Software estimation – cost and quality

Mar 2022

Anh Nguyen Duc

Agenda

- ❖ The role of project estimation
- ❖ Traditional approaches for effort estimation
- ❖ Machine Learning in effort prediction
- ❖ Defect prediction models
- ❖ Within vs. Between project prediction
- ❖ Just-in-time defect prediction

What is a Project Estimate?

A declaration about needed

- cost and
- time
- quality
- human skill
- for delivering the project scope

How productive is this team/ developer in this project?

- What else?

Importance of project estimation

- ✓ Ressource allocation decisions
- ✓ Basis for the decision to start (or not to start)a project
- ✓ Foundation for project planning and set-up (business case)
- ✓ Foundation for project controlling
- ✓ If project time is a given,number of ressources can be determined
- ✓ Owner of an estimate is an indication about who is taking the project risk
- ✓ Decision and ressource allocation implications => Estimates are often part of political games
- ✓ Estimating is a core task of project management

Challenges for estimation

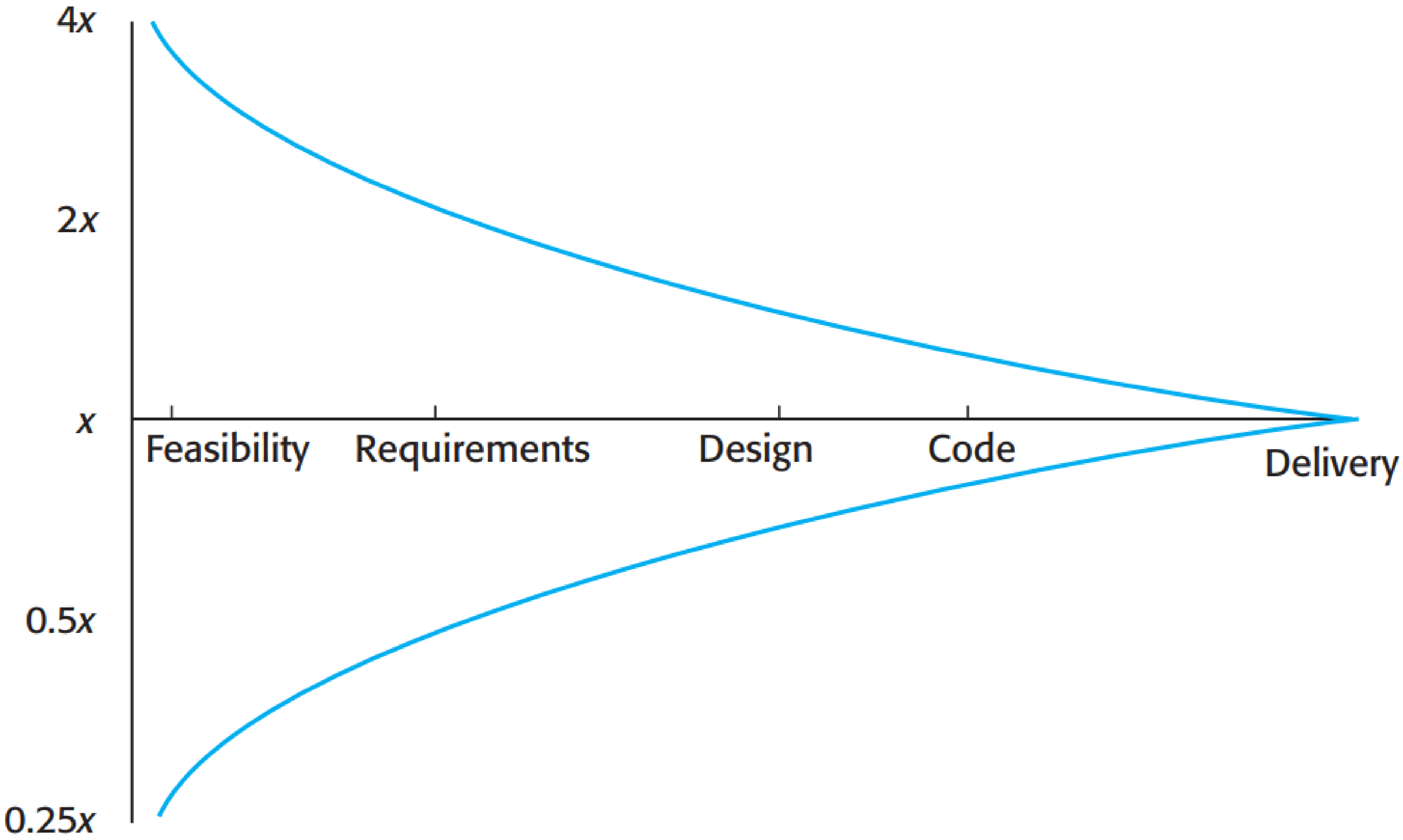
- ✓ Incomplete knowledge about:
  - ❖ *Project scope and changes*
  - ❖ *Prospective resources and staffing*
  - ❖ *Technical and organizational environment*
  - ❖ *Infrastructure*
  - ❖ *Feasibility of functional requirements*
- ✓ *Comparability of projects in case of new or changing technologies, staff, methodologies*
- ✓ *Learning curve problem*
- ✓ *Different expectations towards project manager*

Challenges for estimation

- ✓ Estimation is too low
  - ✓ *Scope and tasks(WBS) incomplete/unknown*
- ✓ Estimation is too high
  - ✓ *Political / human reasons*
  - ✓ *Learning curve*
- ✓ New technologies can make new parameters necessary

Challenges for estimation

Estimation uncertainty



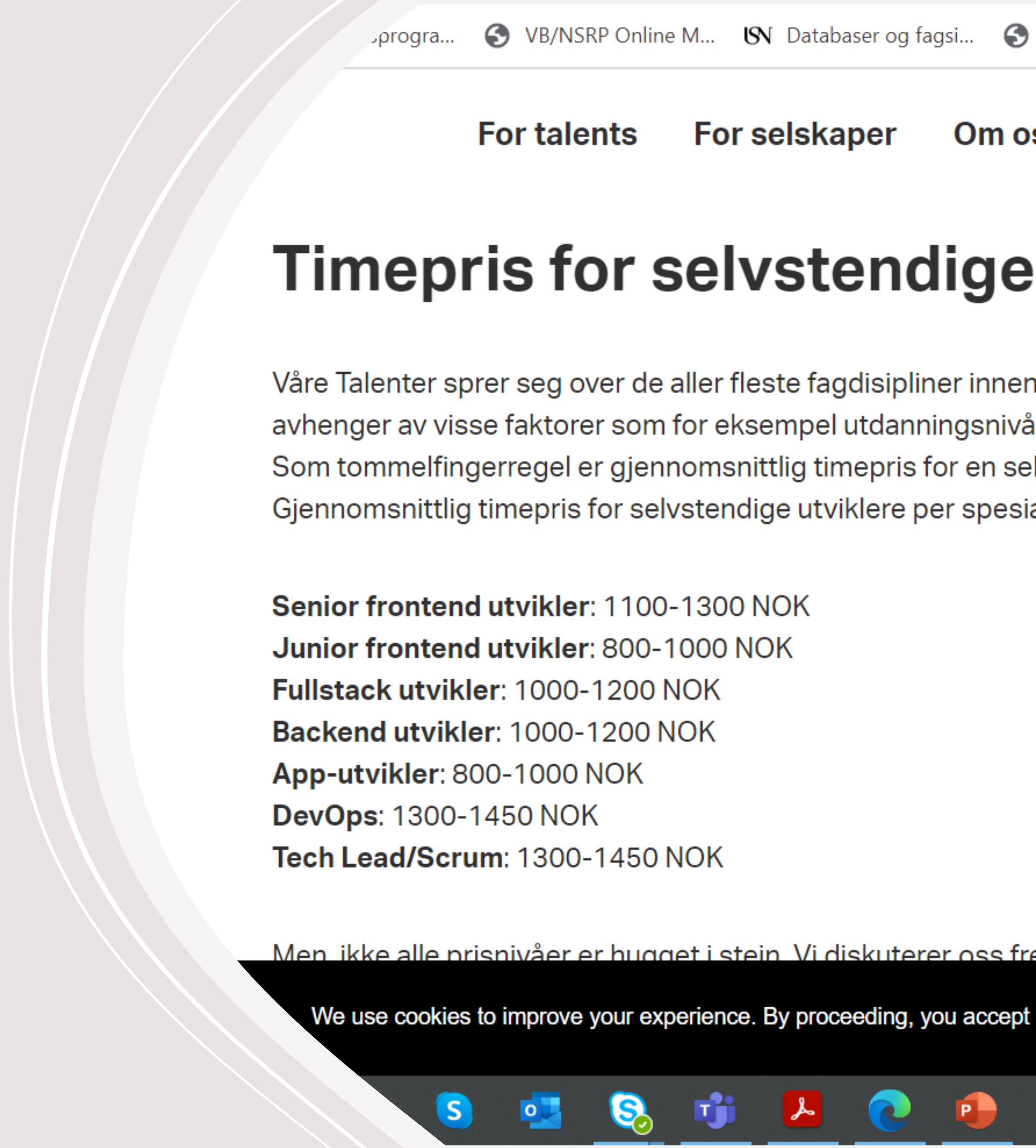
Estimating Effort

Cost Estimation

- ✓ Effort costs (the dominant factor in most projects)
    - ✓ – salaries of engineers involved in the project
    - ✓ – costs of building, heating, lighting
    - ✓ – costs of networking and communications
    - ✓ – costs of shared facilities (e.g library, staff restaurant, etc.)
    - ✓ – costs of pensions, health insurance, etc.
- ✓ Other costs
    - ✓ – *Hardware and software costs*
    - ✓ – *Travel and training costs*

Cost Estimation

- ✓ Staff categories (based on experience, qualification and skills),for example:
  - ✓ teamlead, junior business analyst, senior business analyst, junior programmer, senior programmer, subject matter expert
- ✓ Cost rate: Cost per person per day
  - ✓ 2 alternatives for cost rate:
    - ✓ *Single cost ratef or all types(no differentiation necessary)*
    - ✓ *Assign different cost rates to different categories*
- ✓ Personnel cost = person days x cost rate



