	**0.73**	**0.73**	0.70	0.68
Max CC	0.71	0.69	0.65	0.64

In bold the result provide by the best threshold, for each model.

- There seems to be no best threshold: no threshold maximizes FM for all models.

Berek Dataset: Average Recall with BLR

var	All	0.5	MC	MS/2
WMC	0.75	0.69	0.81	**0.94**
CBO	0.88	0.75	**0.94**	**0.94**
RFC	**0.94**	0.88	**0.94**	**0.94**
CA	0.81	0.75	0.88	**0.94**
CE	0.75	0.63	**0.94**	**0.94**
LOC	**0.94**	**0.94**	**0.94**	**0.94**
MOA	0.56	0.56	0.75	**1.00**
CAM	0.75	0.69	**0.94**	**0.94**
AMC	0.75	0.69	**0.88**	**0.88**
Max CC	0.63	0.56	0.75	**0.94**

- MS/2 maximizes Recall for all models. It is the best threshold with respect to recall.
- MC provides similar performance (it is a bit less risk-averse)

Results for all datasets, with BLR

Best model for each dataset

Project	var.	n	AP/n	F-measure max	F-measure thresholds	recall max	recall thresholds
ckjm	LCOM	10	0.50	0.86	MS/2	1.00	MS/2
intercafe	CBO	27	0.15	0.86	0.5	0.75	All 0.5 MC MS/2
ivy-1.1	LCOM	111	0.57	0.80	MC MS/2	1.00	MC MS/2
lucene-2.2	NPM	247	0.58	0.79	MC MS/2	1.00	MC MS/2
lucene-2.4	RFC	340	0.60	0.75	MC MS/2	1.00	MC MS/2
nieruchomosci	MaxCC	27	0.37	0.89	MS/2	1.00	MS/2
pbeans1	LCOM	26	0.77	1.00	MC MS/2	1.00	MC MS/2
pdftranslator	LCOM	33	0.45	0.81	MC	1.00	MS/2
poi-1.5	LCOM	237	0.59	0.76	0.5	1.00	MC MS/2
poi-2.5	WMC	385	0.64	0.83	0.5	1.00	MC MS/2
poi-2.5	NPM	385	0.64	0.83	0.5	1.00	MC MS/2
poi-2.5	LCOM3	385	0.64	0.83	0.5	1.00	MC MS/2
poi-3.0	RFC	442	0.64	0.82	0.5	1.00	MC MS/2
poi-3.0	CE	442	0.64	0.82	tr	1.00	MC MS/2
sklebagd	WMC	20	0.60	0.92	MC	1.00	MC MS/2
szybkafucha	CBO	25	0.56	0.89	MC MS/2	0.80	MC MS/2
velocity-1.4	RFC	196	0.75	0.92	MC MS/2	1.00	MC MS/2
workflow	RFC	39	0.51	0.77	MC MS/2	1.00	MC MS/2
xerces-1.4	CBO	588	0.74	0.96	0.5	1.00	MC MS/2
xalan-2.5	NOC	803	0.48	0.70	MC MS/2	1.00	MC MS/2
zuzel	RFC	29	0.45	0.80	MC	0.92	MC MS/2
kalkulator	AMC	27	0.22	0.80	0.5	0.67	All 0.5 MC MS/2
wspomaganiepi	MOA	18	0.67	1.00	MC MS/2	1.00	MC MS/2

- MS/2 always maximizes Recall (and often also FM) !
- MC achieves similar results

Results for PBR

- For PBR models we got very similar results.

Summary of results

- MC and MS/2 have
  - almost always better Recall than the other thresholds
  - often better F-measure than the other thresholds
- The introduced thresholds are
  - suitable for identifying “early symptoms” of possible faultiness of a module
  - derived from properties of the fault-proneness model
  - computed automatically
  - quite accurate in terms of Recall and often F-measure too

Conclusions

- If you have a BLR or PBR model q(x) that relates an interesting external quality q to some internal measure x
- You can use the following thresholds on q

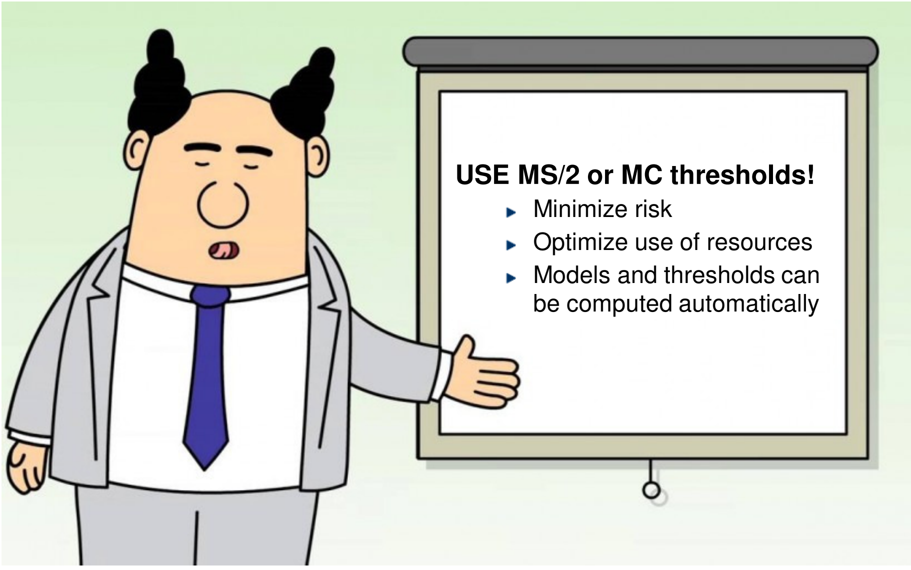
Model	MS/2	MC
PBR	0.1195	0.1587
BLR	0.1464	0.2113

These values apply to any q and any x!

to get risk-averse thresholds on x.

