Predicting Faults Using the Complexity of Code Change

International Conference on Software Engineering

(2009)

Ahmed E. Hassan

2013-07-09

Yoo Jin Lim

KAIST SE LAB

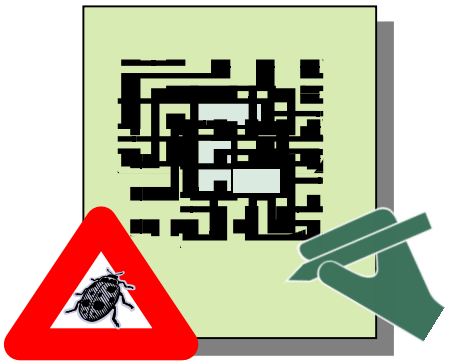
KAIST Software Engineering Laboratory

Contents

- Introduction
- Background
- Code Change Models
- Case Study
- Related Work
- Conclusion
- Discussion
- Appendix

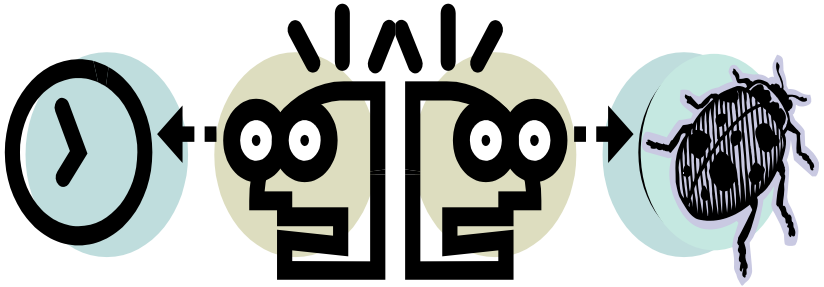
Introduction (1/2)

- Fault Prediction
  - Predict the incidence of faults in code
    - Early warning for developers and managers
    - Affects test efforts, costs, and product quality
  - Commonly associated with code complexity
    - Ex) More LOC → More complex code
  - **Process** metrics outperform **code** metrics
    - Then, consider code change process complexity



Introduction (2/2)

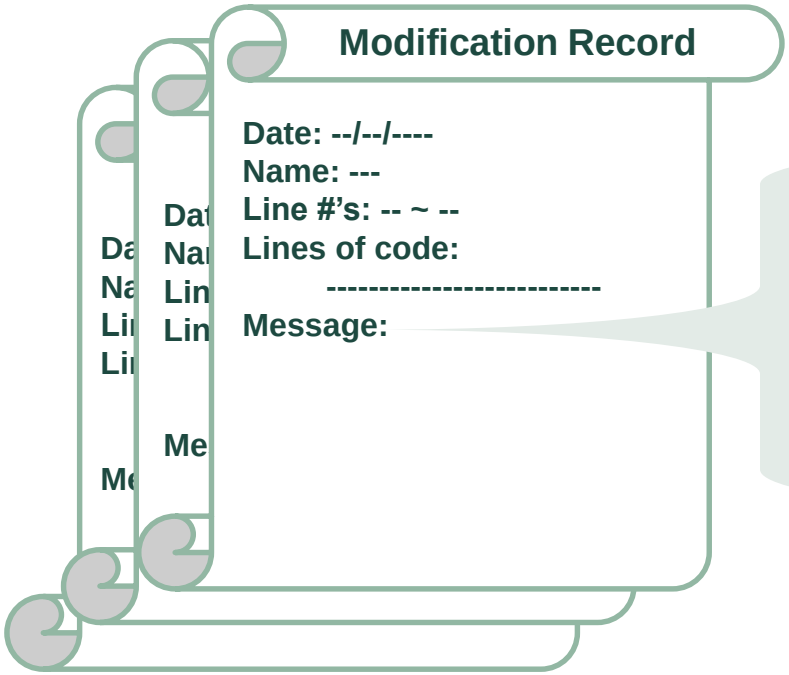
- Motivation
  - To prevent fault occurrences, predict faults.
  - To manage negative effects of complexity, measure it.