Estimating effort- Basic principles

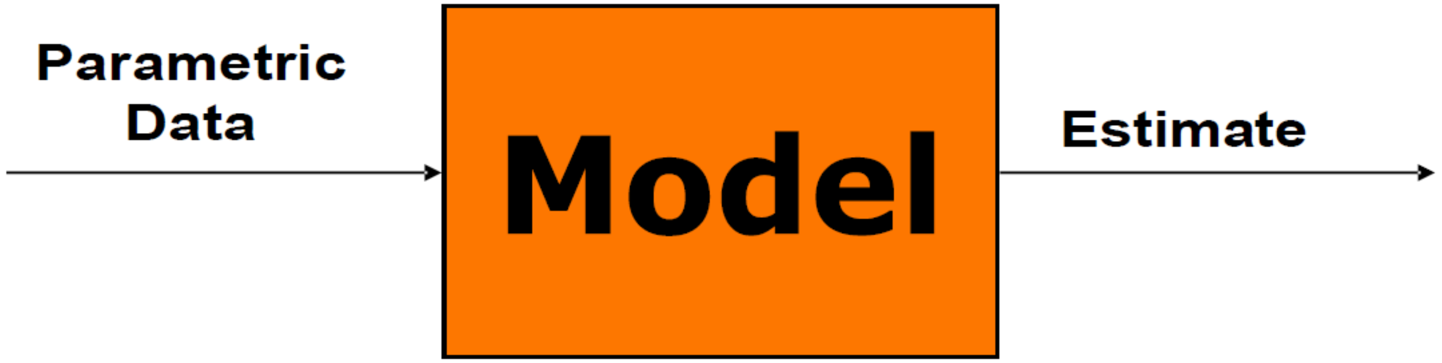
Select an estimation model (or build one first)

Evaluate known information: project scope,resources, software process (for example documentation requirements),system components

Feed this information as parametric input data into the model

Model converts the input into an estimate about the effort

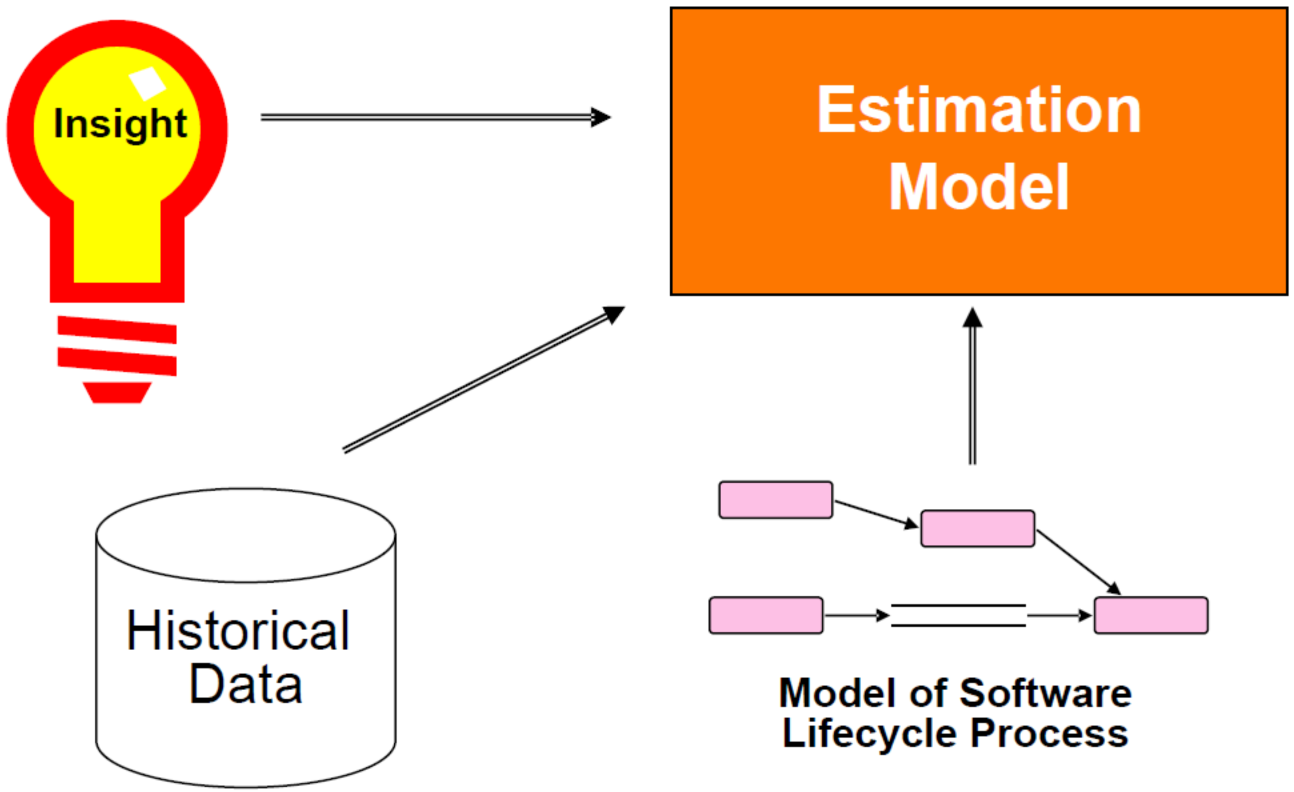
Basic of an Estimation model



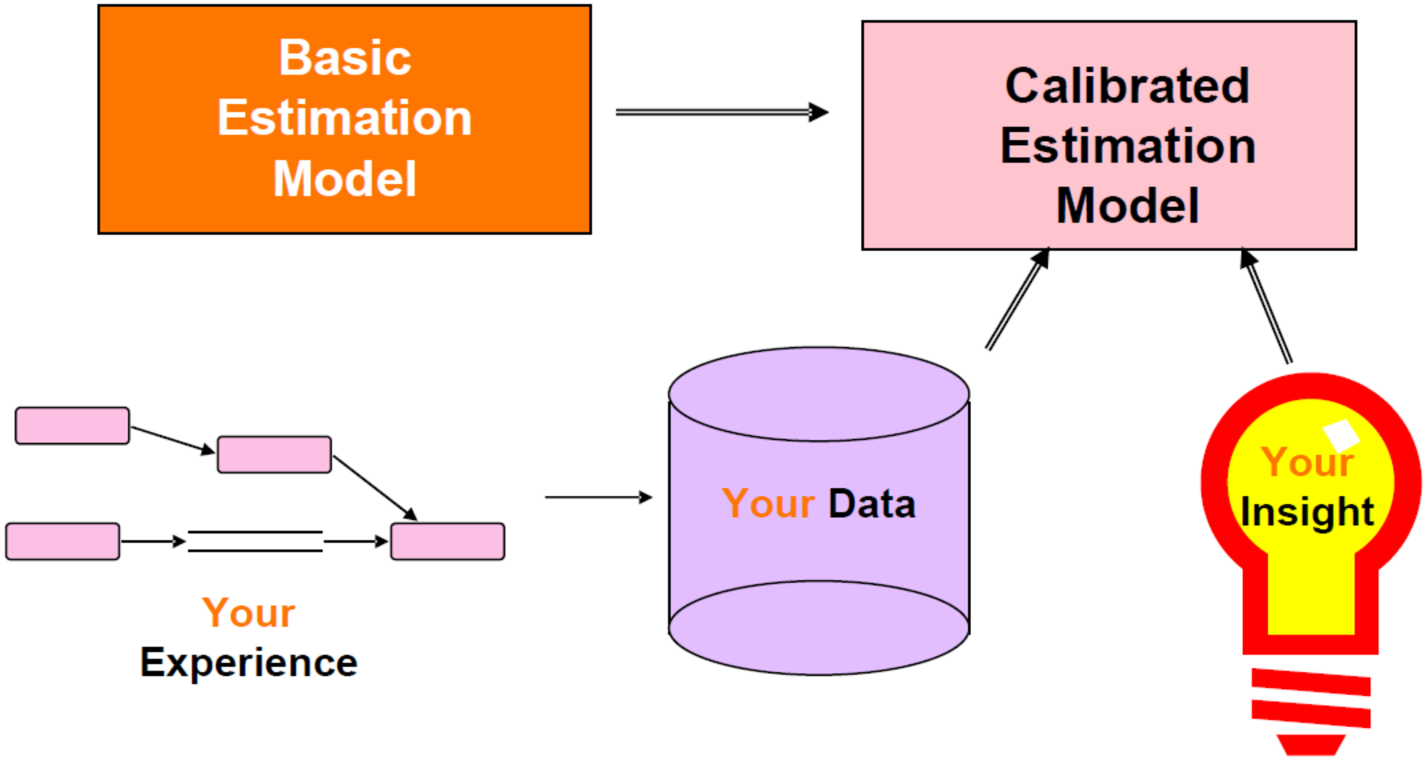
Examples:

Data Input	Estimate
Size & Project Data	Effort & Schedule
System Model	Performance
Software Process	Cycle Time.

How to build an estimation model?



Calibrating the model



Top-Down and Bottom-Up Estimation

Two common approaches for estimations

Top-Down Approach

- Estimate effort for the whole project
- Breakdown to different project phases and work products

Bottom-Up Approach

- Start with effort estimates for tasks on the lowest possible level
- Aggregate the estimates until top activities are reached

Top-Down and Bottom-Up Estimation

Top-Down Approach

- Normally used in the planning phase when little information is available how to solve the problem
- Based on experiences from similar projects
- Not appropriate for project controlling (too high-level)
- Risk add-ons usual as result tends to be too low

Bottom-Up Approach

- Normally used after activities are broken down to task level and estimates for the tasks are available
- Result can be used for project controlling (detailed level)
- Smaller risk add-ons (tends to be too high)
- Often a mixed approach with recurring estimation cycles is used.

Estimation Techniques

- ✓ Expert judgement
- ✓ Estimation by analogy
- ✓ Parkinson's Law
- ✓ Pricing to win
- ✓ Lines of code
- ✓ Function point analysis
- ✓ Algorithmic cost modelling
- ✓ COCOMO

Expert judgement

= Guess from experienced people

- Mostly used top-down for the whole project, but also for some parts of a bottom-up approach
- Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- No better than the participants
- Result justification difficult
- Very inaccurate if there are no experts!