- According to our experimental results, you maximize the number of actually positive modules that are estimated positives, while you still get relatively few negative modules that are estimated positives.
- This means that you get an excellent trade-off between
  - the effectiveness of the development and maintenance effort
  - the costs of quality improvement
  - the costs of using faulty software

Conclusions



Contents

Topics
The context
The problem
Proposal 1: slope-based thresholds
➡ Proposal 2: optimistic-pessimistic approach
Proposal 3: fault-proneness H-index
Final considerations

The context

- In this case, we consider what happens at the end of the coding phase:
- You have a bunch of new modules and have to decide which of these modules deserve “special treatment” (e.g., code inspection) because they are likely faulty.

- Modules developed in the past --whose faultiness is known-- are the training set
- New modules --whose faultiness is unknown-- are the test set.

ICSEA 2016 - 79 - Setting Thresholds in Software Engineering

Conventional Approach

1. A fault-proneness model is derived from the training set
2. The model is used to estimate the test set
   - To this end, a threshold on fp can be set
     - based on local considerations
     - as $\frac{AP_{trainingSet}}{n_{trainingSet}}$

- When actual faultiness data on the test set become available the accuracy of the estimates can be computed.

ICSEA 2016 - 80 - Setting Thresholds in Software Engineering

The optimistic-pessimistic approach[2]

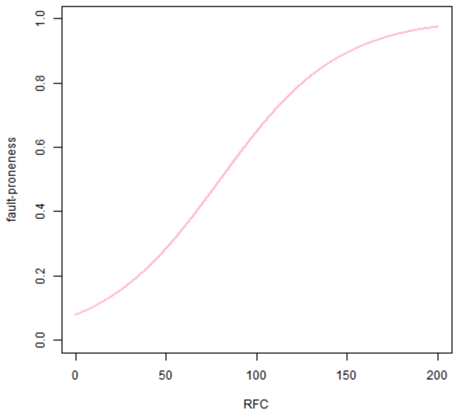
- The test set is used to build two models:
  - An optimistic one
  - A pessimistic one
- Where the models agree, you can be reasonably confident that the obtained classification is right.
- When the models disagree, you should better consider the faultiness of the module in question “uncertain”

[2]Luigi Lavazza and Sandro Morasca, “Identifying Thresholds for Software Faultiness via Optimistic and Pessimistic Estimations”, ESEM 2016

ICSEA 2016 - 81 - Setting Thresholds in Software Engineering

Building the optimistic model

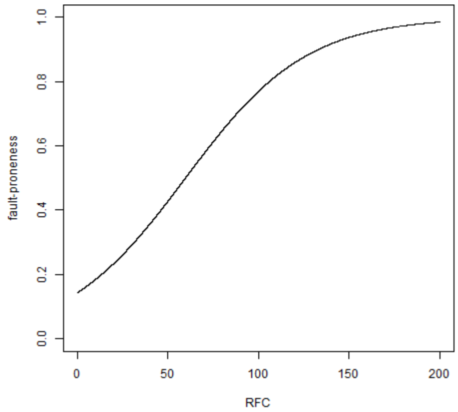
- All the modules in the test set are considered as not faulty
  - This is an optimistic assumption!
- You make the union of the training set and the test set
- You build a BLR model as usual
- The resulting model is optimistic, because of the initial optimistic assumption.



ICSEA 2016 - 82 - Setting Thresholds in Software Engineering

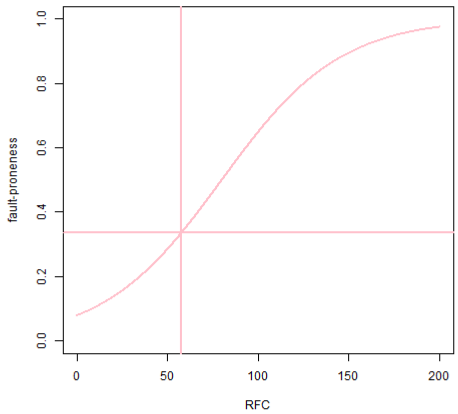
Building the pessimistic model

- All the modules in the test set are considered as not faulty
  - This is a pessimistic assumption!
- You make the union of the training set and the test set
- You build a BLR model as usual
- The resulting model is pessimistic, because of the initial pessimistic assumption.