- Goal
  - Suggest a predictor of future faults based on code change complexity models

Background (1/3)

- Code change process
  - Pattern of source code modifications
  - Recorded by source control systems
  - Repository has all files' change history



- Fault Repairing modifications (FR)
- General Maintenance modifications (GM)
- Feature Introduction modifications (FI)

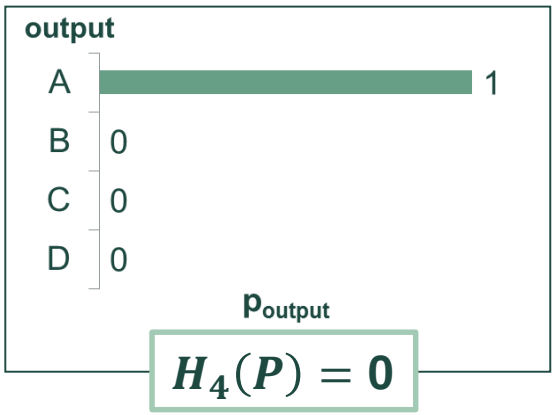
Background (2/3)

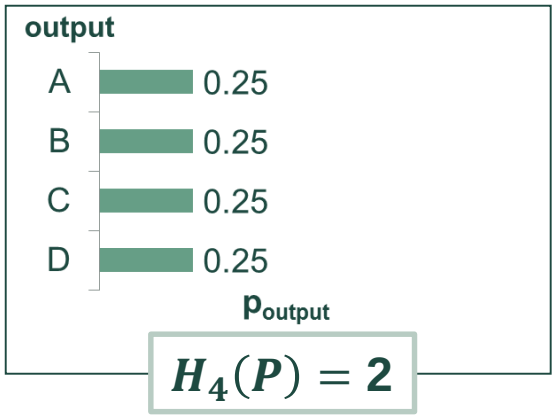
- Shannon Entropy
  - Measure **entropy (amount of uncertainty)** in a distribution

$$H_n(P) = -\sum_{k=1}^{n}(p_k * \log_2 p_k), \qquad p_k \geq 0 \ \ and \ \sum_{k=1}^{n} p_k = 1$$

$p_k = probability\ of\ occurence\ for\ element\ k$

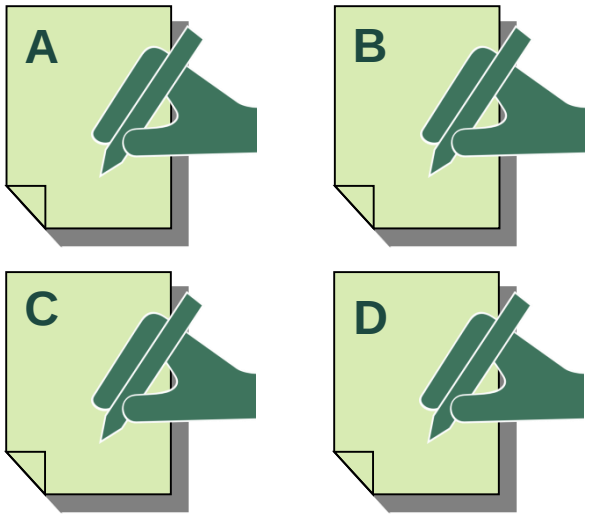
  - Minimal vs. Maximum Entropy
    - Ex) Output distribution of a system with 4 symbols





Background (3/3)

- Quantifying the code change complexity
  - Highly scattered changes
  - Highly complex project