Estimation by analogy

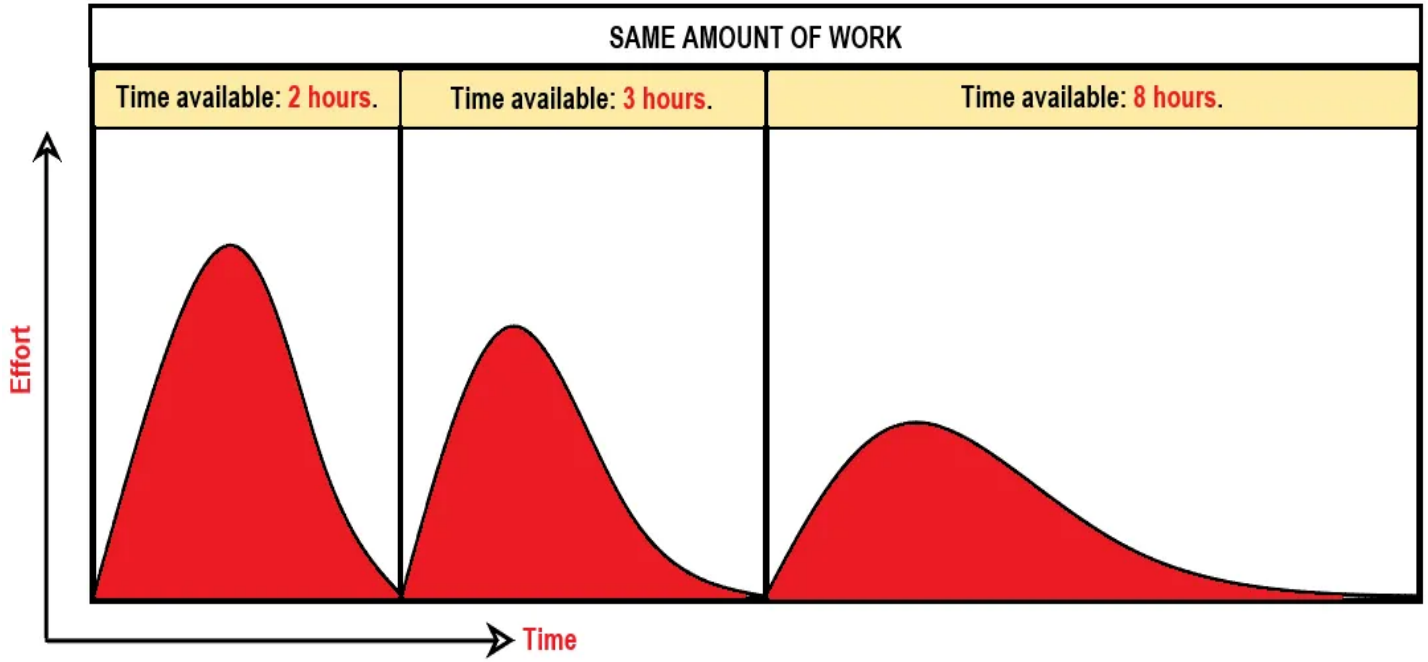
(do you work on the same project before?)

- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Accurate if project data available
- Impossible if no comparable project has been tackled. Needs systematically maintained cost database

Parkinson’s Law

“work expands to fill the time allotted for its completion”

- The project costs whatever resources (people, money, time) are available
- No overspend
- System is usually unfinished



Pricing to win

The project costs whatever the customer has to spend on it

- You get the contract
- The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required

Line of Code (LOC)

Traditional way for estimating application size (FORTRAN and assembler -> line-oriented languages)

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Line of Code (LOC)

Traditional way for estimating application size (FORTRAN and assembler -> line-oriented languages)

Advantage: Easy to do

Disadvantages:

- No standard definition for Line of Code (logical versus physical)
- Of no help given a written project scope or functional design
- You get what you measure: If the number of lines of code is the primary measure of productivity, programmers ignore opportunities of reuse
- Multi-language environments: Hard to compare mixed language projects with single language projects

The use of lines of code metrics for productivity should be regarded as professional malpractice (Caspers Jones).

Function Point Estimation

Based on FP metric for the size of a product

Based on the number of inputs (Inp), outputs (Out), inquiries (Inq), master files (Maf), interfaces (Inf)

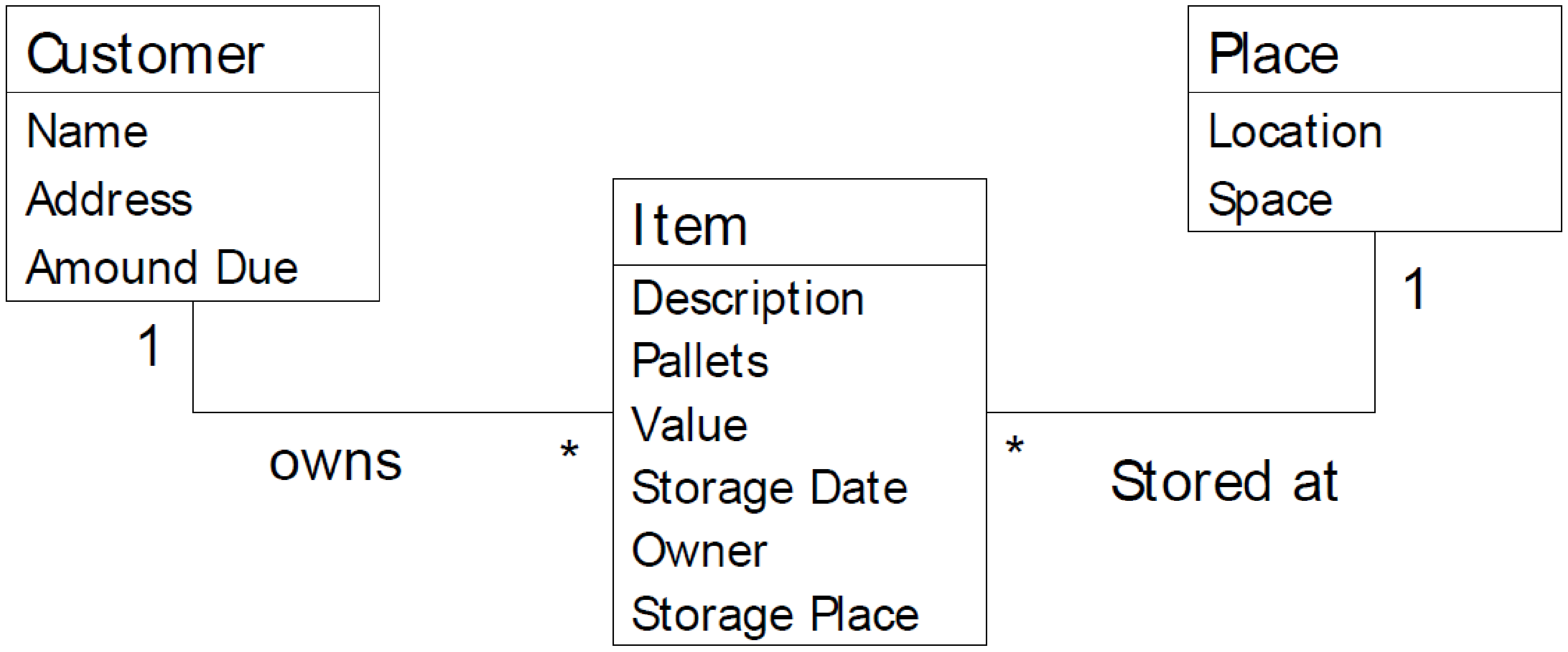
Step 1: Classify each component of the product (Inp, Out, Inq, Maf, Inf) as simple, average average, or complex

- Assign the appropriate number of function points
- The sum of function pointers for each component gives UFP (unadjusted function points)

Function Point Estimation

- Step 2: Compute the technical complexity factor (TCF)
  - Assign a value from 0 (“not present”) to 5 (“strong influence throughout”) to each of 14 factors such as transaction rates, portability
  - Add the 14 numbers: This gives the total degree of influence (DI)
    - *$TCF = 0.65 + 0.01 \times DI$*
    - *The technical complexity factor (TCF) lies between 0.65 and 1.35*
- Step 3 The number of function points (FP) is:
  - *$FP = UFP \times TCF$*

Function Point Estimation (example)



Function Point Estimation (example)

Mapping Functions to Transaction Types

Functions	Transaction Type
Add Customer Change Customer Delete Customer Receive payment Deposit Item Retrieve Item Add Place Change Place Data Delete Place	External Inputs
Print Customer item list Print Bill Print Item List	External Outputs
Query Customer Query Customer's items Query Places Query Stored Items	External Inquiries

Function Point Estimation (example)

Calculate the Unadjusted Function Points

Function Type	Number		Weight Factors: simple	average	complex		
External Input (EI)		x	3	4	6	=	
External Output (EO)		x	4	5	7	=	
External Queries (EQ)		x	3	4	6	=	
Internal Datasets (ILF)		x	7	10	15	=	
Interfaces (EIF)		x	5	7	10	=	

Unadjusted Function Points (UFP) =

The unadjusted function points are adjusted with general system complexity (GSC) factors

Function Point Estimation (example)

After the GSC factors are determined, compute the

Value Added Factor (VAF): $\mathbf{VAF} = 0.65 + 0.01 * \sum_{i=1}^{14} \mathbf{GSC_i}$ $\qquad \mathbf{GSC_i} = 0,1,\ldots,5$

Function Points (FP) = Unadjusted Function Points (UFP)* VAF

PF (Performance factor) = Number of function points that can be completed per day

Effort = FP / PF

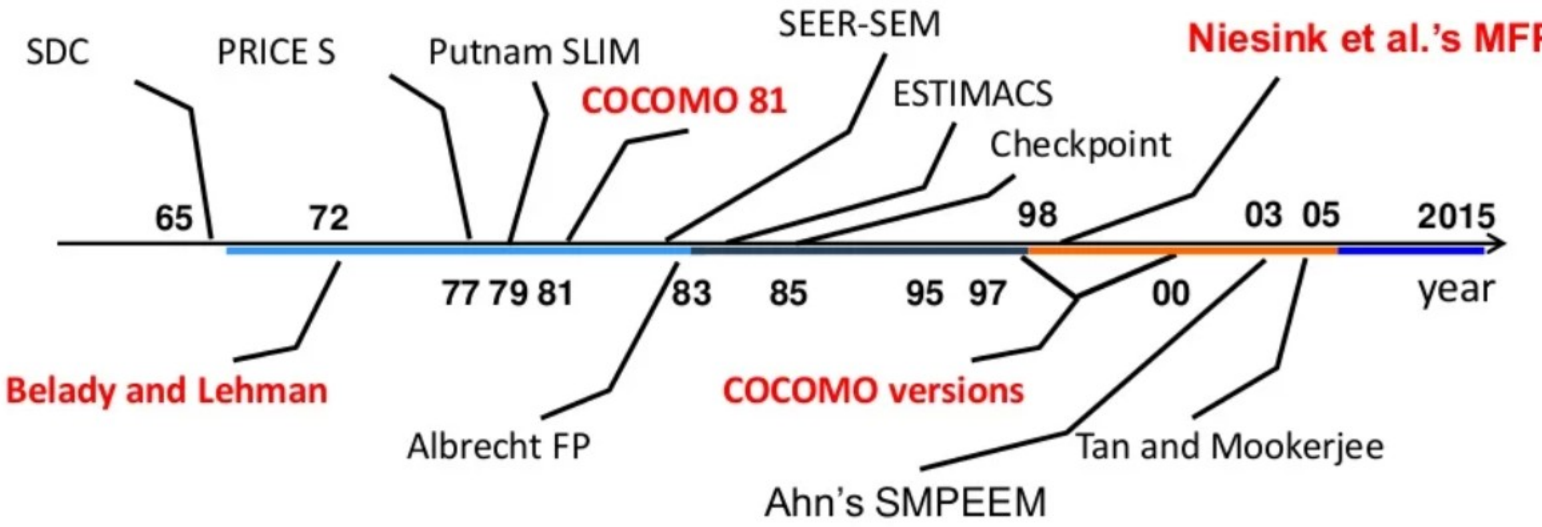
The unadjusted function points are adjusted with general system complexity (GSC) factors