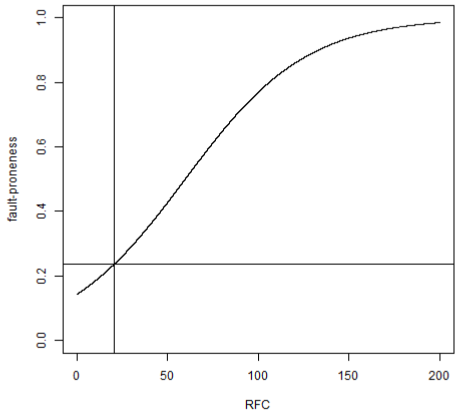
Optimistic Model Threshold and Optimistic Estimated Faultiness Model

- Select a threshold for the optimistic model and build an optimistic estimated faultiness model



Pessimistic Model Threshold and Pessimistic Estimated Faultiness Model

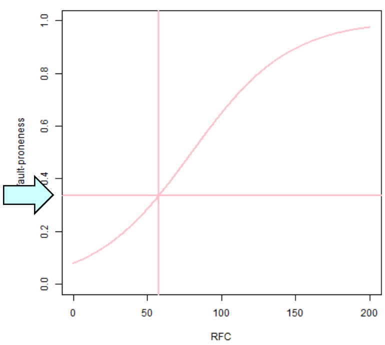
- Select a threshold for the pessimistic model and build a pessimistic estimated faultiness model

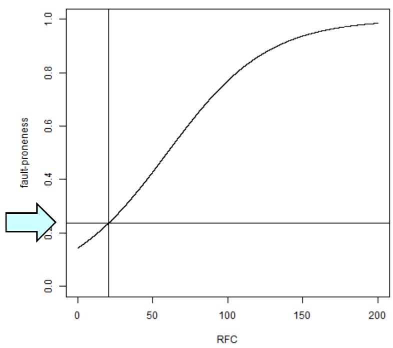


ICSEA 2016 - 85 - Setting Thresholds in Software Engineering

Where to place fp thresholds?

- As usual, we have to decide where to place thresholds for fault-proneness.





ICSEA 2016 - 86 - Setting Thresholds in Software Engineering

Possible fp thresholds

- Several thresholds are possible:
  - Pessimistic threshold: fraction of modules that are known to be positive

$$t_{pess} = \frac{AP_{trainingSet}}{n_{trainingSet} + n_{testSet}}$$

  - Optimistic threshold: fraction of modules that are known to be positive or are unknown

$$t_{opt} = \frac{AP_{trainingSet} + UK}{n_{trainingSet} + n_{testSet}}$$

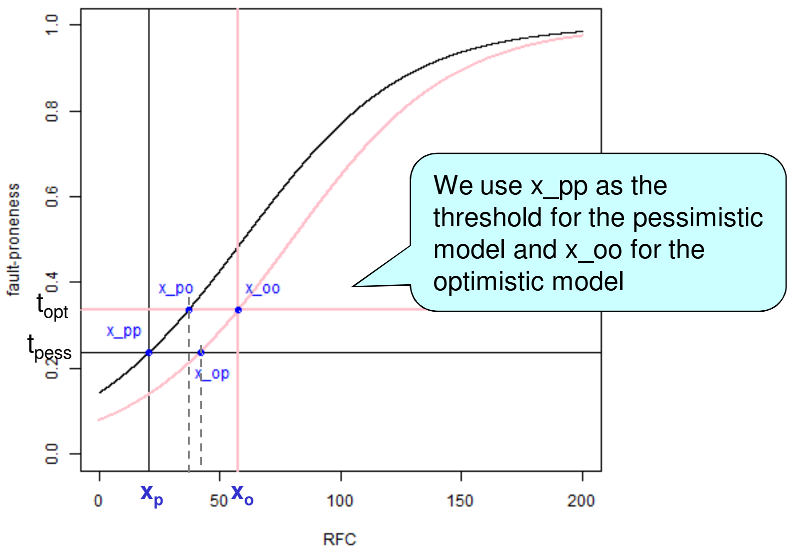
UK is the number of unknown, i.e., $n_{testSet}$

  - Neutral threshold:

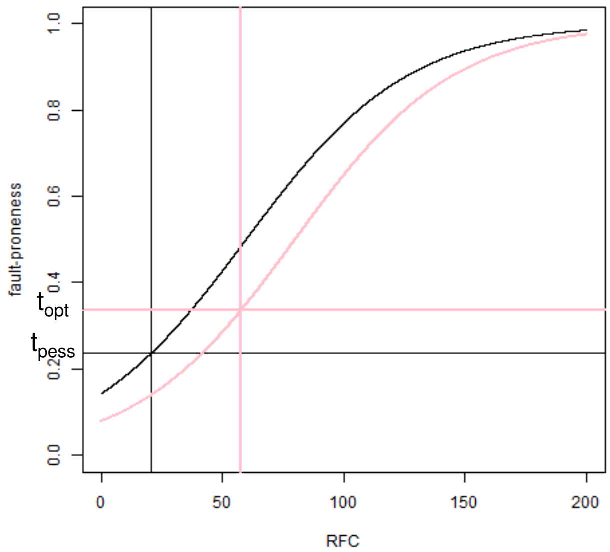
$$t_{neut} = \frac{AP_{trainingSet}}{n_{trainingSet}}$$