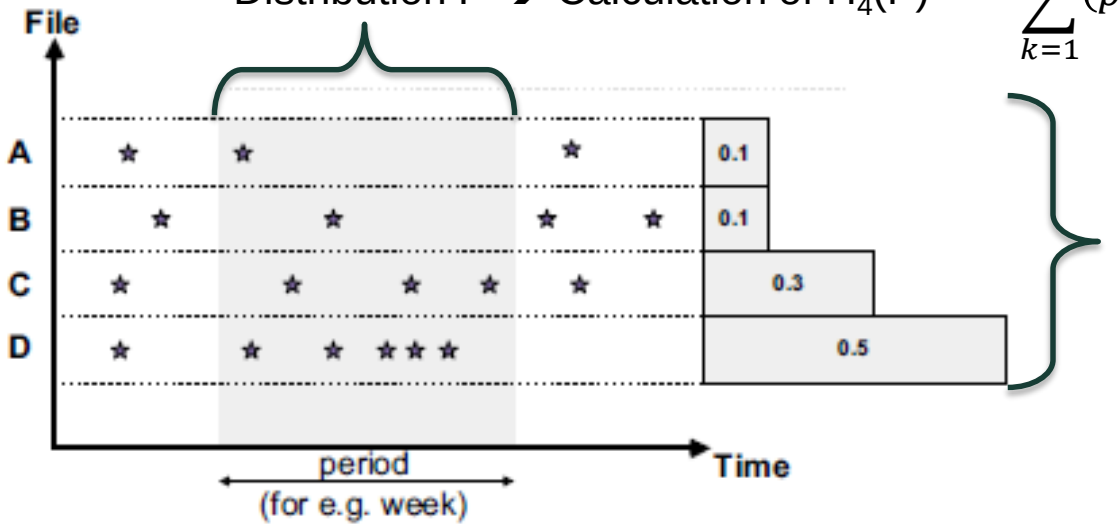


Change patterns with **high entropy** are **harder** to track!

Basic Code Change Model(1/2)

- Complexity of a change period

Distribution P ➔ Calculation of $H_4(P) = -\sum_{k=1}^{4}(p_k * \log_2 p_k) = \mathbf{1.68}$