Function Point Estimation (challenges)

- Complete description of functions necessary
  - Often not the case in early project stages -> especially in iterative software processes
- Internal functions (algorithms) rather underestimated, as model is based on user oriented requirements and functions
- Only complexity of specification is estimated
  - Implementation is often more relevant for estimation
- High uncertainty in calculating function points:
  - Weight factors are usually deducted from past experiences (environment, used technology and tools may be out-of-date in the current project)
- Not suitable for project controlling.

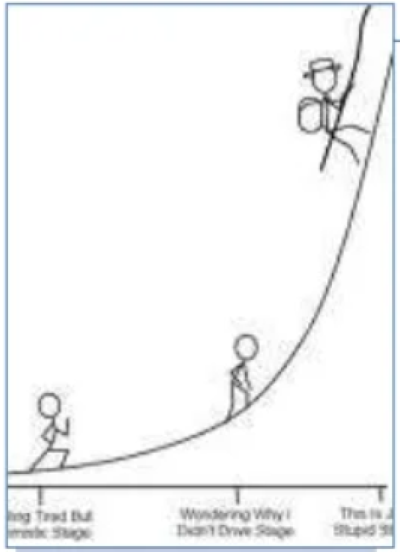
Algorithmic cost modelling

- Most models before 2005 are basing on some form of parametric formulas
- After 2005, more advanced prediction and classification approaches



- 65-85: the search for right parametric forms
- 85-95: advances in size and complexity metrics
- 95-05: proliferation of software development styles
- 05-15: advances in prediction and classification models

Belady and Lehman (1972)



$$Effort = p + K^{c-d}$$

- *Effort*: total maintenance effort
- *p*: productive effort, including analysis, design, code testing
- *d*: degree of maintenance team familiarity with the software
- *c*: complexity caused by lack of structured design and document
- *K*: empirical constant, depends on the environment

The COCOMO model

- Effort (person month) is a function of size (LOC)
- Exists in three stages
    - Basic - Gives a 'ball-park' estimate based on product attributes Intermediate
    - Modifies basic estimate using project and process attributes
    - Advanced - Estimates project phases and parts separately

Effort = b x Size ^{c}

- Organic: small teams develops software in known environment (*b*=2.4, *c*=1.05)
- Embedded: inflexible and constrained environment (*b*=3.6, *c*=1.20)
- Semidetached: varying levels of team experience working on larger projects (*b*=3.0, *c*=1.12)

Effort = b x Size ^{c} x EAF

- Intermediate model
- b, c: calibrated factors
- EAF: effort adjustment factor

COCOMO suite of models (1981-2007)

Advantages

- Repeatable estimations
- Easy to modify input
- Easy to customize and refine formula

Disadvantages

- Subjective inputs
- Unable to deal with exceptional conditions
- Mainly designed for waterfall
- Needs historical data for calibration

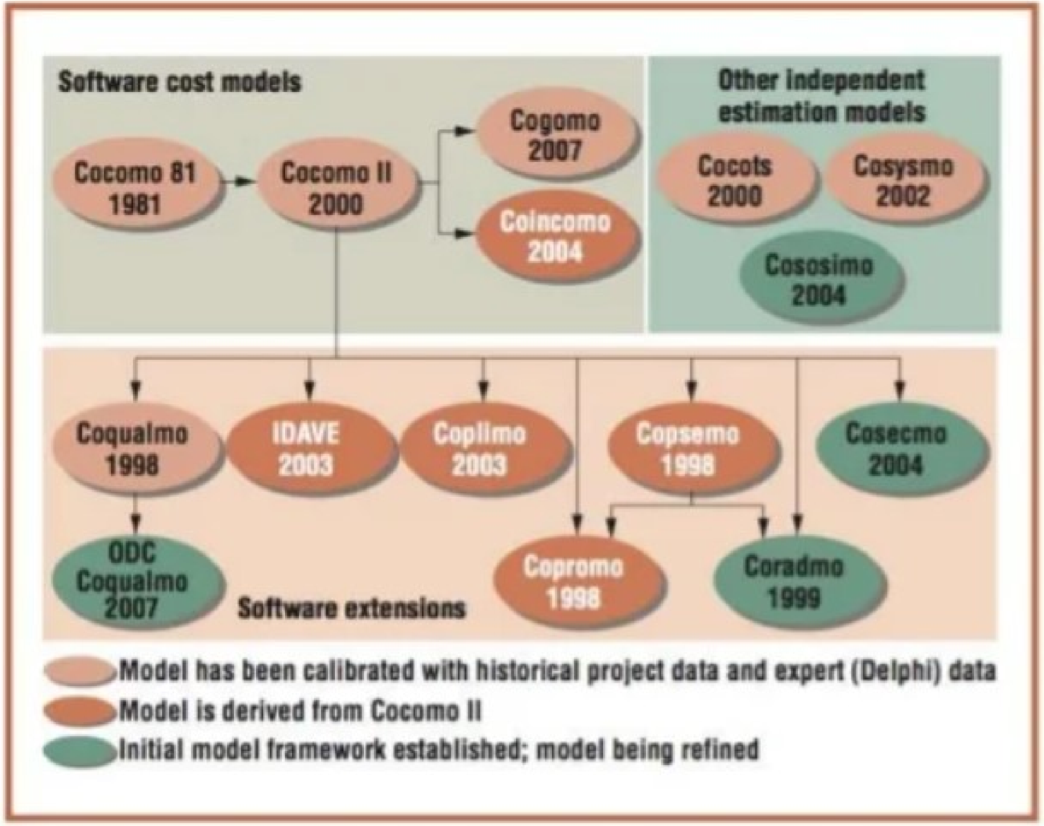


Figure from Boehm et al. 2008

Machine Learning-based Estimating Effort

Types of ML techniques

Case-Based Reasoning (CBR)

Decision Trees (DT)

Bayesian Networks (BN)

Support Vector Regression (SVR)

Genetic Algorithms (GA)

Genetic Programming (GP)

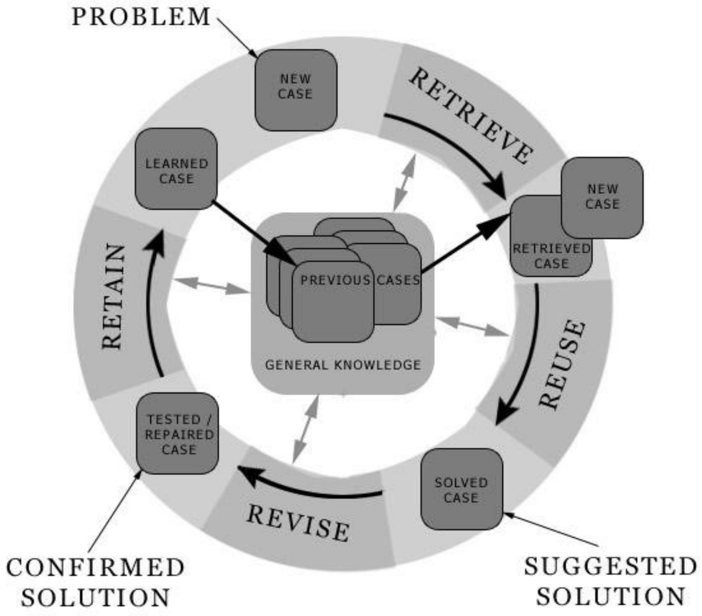
Association Rules (AR)

Artificial Neural Networks (ANN)

J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, “Systematic literature review of machine learning based software development effort estimation models,” *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, Jan. 2012

Case-Based Reasoning (CBR)

- Finding a similar past case, and reusing it in the new problem situation
- CBR is the most widely used method in the software estimation practice



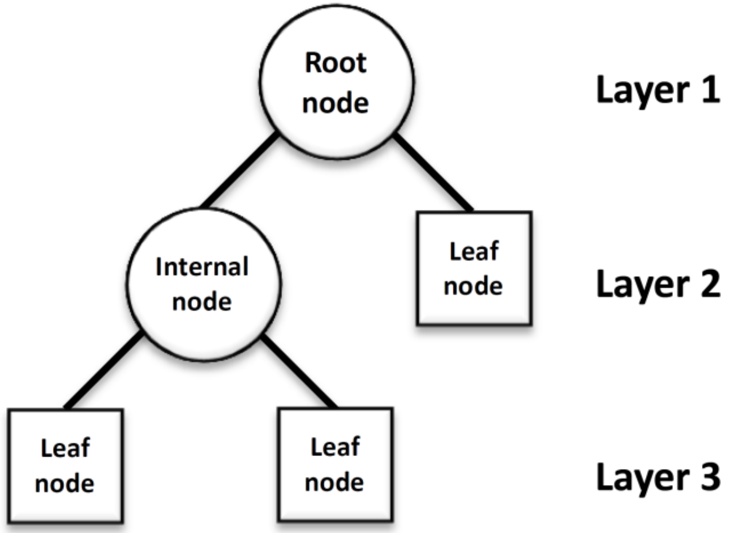
Similarity measure:
- Manhattan distance
- Euclidean distance
- Minkowski distance
- grey relational coefficient
- Gaussian distance
- Mahalanobis distance