- Note that $t_{opt} > t_{neut} > t_{pess}$

Intersections

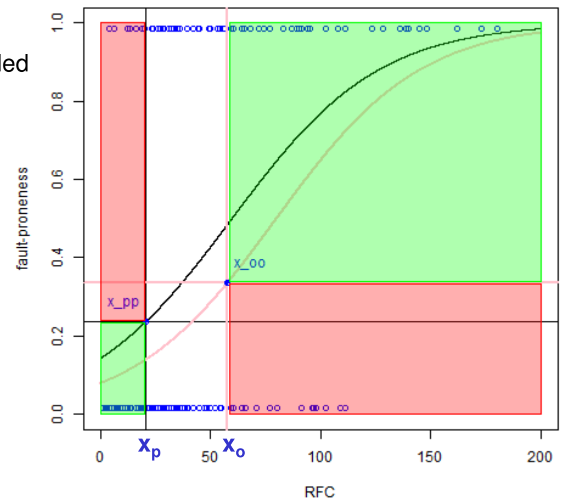


Optimistic and Pessimistic Thresholds and Models

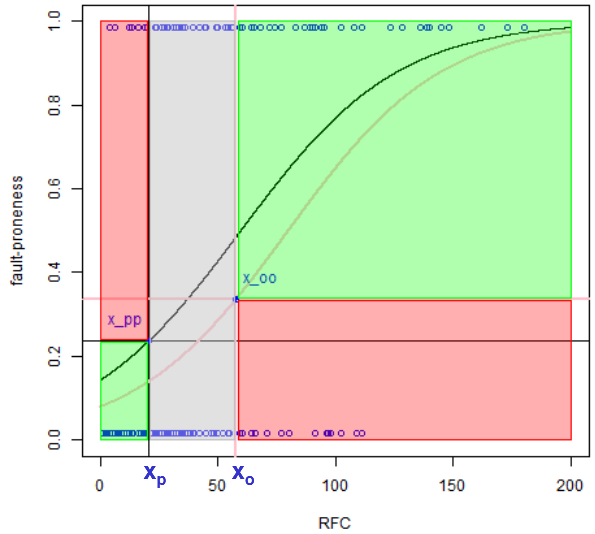


Classification using the optimistic-pessimistic approach

- Modules are classified as follows:
  - $x \le x_p \Rightarrow$ negative
  - $x \ge x_o \Rightarrow$ positive
  - $x_p < x < x_o \Rightarrow$ undecided



Grey Zone



Other estimation approaches

- The reference approach
  - The test set is classified based on the model derived from the training set
- The pessimistic model approach alone
  - $x \le x_{pp} \Rightarrow$ negative
  - $x > x_{pp} \Rightarrow$ positive
- The optimistic model approach alone
  - $x \le x_{oo} \Rightarrow$ negative
  - $x > x_{oo} \Rightarrow$ positive

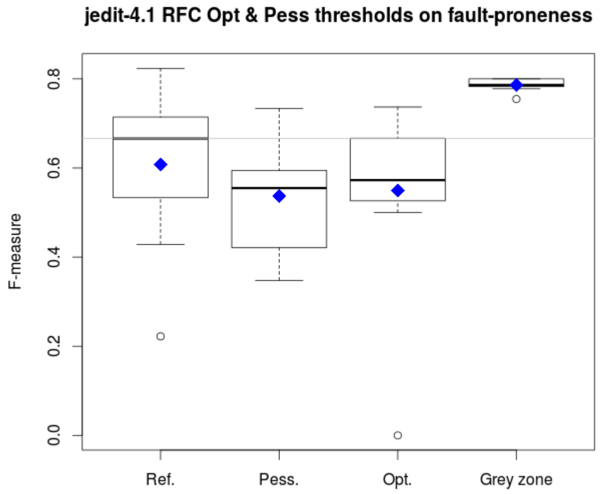
Comparison

- We compared the classification obtained using the optimistic-pessimistic approach with the classifications obtained using other approaches.
- Note: when considering the optimistic-pessimistic approach, only classified modules were considered in the computation of the accuracy indicators.

Data sets

- We used 48 real-life datasets hosted on the PROMISE repository
- We carried out 10-fold cross-validation
- We almost always obtained the best results with
  - $x_p = x_{pp}$ and $x_o = x_{oo}$, or
  - $x_p = x_{po}$ and $x_o = x_{oo}$

Results of the comparison



Conclusion

- By means of the traditional approach you get quite variable results, because modules in the grey zone are classified as either faulty or not faulty anyway.
- With the optimistic-pessimistic approach the modules in the grey zone are not estimated, thus avoiding many classification errors.
- Note: if a module is in the grey zone of the CBO models, it could very well be out of the grey zone of the RFC model …

Contents

Topics
The context
The problem
Proposal 1: slope-based thresholds
Proposal 2: optimistic-pessimistic approach
Proposal 3: fault-proneness H-index
Final considerations

Proposal[3]

- A new approach to building an estimated faultiness model based on the definition of the Fault-proneness H-Index, an extension to the H-index
- Basic idea
  - the H-Index identifies the most important papers of a researcher
  - the Fault-proneness H-Index identifies the most fault-prone modules in a set of modules
- Advantage
  - we do not need to set a threshold ourselves, but the threshold is derived from the data