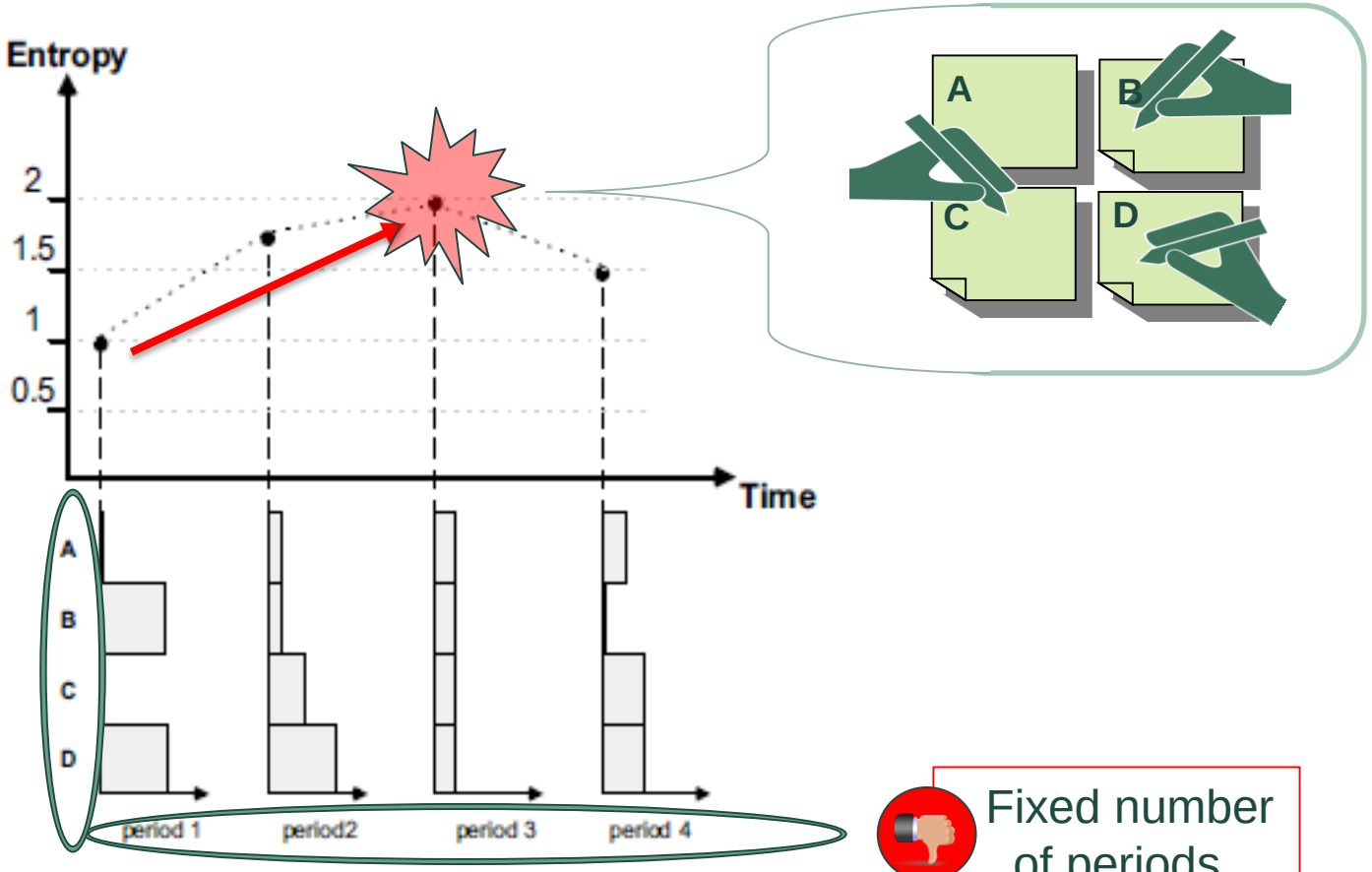


$$p_{file} = \frac{Change\ count\ of\ a\ file}{Total\ change\ count}$$

- Measurement approach
  - Use the **file** as unit of code
  - Only use the **FI** modifications
  - Quantify for **several** changes within a **period**

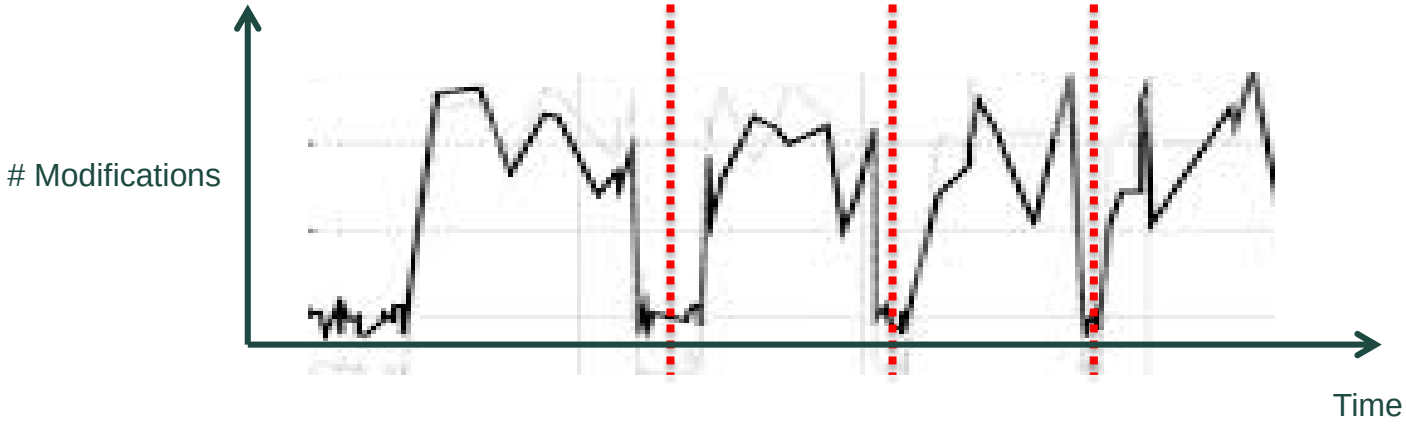
Basic Code Change Model(2/2)

- Evolution of code change entropy



Extended Code Change Model(1/2)

- Evolution Periods
  - Time based periods
  - Modification limit based periods
  - Burst based periods
    - Burst = many code modifications followed by none
    -  The most general method



Extended Code Change Model(2/2)