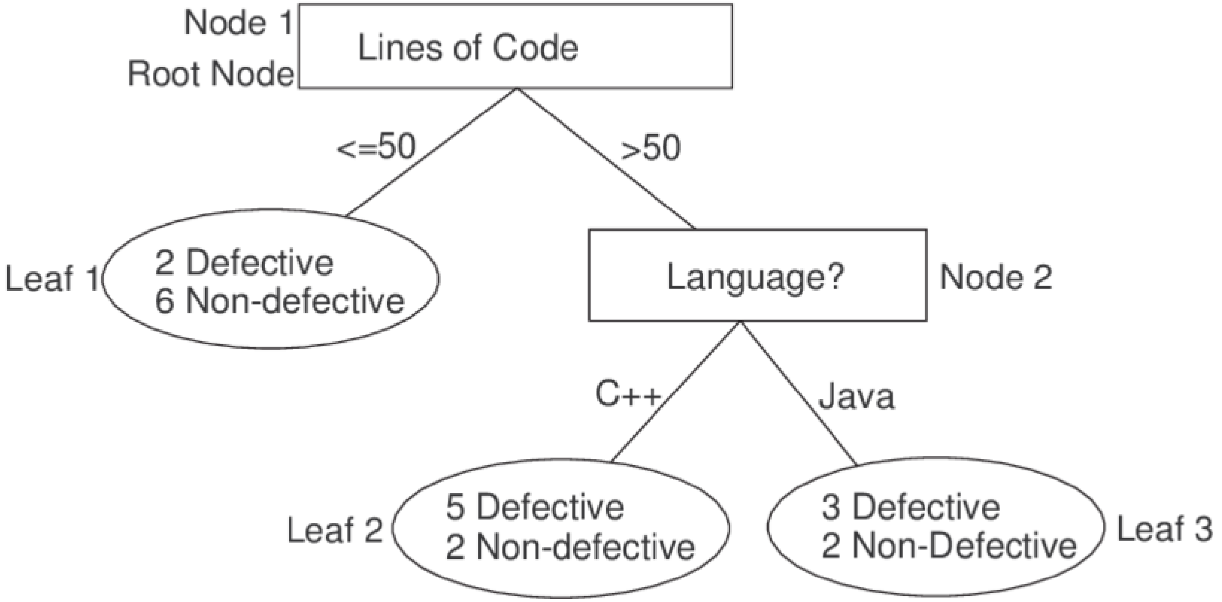
ID	Feature	Full name or explanation
1	TeamExp	Team experience in years
2	ManagerExp	Project manager's experience in years
3	YearEnd	Year of completion
4	Length	The length of project
5	Transactions	Number of transaction processed
6	Entities	Number of entities
7	PointsAdjust	Adjusted function points
8	PointsNonAjust	Unadjusted function points
9	Envergure	Complex measure derived from other factors
10	Effort	Measured in person-hours

Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33:33–53
E. Rashid, S. Patnaik, and V. Bhattacherjee, "Software Quality Estimation using Machine Learning: Case-based Reasoning Technique," *International Journal of Computer Applications*, vol. 58, no. 14, pp. 43–48, Nov. 2012.

Decision Trees (DT) /
Classification and regression trees (CART)

- relatively easy to understand and are also very effective
- DT splits the original dataset to sub datasets which are more homogeneous.
- Recursive Binary Splitting: *all the features are considered and different split points are tried*



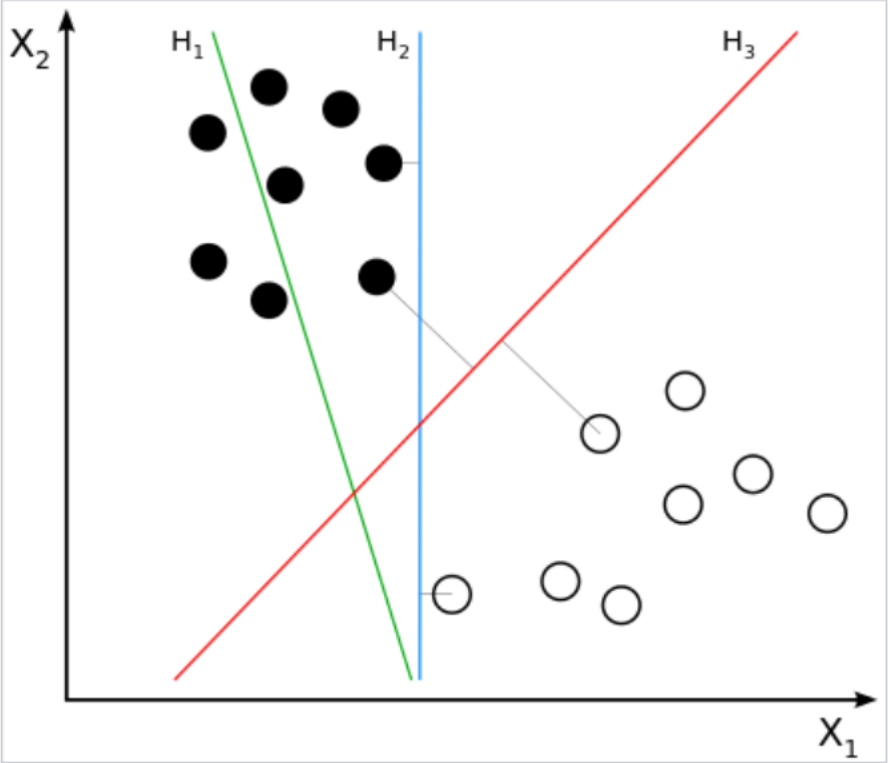


When to stop splitting?

Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33:33–53
E. Rashid, S. Patnaik, and V. Bhattacherjee, "Software Quality Estimation using Machine Learning: Case-based Reasoning Technique," *International Journal of Computer Applications*, vol. 58, no. 14, pp. 43–48, Nov. 2012.

Support Vector Machine (SVM)

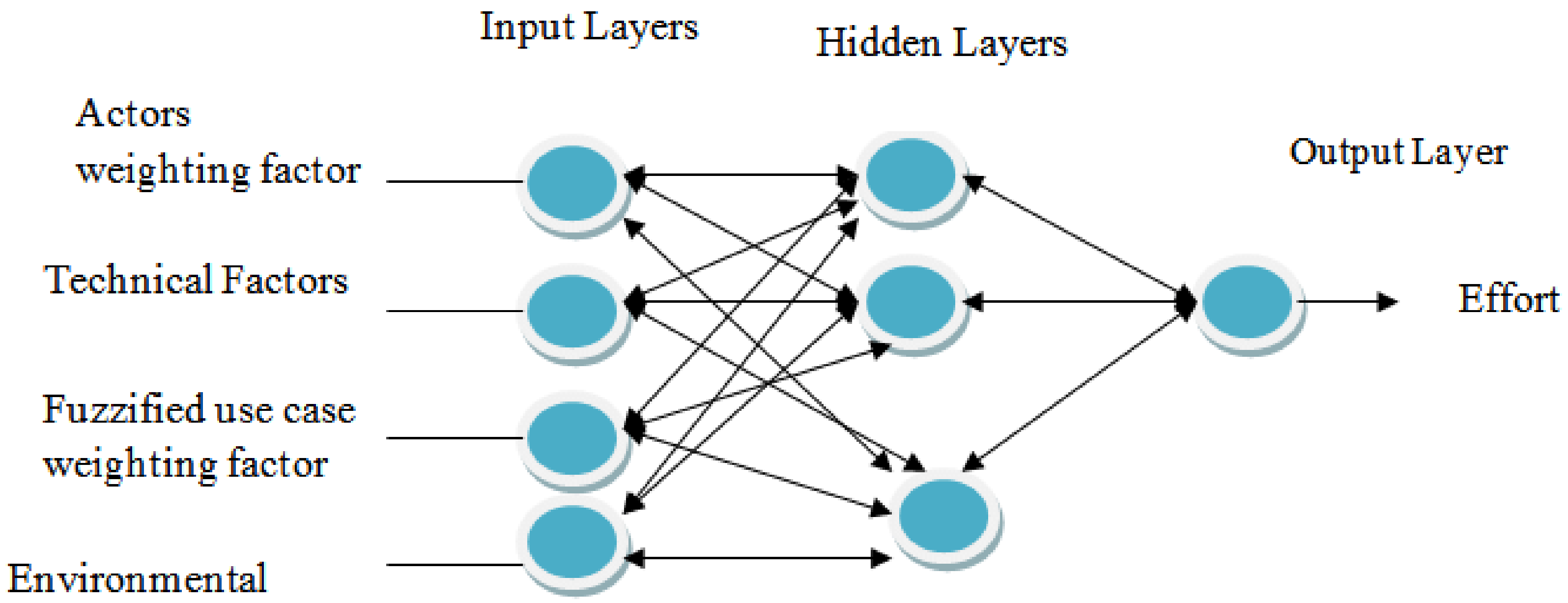
- Builds a model that assigns new case to one category or the other (binary classifier)
- Map training cases to points in space to maximise the width of the gap between the two categories.
- A new case is then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.



H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximal margin.

Artificial Neural Network (ANN)

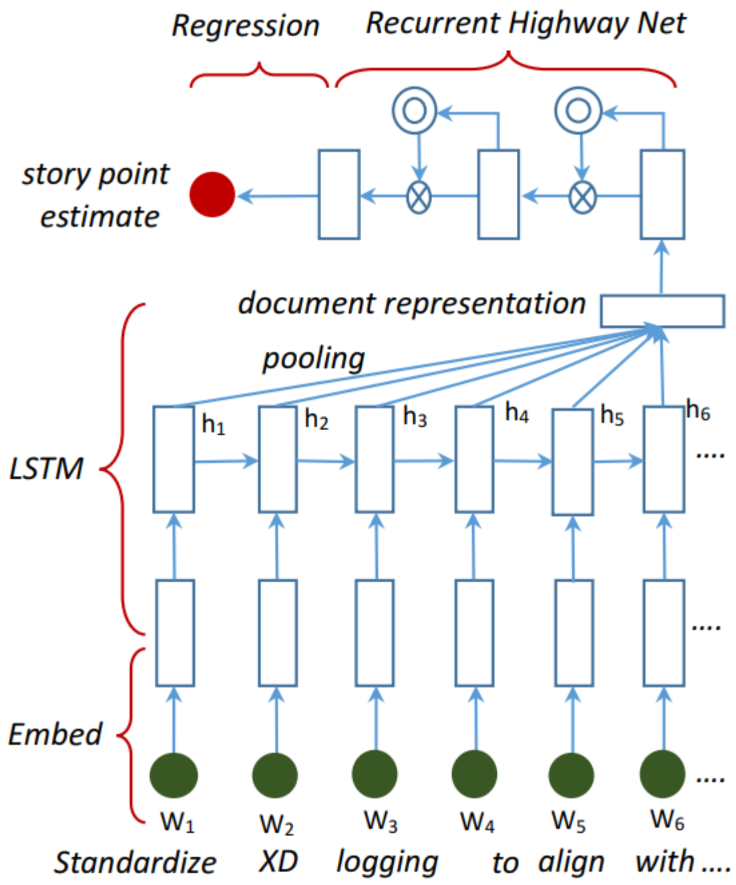
- Input layers: size of the software in function points or Lines Of Code (LOC), and other effort drivers such as complexity of the software, database size, experience etc.
- Each node has its own linear regression model, compose of input and output variables



K. K. Gebretsadik and W. T. Sewunetie, “Designing Machine Learning Method for Software Project Effort Prediction,” *Computer Science and Engineering*, vol. 9, no. 1, pp. 6–11, 2019.