[3] Sandro Morasca, “Classifying Faulty Modules with an Extension of the H-index,” ISSRE 2015

H-index computation

- Order absolute frequencies af(z) in decreasing order
- Set z = 0 as the initial value of the H-Index
- Increase the value of z by 1 as long as af(z) ≥ z
- The value of the H-index is the last value z such that af(z) ≥ z
- The value of *h* can be found at the intersection of two functions
  - af(z), which is decreasing
  - z, which is linearly increasing

My H-index

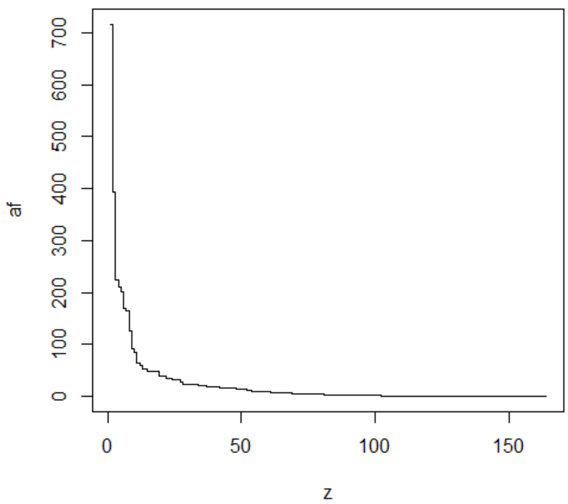
Rank	Title	Citations	Authors	Journal/boo
1	SPADE: An environment for software process analysis, design, and enactment	196	S Bandinelli, A Fuggetta, C Ghezzi, L Lavazza	Software pr
2	Modeling and improving an industrial software process	156	S Bandinelli, A Fuggetta, L Lavazza, M Loi, GP Pic	IEEE Transac
3	A conceptual basis for feature engineering	146	C Reid Turner, A Fuggetta, L Lavazza, AL Wolf	The Journal
4	Deriving executable process descriptions from UML	122	E Di Nitto, L Lavazza, M Schiavoni, E Tracanella, .	Proceedings
5	Combining UML and formal notations for modelling real-time systems	89	L Lavazza, G Quaroni, M Venturelli	ACM SIGSOF
6	Applying GQM in an industrial software factory	66	A Fuggetta, L Lavazza, S Morasca, S Cinti, G ...	ACM Transa
7	The architecture of SPADE-1 process-centered SEE	61	S Bandinelli, M Braga, A Fuggetta, L Lavazza	Lecture Note
8	Translation and optimization of logic queries: the algebraic approach	56	S Ceri, G Gottlob, L Lavazza	Proceedings
9	The GOODSTEP Project: General Object-Oriented Database for Software Engineering	48	The GOODSTEP Team	APSEC'94
10	Algres: an advanced database system for complex applications	50	S Ceri, S Crespi-Reghizzi, R Zicari, G Lamperti, LA	IEEE Softwar
11	Providing automated support for the GQM measurement process	52	L Lavazza	IEEE Softwar
12	OpenBQR: a framework for the assessment of OSS	50	Davide Taibi, Luigi Lavazza and Sandro Morasca	OSS 2007
13	Feature engineering	44	CR Turner, AL Wolf, A Fuggetta, L Lavazza	Proceedings
14	An experience in process assessment	40	F Cattaneo, A Fuggetta, L Lavazza	Proceedings
15	Combining Problem Frames and UML in the Description of Software Requirements	30	L Lavazza, V. Del Bianco	FASE 2006
16	SystemC/C-based model-driven design for embedded systems	29	Riccobene, Scandurra, Bocchio, Rosti, Lavazza, M	TECS
17	Model-based functional size measurement	33	Lavazza, Del Bianco, Garavaglia	ESEM 2008
18	Enhancing Requirements and Change Management through Process Modelling	31	Lavazza, Valetto	ICRE 2000
19	A UML-based approach for representing problem frames	25	L Lavazza, V. Del Bianco	IEE Seminar
20	A case study in COSMIC functional size measurement: The rice cooker revisited	27	L Lavazza, V Del Bianco	Software Pr
21	Automated support for process-aware definition and execution of measurement pl	25	Lavazza, Barresi	ICSE2005
22	Automated Measurement of UML Models: an open toolset approach	23	L Lavazza, A Agostini	J. of Object T
23	Requirements-based estimation of change costs	22	L Lavazza, G Valetto	
24	An investigation of the users' perception of OSS quality	21	Del Bianco, Vieri, Luigi Lavazza, Sandro Morasca,	OSS 2010
25	Model checking UML specifications of real time software	24	Del Bianco, V. Lavazza, L. Mauri, M.	ICECCS 2002
26	A Survey on Open Source Software Trustworthiness	21	Del Bianco, Vieri, Luigi Lavazza, Sandro Morasca,	IEEE SW
27	Managing software artifacts on the Web with Labyrinth	21	Cattaneo, Fabiano, Elisabetta Di Nitto, Alfonso F	ICSE 2000
28	Quality of Open Source Software: The QualiPSo Trustworthiness Model	19	Del Bianco, V. and Lavazza, L. and Morasca, S. ar	OSS 2009