- Adaptive System Sizing
  - Normalized Static Entropy, H = [0,1]

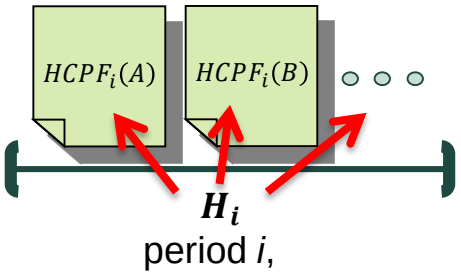
$$H(P) = \frac{1}{Max\ Entropy\ for\ Distribution} * H_n(P) = \frac{1}{\log_2 n} * H_n(P)$$

    - n = number of all files
    - → entropy of distributions of different sizes can be compared

  - Adaptive Sizing Entropy, H'
    - n' = number of *recently* modified files
      - Using time: Set of files modified in the preceding *x* months
      - Using previous periods: those modified in the preceding *x* periods

File Code Change Model(1/4)

- Complexity of a **file**
  - History Complexity Period Factor **(HCPF)**
    - For a file ***j*** during period ***i***, a set of files F_i is modified:



$$HCPF_i(j) = \begin{cases} c_{ij} * H_i, & j \in F_i \\ 0, & otherwise \end{cases}$$

c_{ij} = contribution of entropy H_i to file j

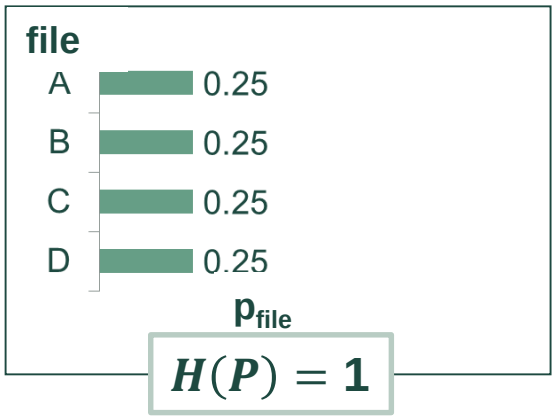
- HCPF¹ with $c_{ij} = 1$: All modified files are affected by full entropy.
- HCPF² with $c_{ij} = p_j$: The more a file is modified, more it is affected.
- HCPF³ with $c_{ij} = \frac{1}{|F_i|}$: The more modified files, less each is affected.

File Code Change Model(2/4)

- Example of **HCPF**

$$HCPF_i(j) = \begin{cases} c_{ij} * H_i, & j \in F_i \\ 0, & otherwise \end{cases}$$

  - Calculate **HCPF** for file ***A*** during this period



HCPF^1(A)	$1 * 1 = \mathbf{1}$
HCPF^2(A)	$0.25 * 1 = \mathbf{0.25}$
HCPF^3(A)	$\frac{1}{4} * 1 = \mathbf{0.25}$

File Code Change Model(3/4)

- History Complexity Metric **(HCM)**
  - Simple HCM for a file
    - For a file ***j*** over a set of evolution periods **{*a,…,b*}**:

$$HCM_{\{a,\ldots,b\}}(j) = \sum_{i \in \{a,\ldots,b\}} HCPF_i(j)$$

$HCPF^1$
$HCPF^2$
$HCPF^3$

  - HCM for a subsystem
    - For **all** files in subsystem ***S***

$$HCM_{\{a,\ldots,b\}}(S) = \sum_{j \in S} HCM_{\{a,\ldots,b\}}(j)$$

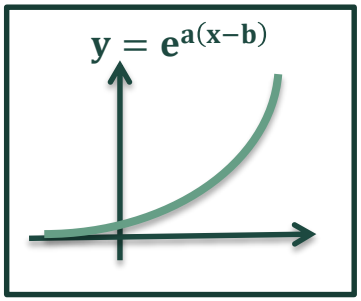
HCM^{1s}
HCM^{2s}
HCM^{3s}

File Code Change Model(4/4)