Deep Learning

- Model for Agile projects, predict efforts for the next Iteration
- Adopt Long Short Term Memory / Recurrent Neural Network
- Input: SP: story points, TD length: the number of words in the title and description of an issue, LOC: line of code

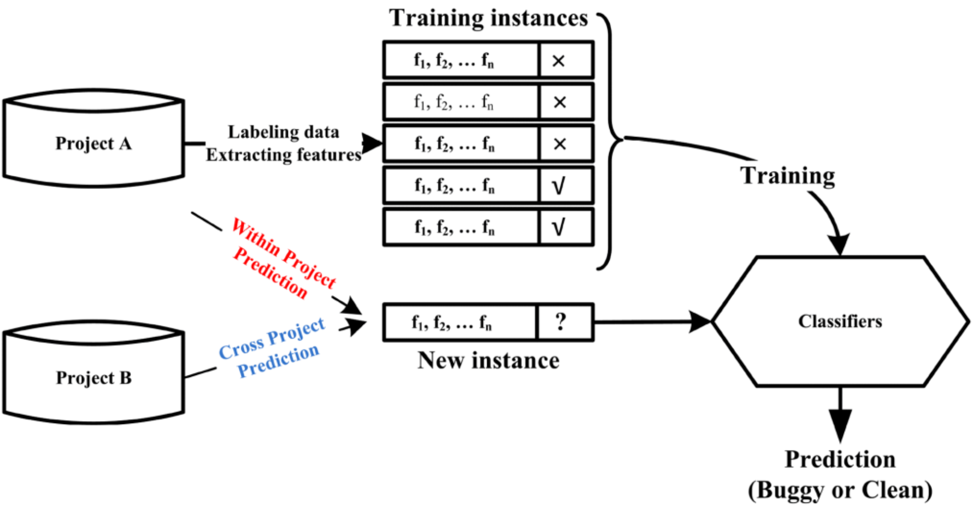


M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, Jul. 2019

Defect Prediction

Importance of defect prediction

- Software defect: flaw or imperfection in software work or software process
- A defect is also referred as a fault or a bug
- Every software project has bugs, which we hope to reduce.
- Defect prediction is essential in the field of software quality and software reliability
- Focus on predicting those defects that affect project and product performance



Urgent Severe	Urgent Not Severe
Not Urgent Severe	Not Urgent Not Severe

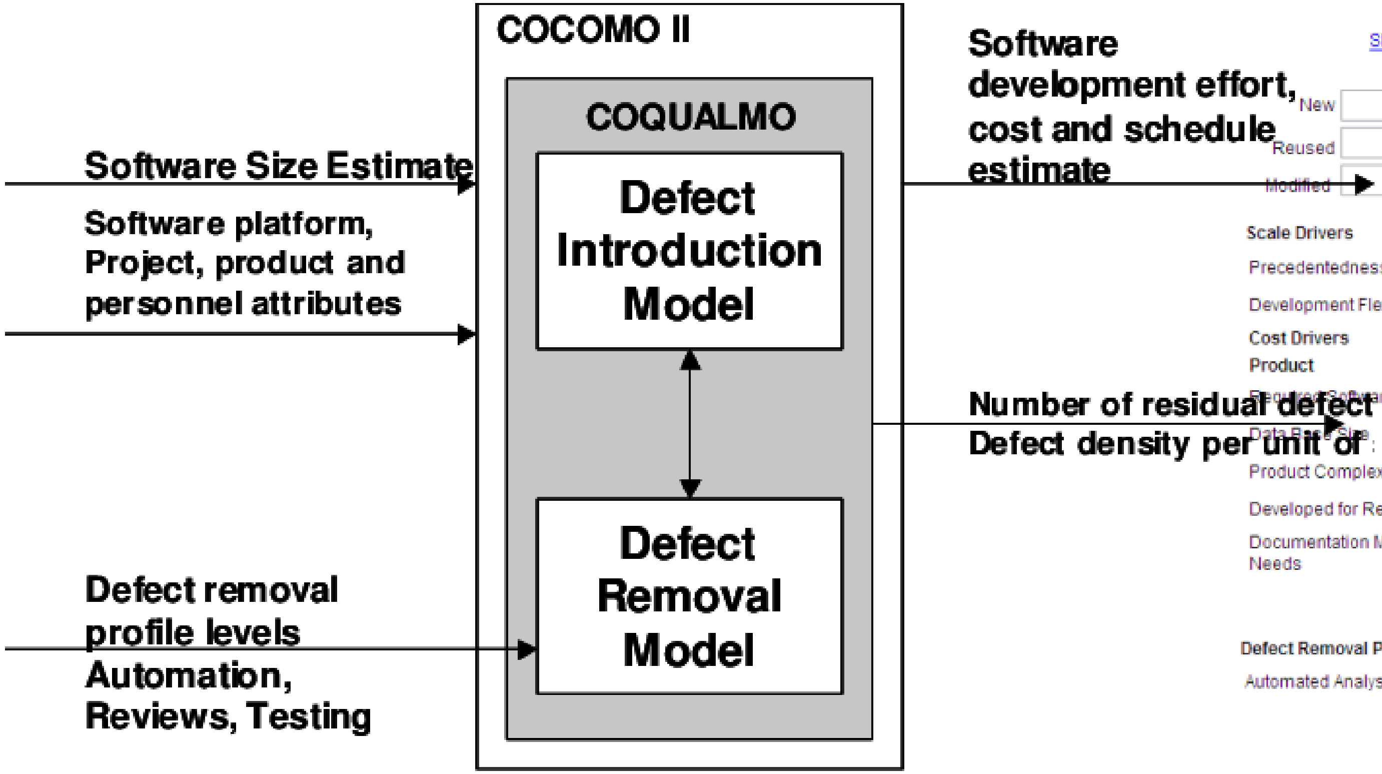
Pareto rule (20-80 rule)

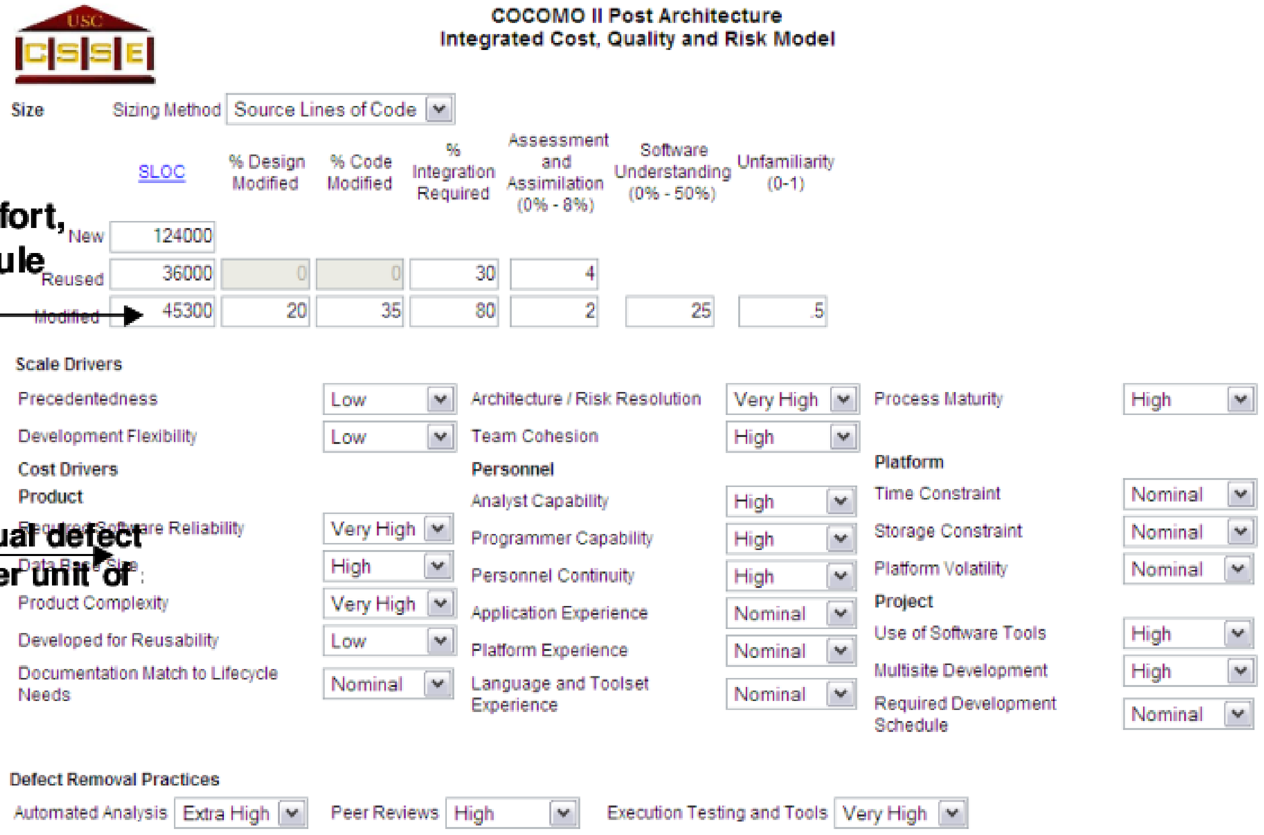
Most defects (80%) exist in few modules (20%)

Defect data is usually imbalanced

Early days prediction models - COQUALMO

- predicting number of residual defects/KSLOC (Thousands of Source Lines of Code) or defects/FP (Function Point) in a software product
- Base on COCOMO II model