My H-index

Rank	Title	Citations	Authors	Journal/book/conference
1	SPADE: An environment for software process analysis, design, and enactment	196	S Bandinelli, A Fuggetta, C Ghezzi, L Lavazza	Software process modelling
2	Modeling and improving an industrial software process	156	S Bandinelli, A Fuggetta, L Lavazza, M Loi, GP Pi	IEEE Transactions on Softwa
3	A conceptual basis for feature engineering	146	C Reid Turner, A Fuggetta, L Lavazza, AL Wolf	The Journal of Systems & So
4	Deriving executable process descriptions from UML	122	E Di Nitto, L Lavazza, M Schiavoni, E Tracanella,	Proceedings of the 24th Int.
5	Combining UML and formal notations for modelling real-time systems	89	L Lavazza, G Quaroni, M Venturelli	ACM SIGSOFT Software Engi
6	Applying GQM in an industrial software factory	66	A Fuggetta, L Lavazza, S Morasca, S Cinti, G ...	ACM Transactions on Softwa
7	The architecture of SPADE-1 process-centered SEE	61	S Bandinelli, M Braga, A Fuggetta, L Lavazza	Lecture Notes in Computer S
8	Translation and optimization of logic queries: the algebraic approach	56	S Ceri, G Gottlob, L Lavazza	Proceedings of the 12th Int.
9	Providing automated support for the GQM measurement process	52	L Lavazza	IEEE Software
10	Algres: an advanced database system for complex applications	50	S Ceri, S Crespi-Reghizzi, R Zicari, G Lamperti, LA	IEEE Software
11	OpenBQR: a framework for the assessment of OSS	50	Davide Taibi, Luigi Lavazza and Sandro Morasca	OSS 2007
12	The GOODSTEP Project: General Object-Oriented Database for Software Engineerin	48	The GOODSTEP Team	APSEC'94
13	Feature engineering	44	CR Turner, AL Wolf, A Fuggetta, L Lavazza	Proceedings of the 9th inter
14	An experience in process assessment	40	F Cattaneo, A Fuggetta, L Lavazza	Proceedings of the 17th Int.
15	Model-based functional size measurement	33	Lavazza, Del Bianco, Garavaglia	ESEM 2008
16	Enhancing Requirements and Change Management through Process Modelling	31	Lavazza, Valetto	ICRE 2000
17	Combining Problem Frames and UML in the Description of Software Requirements	30	L Lavazza, V. Del Bianco	FASE 2006
18	SystemC/C-based model-driven design for embedded systems	29	Riccobene, Scandurra, Bocchio, Rosti, Lavazza, M	TECS
19	A case study in COSMIC functional size measurement: The rice cooker revisited	27	L Lavazza, V Del Bianco	Software Process and Produ
20	A UML-based approach for representing problem frames	25	L Lavazza, V. Del Bianco	IEE Seminar Digests (IWAAPI
21	Automated support for process-aware definition and execution of measurement pl	25	Lavazza, Barresi	ICSE2005
22	Model checking UML specifications of real time software	24	Del Bianco, V. Lavazza, L. Mauri, M.	ICECCS 2002
23	Automated Measurement of UML Models: an open toolset approach	23	L Lavazza, A Agostini	J. of Object Technology
24	Requirements-based estimation of change costs	22	L Lavazza, G Valetto	
25	An investigation of the users' perception of OSS quality	21	Del Bianco, Vieri, Luigi Lavazza, Sandro Morasca	OSS 2010
26	A Survey on Open Source Software Trustworthiness	21	Del Bianco, Vieri, Luigi Lavazza, Sandro Morasca	IEEE SW
27	Managing software artifacts on the Web with Labyrinth	21	Cattaneo, Fabiano, Elisabetta Di Nitto, Alfonso I	ICSE 2000
28	Quality of Open Source Software: The QualiPSo Trustworthiness Model	19	Del Bianco, V. and Lavazza, L. and Morasca, S. a	OSS 2009

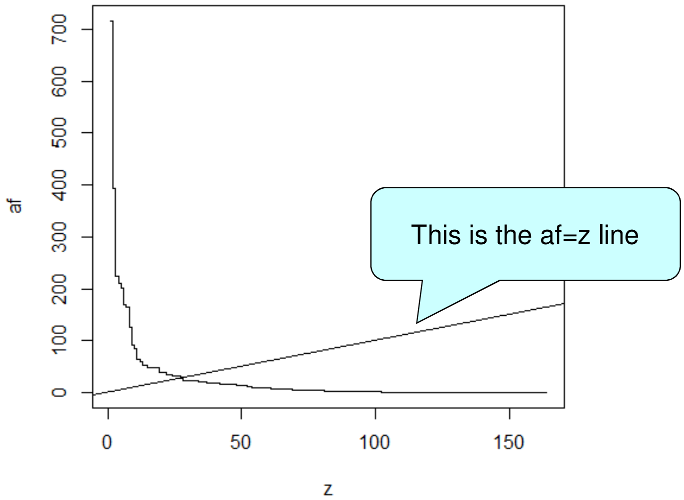
ICSEA 2016 - 101 - Setting Thresholds in Software Engineering