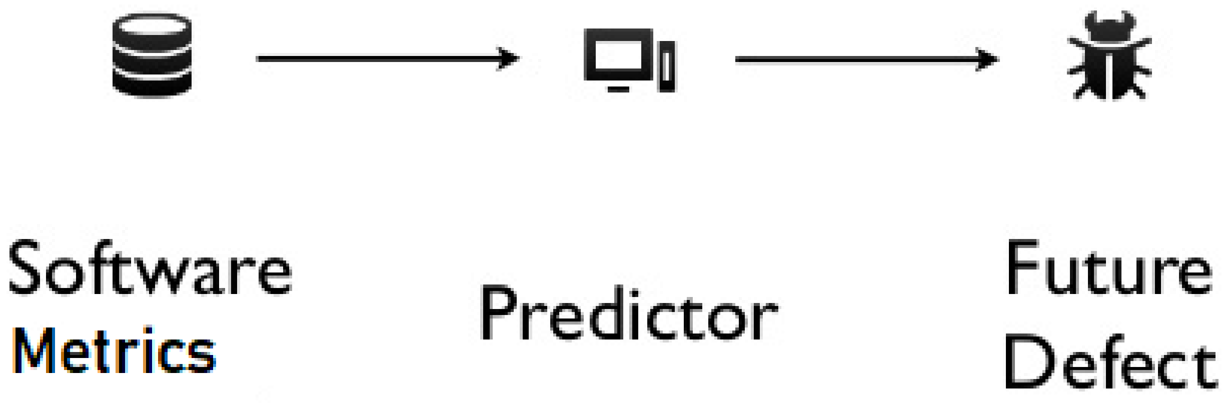
Algorithmic approaches - Basic principles

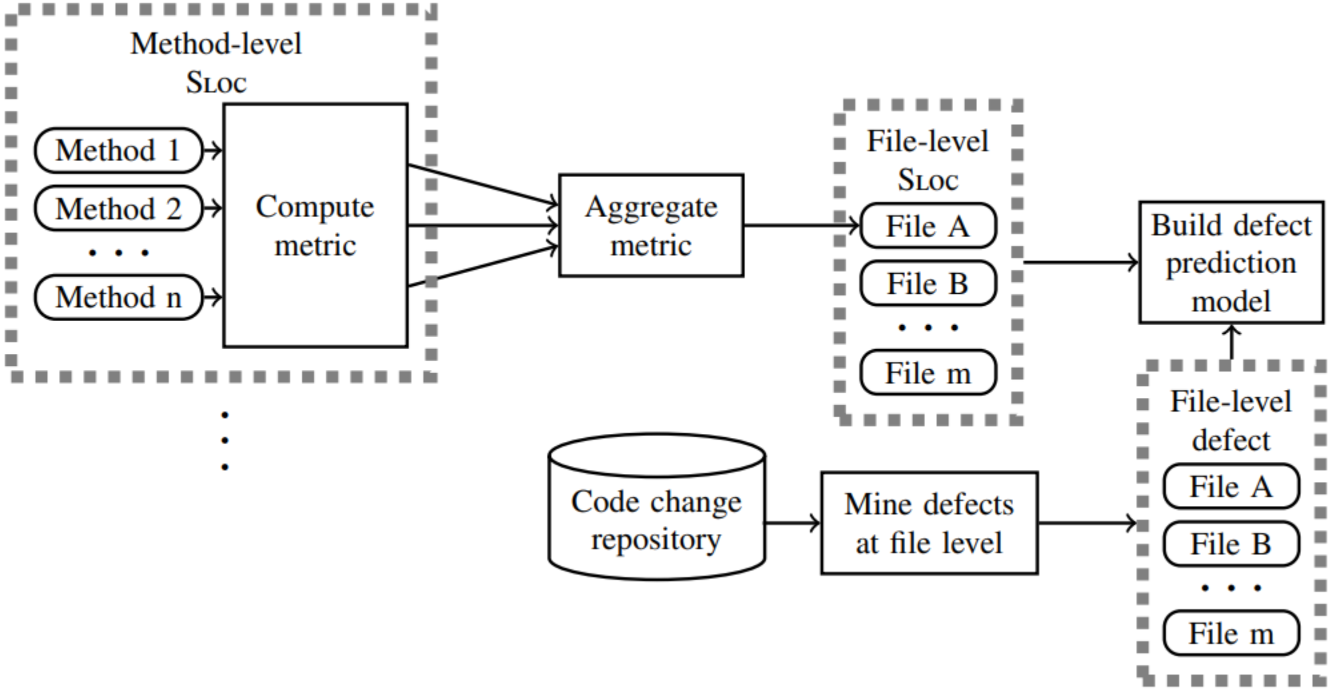
Software defect prediction – mainly base on historical data

When to use: (1) use for planning (2) in-process monitoring of defect discovery numbers

What to predict?

- Number of defect
- Defect density
- Defect proneness

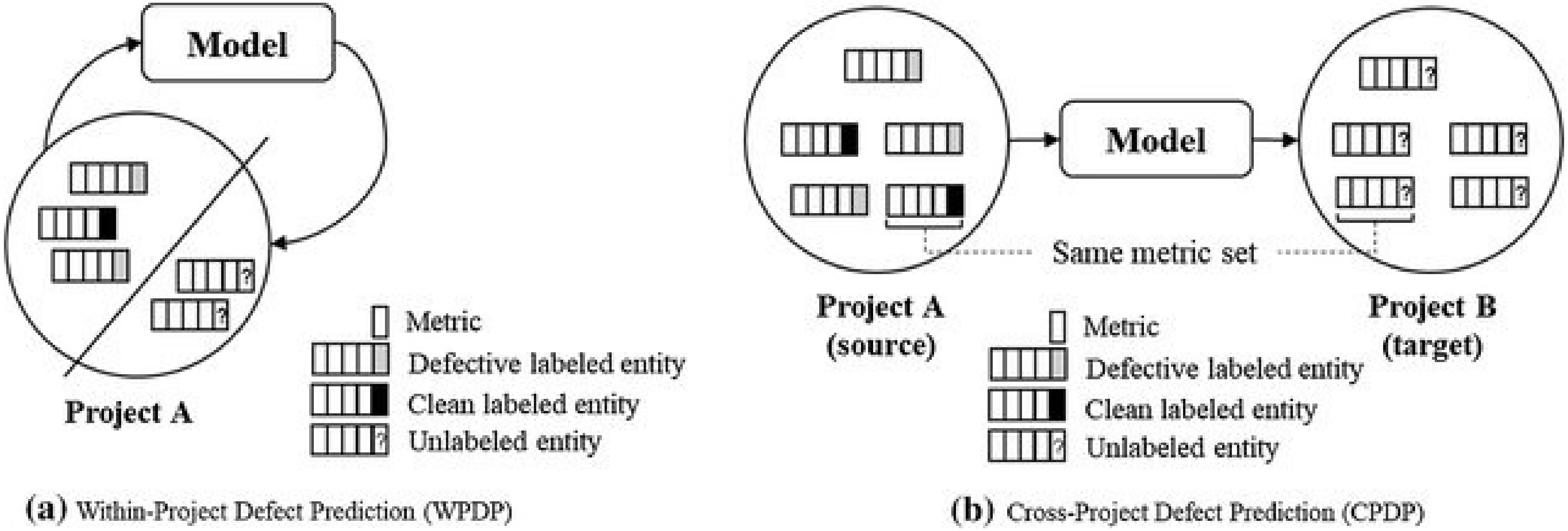




Software entity	Software attribute	Software metrics	References	# of Refs
Product	Size	LOC or number of use cases	S2, S37, S15, S21, S49, S32, S47, S35, S27, S13, S42, S34, S10, S19, S41, S16, S36,	17
		Size of artifact	S1, S13	2
		Code metrics: size metrics from NASA projects (Halstead size metrics)	S50, S33, S21, S28, S11, S12, S3, S9, S4, S25	10
		Requirement metrics: action, conditional, continuance, imperative, incomplete, option, risk level, source, weak phrase	S33, S28, S11, S12, S3, S9, S4, S20, S25	9
	Structure	McCabe Metrics (Complexity etc.)	S52, S21, S17, S47, S42, S44, S30, S39, S22,	12
		Halstead Metrics (total number of operators, operands etc.)	S7, S48, S19	
		Object-oriented Metrics (Complexity, Length, Coupling, Cohesion, Modularity, Reuse)	S51, S50, S29, S18, S14, S6, S16, S36, S19	9
		Design metrics from UML (e.g. Chidamber and Kemerer (1994))		
		Data flow complexity (DC), cyclomatic complexity (CC)	S8, S24, S43	3
		Requirements complexity (RC), Complexity of new functionality	S2, S37, S32, S35, S34, S48	6
		Program dependencies	S38	1
		Design metrics: edge count, node count, branch count, decision count, multiple condition count and condition count, densities, complexities	S20	1
		Architectural design metrics to quantify SDL (Specification and Description Language) blocks	S45	1
		Design, review or development effort measured in person hour	S1, S5, S23, S37, S31, S40, S35, S13, S43	9
		Creation effort, review effort	S26, S46	2
		Design review effectiveness (DRE)	S30, S7	2
		Review, inspection and walkthrough (RIW)	S2, S8, S32, S24, S34, S30, S7, S10, S48	9
		Total months of the project duration	S15, S31, S41	3
		Requirements stability (RS), requirement change request (RCR)	S2, S37, S17, S8, S32, S35, S27, S24, S34, S43, S30, S7, S10, S48	14
		Capability Maturity Model Integration (CMMI) Level	S2, S37, S8, S32, S40, S35, S24, S34, S30, S7,	10
		Number of detected defects from review	S1, S37, S15, S35, S13	5
		Requirement fault density (RFD), design defect density (DDD), fault days number (FDN), code defect density (CDD),	S15, S17, S8, S27, S24, S10, S48,	7
		Analysis, design, review quality	S37, S40, S35, S43	4
		Quality of documented test cases (QDT)	S35, S7, S41	3
		Defined process followed (DPF)	S2, S32, S34, S35, S37	5
		Number of stakeholders/members	S15, S49, S41	3
		Development language	S37, S15	2
		Configuration management	S37, S35, S41	3
		Project planning	S37, S35	2
		Scale of distributed communication	S37, S35	2
		Vendor management	S37, S35	2
		DBMS type, development solution, industry area	S15, S41	2
		Techno complexity	S26, S49, S46	3
		Urgency	S46	1
		Novelty to developer	S49	1
	Human characteristics	Staff experience	S2, S37, S32, S40, S35, S27, S34, S43, S7, S10	10
		Staff motivation	S37, S35	2
		Programmer capability	S37, S35, S30, S7	4
		Staff training quality	S37, S35	2
		Internal communication/interaction	S37, S35	2
		Productivity	S15	1
		Practitioners level	S26, S46	2
		Stakeholder involvement	S2, S32, S34	3
		People dependence	S41	1

R. Özakıncı and A. Tarhan, “Early software defect prediction: A systematic map and review,” *Journal of Systems and Software*, vol. 144, pp. 216–239, Oct. 2018, doi: 10.1016/j.jss.2018.06.025.

With-in vs. Cross-project prediction

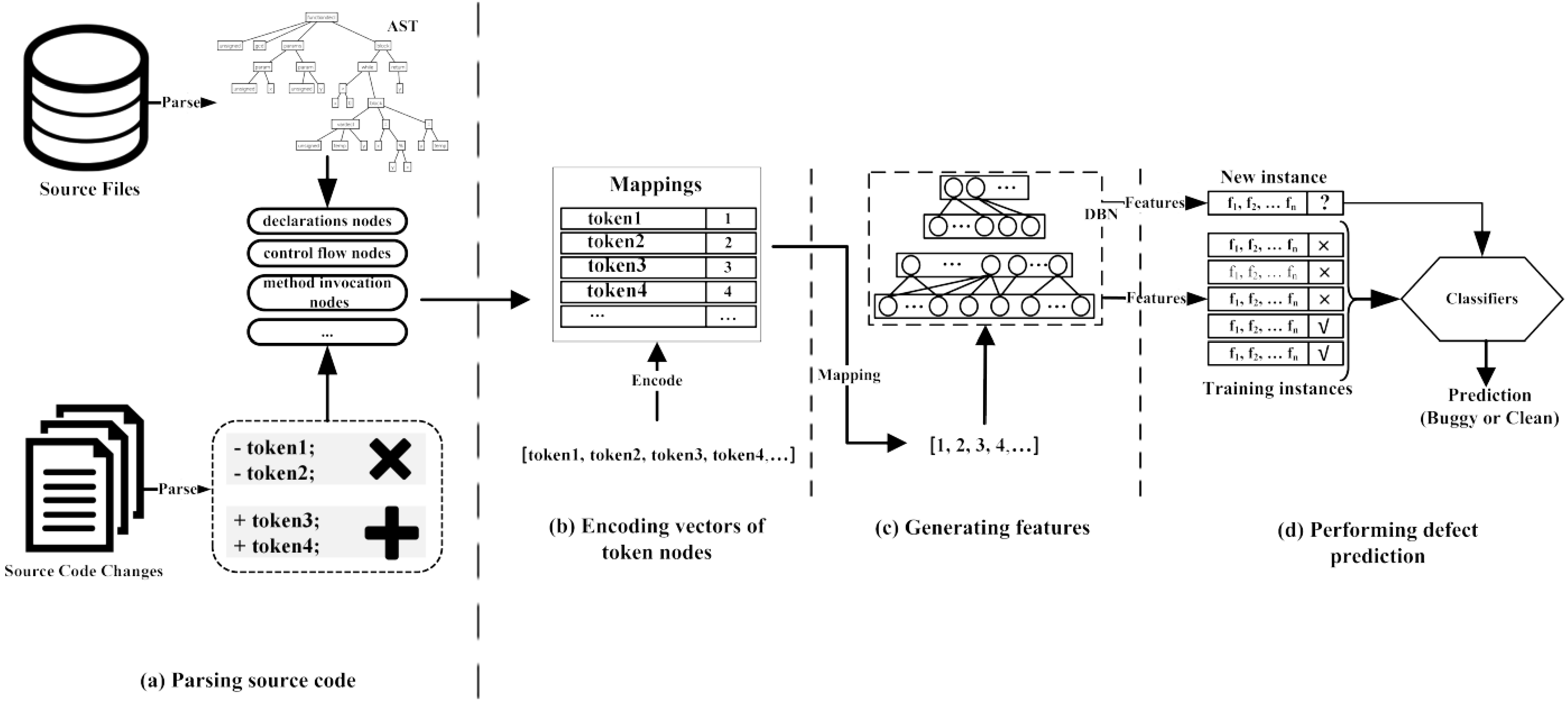


due to the diversity in development processes, a defect prediction model is often not transferable and requires to be rebuilt when the target project changes

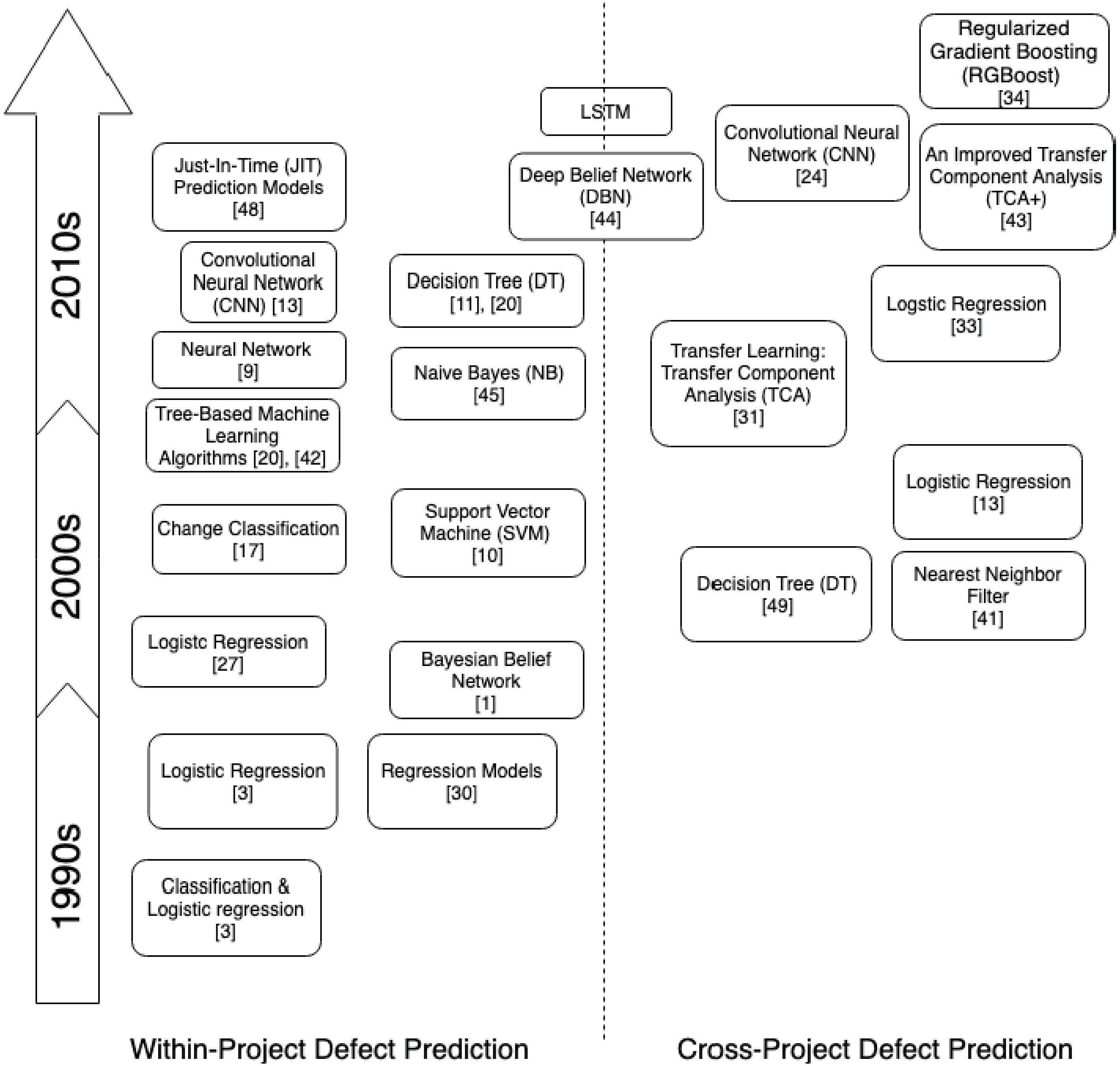
Critiques on defect prediction

- the unknown relationship between defects and failures
- problems with the “multivariate” statistical approach
- problems of using size and complexity metrics as sole “predictors” of defects
- problems in statistical methodology and data quality
- false claims about software decomposition and the “Goldilock’s Conjecture”

A typical ML model for defect prediction



History of ML/AI approaches in defect prediction



S. Omri and C. Sinz, “Deep Learning for Software Defect Prediction: A Survey,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, New York, NY, USA: Association for Computing Machinery, 2020, pp. 209–214. Accessed: Mar. 19, 2022. [Online]

Just-in-time defect prediction

Fixing Commit Message

Revert "Make VisibleRefFilter.Filter reuse the refs passed from JGit."

This reverts commit b032a529f83892dfbdfb375c47a90d89756dd8ab. This commit introduced an issue where tags were not replicated under certain circumstances.

Bug: Issue 2500
Bug: Issue 1748
Change-Id: I9c902b99c7f656c7002cf3eab9e525f22a22fb85

Defective Commit: b032a529f83892dfbdfb375c47a90d89756dd8ab

2 gerrit-server/src/main/java/com/google/gerrit/server/git/VisibleRefFilter.java

```diff
    if (!deferredTags.isEmpty() && (!result.isEmpty() || filterTagsSeperately)) {
      TagMatcher tags = tagCache.get(projectName).matcher(
          tagCache,
          db,
-         filterTagsSeperately ? filter(db.getAllRefs()).values() : result.values());
+         filterTagsSeperately ? filter(refs).values() : result.values());
      for (Ref tag : deferredTags) {
        if (tags.isReachable(tag)) {
          result.put(tag.getName(), tag);
        }
      }
    }
```

Fixing Commit: 6db280663f836096c30a9626e7170f4a36d8cc1f

2 gerrit-server/src/main/java/com/google/gerrit/server/git/VisibleRefFilter.java

```diff
    if (!deferredTags.isEmpty() && (!result.isEmpty() || filterTagsSeperately)) {
      TagMatcher tags = tagCache.get(projectName).matcher(
          tagCache,
          db,
-         filterTagsSeperately ? filter(refs).values() : result.values());
+         filterTagsSeperately ? filter(db.getAllRefs()).values() : result.values());
      for (Ref tag : deferredTags) {
        if (tags.isReachable(tag)) {
          result.put(tag.getName(), tag);
        }
      }
    }
```