H-Index Graphical Representation



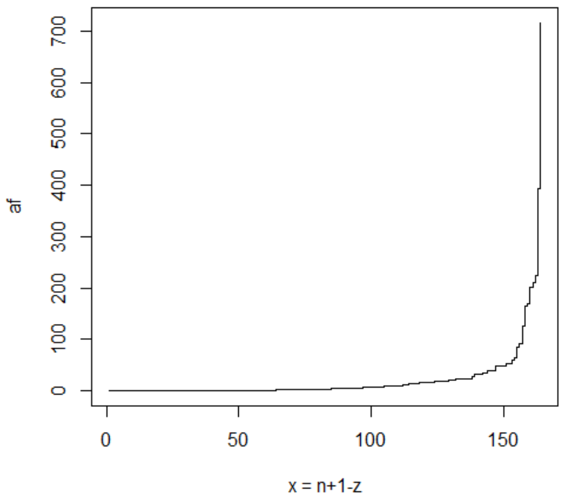
ICSEA 2016 - 102 - Setting Thresholds in Software Engineering

H-Index Graphical Representation



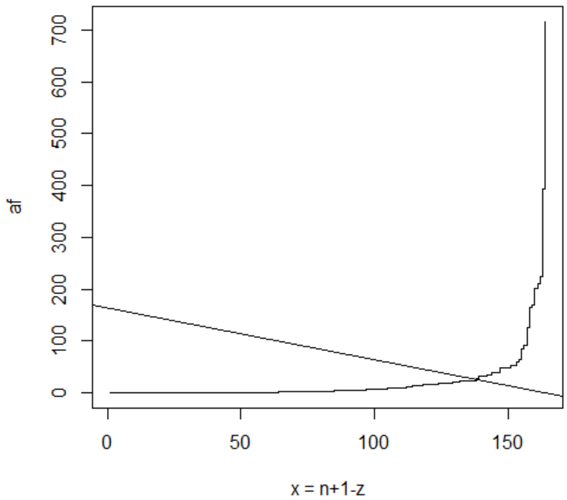
ICSEA 2016 - 103 - Setting Thresholds in Software Engineering

H-Index Graphical Representation



ICSEA 2016 - 104 - Setting Thresholds in Software Engineering

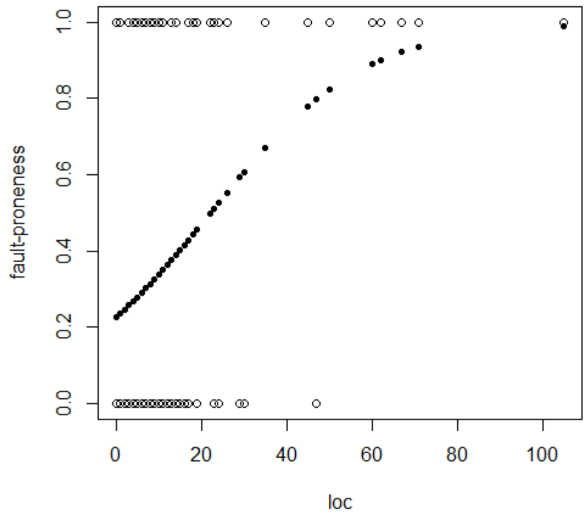
H-Index Graphical Representation



Fault-proneness H-Index Computation

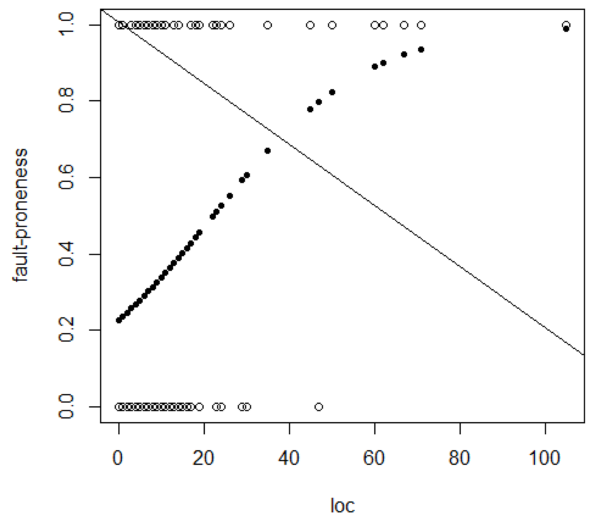
- Order the modules in decreasing order of estimated fault-proneness FP
- Set z = 0 as the initial value of the FPH-Index
- Increase the value of z by 1=n as long as $FP(x_m) \geq z/n$
- The value of *fph* is the last value of $FP(x_m)$ for which $FP(x_m) \geq z/n$ holds
- The value of *fph* can be found at the intersection of two functions
  - $FP(x_m)$, which is decreasing with z
  - z/n, which is linearly increasing

Fault-proneness H-Index Graphical Representation



ICSEA 2016 - 107 - Setting Thresholds in Software Engineering

Fault-proneness H-Index Graphical Representation



ICSEA 2016 - 108 - Setting Thresholds in Software Engineering

Results

- The H-index-based estimation technique
  - has generally higher values of Recall
  - has generally lower values of Precision
  - has generally higher values of F-measure when the weight of Recall is comparatively high

Contents

Estimates based uniquely on internal measures

- Do we get good results (i.e., accurate estimates) with this strategy?
- Not really[4].
- Let's see some experimental results.

[4] L. Lavazza, S. Morasca, "An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measure", *PROMISE 2016*.

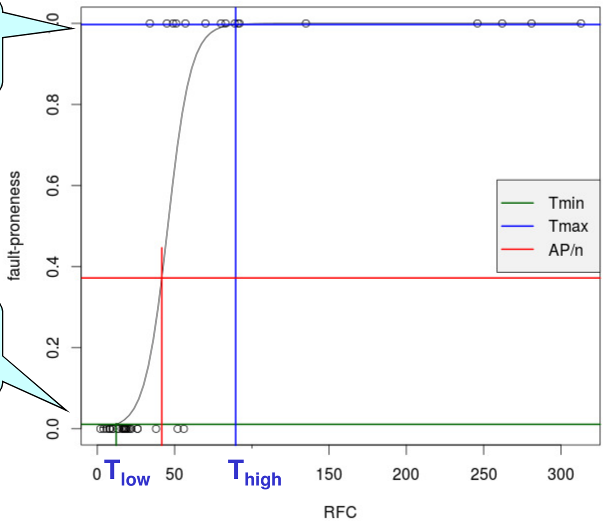
Estimates based uniquely on internal measures

- Let us consider the proposal by Erni & Lewerentz (or by Lanza and Marinescu)
- $T_{low} = \mu - \sigma$
- $T_{high} = \mu + \sigma$
- Where μ is the mean and σ is the standard deviation
- The threshold do not depend on faultiness data, but just in internal measures.
- What happens when we take into consideration faultiness data?
- Let's see how the thresholds behave in fault-proneness models.

Estimates based uniquely on internal measures

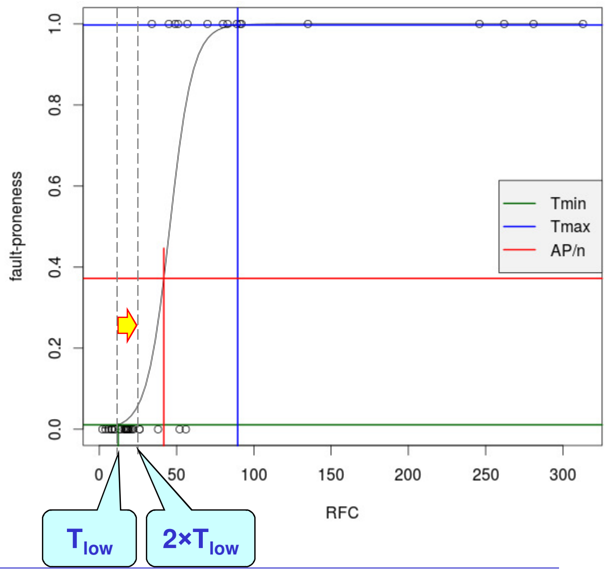
fp(T_{high}) is excessively high!

fp(T_{low}) is excessively low!



Estimates based uniquely on internal measures

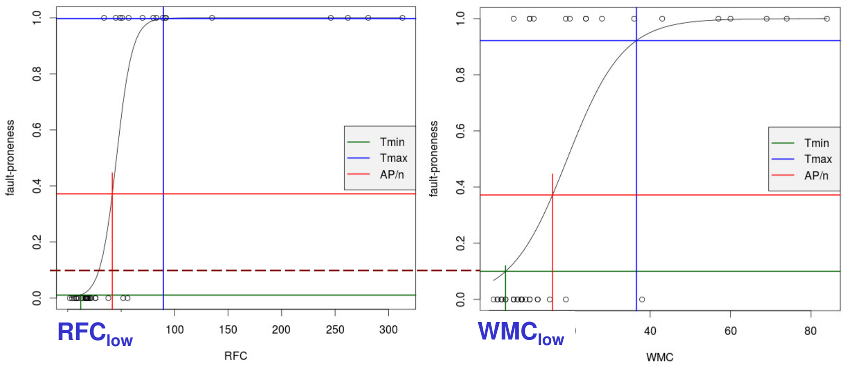
- T_{low} is excessively low: by doubling it you our estimates improve in every respect:
  - TN increases substantially
  - TP and FN do not change
  - FP decreases substantially



Estimates based uniquely on internal measures

- Even though the low thresholds for RFC and WMC are computed in the same way, they give very different results.
  - If you use RFC_{low} you get fp < 0.01
  - If you use WMC_{low} you get 0.1 (circa)

For these models.
The difference could be larger!



ICSEA 2016 - 115 - Setting Thresholds in Software Engineering

Conclusions

- There are many different ways of setting thresholds
- I would recommend methods based on information about internal measures <u>and</u> faultiness information
- Which one is the best?
  - Time will tell . . .
  - Does a “best” method really exist?

ICSEA 2016 - 116 - Setting Thresholds in Software Engineering

Future work

- A lot . . .