- History Complexity Metric **(HCM)**
  - Decay model of HCM
    - Earlier modifications have less contribution to complexity

$$HCM_{\{a,\ldots,b\}}(j) = \sum_{i \in \{a,\ldots,b\}} e^{\varphi * (T_i - Current\ Time)} * HCPF_i^1(j)$$

    - T_i = end time of period i
    - φ = decay factor



  - Four metrics in total

$$HCM^{1s}, HCM^{2s}, HCM^{3s}, \text{ and } HCM^{1d}$$

Case Study (1/4)

- Summary of the studied systems

App. Name	Type	Start Date	Subsystem Count	Prog. Lang.
NetBSD	OS	Mar 1993	235	C
FreeBSD	OS	Jun 1993	152	C
OpenBSD	OS	Oct 1995	265	C
Postgres	DBMS	Jul 1996	280	C
KDE	Windowing System	Apr 1997	108	C++
KOffice	Productivity Suite	Apr 1998	158	C++

- Study approach
  - Build Statistical Linear Regression (SLR Model)
    - **To predict faults** in subsystems during the **4th and 5th** years
  - Measure & compare the error between models
    - Modifications vs. Faults vs. **Entropy (4)**
  - Determine statistical significance of the difference in error

Case Study (2/4)

Linear Regression Models

- $y = \beta_0 + \beta_1 x$
  - y = number of faults in a subsystem (FR modifications)
  - x = specific metrics for each subsystem

SLR Model	Value of *x*
Model m	Number of modifications
Model f	Number of prior faults
Model HCM1s	HCM_{1s} value
Model HCM2s	HCM_{2s} value
Model HCM3s	HCM_{3s} value
Model HCM1d	HCM_{1d} value

App	R_f^2	R_m^2	R_{1s}^2	R_{2s}^2	R_{3s}^2	R_{1d}^2
NetBSD	0.57	0.55	0.54	0.53	0.61	0.71
FreeBSD	0.65	0.48	0.57	0.58	0.59	0.65
OpenBSD	0.45	0.44	0.54	0.55	0.54	0.57
Postgres	0.57	0.36	0.49	0.51	0.60	0.61
KDE	0.31	0.26	0.28	0.29	0.36	0.57
KOffice	0.30	0.27	0.33	0.33	0.27	0.41

- The SLR Model **HCM^{1d}** has the **best fit** for all.

Case Study (3/4)

Prediction Error for the SLR Models

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

- $\hat{y}_i$ = Number of predicted faults in the subsystem in **4th and 5th** years
- Absolute prediction error: $e_i = |\hat{y}_i - y_i|$
- Total prediction error: $E = \sum_{i=1}^{n} e_i^2$

Statistical Significance of Differences b/t Models

- Paired t-test is used
- P-value < 0.05: we can with high probability reject H_0:

$$H_0: \mu(e_A, i - e_B, i) = 0$$

Case Study (4/4)

- Statistical significance of differences in error
  - Significant results (P-value < 0.05) highlighted

	Modifications vs. Faults	Modifications vs. Entropy		Faults vs. Entropy	
App. Name	$E_m - E_f$ **(%)**	$E_{HCM3s} - E_m$ **(%)**	$E_{HCM1d} - E_m$ **(%)**	$E_{HCM3s} - E_f$ **(%)**	$E_{HCM1d} - E_f$ **(%)**
NetBSD	+11.7 (+04%)	-39.8 (-14%)	-106.5 (-36%)	-28.1 (-10%)	-94.8 (-34%)
FreeBSD	+71.2 (+48%)	-47.4 (-22%)	-72.0 (-33%)	+23.8 (+16%)	-00.8 (-01%)
OpenBSD	+03.7 (+02%)	-40.4 (-18%)	-53.8 (-23%)	-36.6 (-16%)	-50.5 (-22%)
Postgres	+47.2 (+49%)	-52.7 (-37%)	-56.9 (-40%)	-05.5 (-06%)	-09.7 (-10%)
KDE	+26.3 (+07%)	-52.1 (-13%)	-165.2 (-42%)	-25.7 (-07%)	-138.9 (-38%)
KOffice	+26.3 (+04%)	+03.3 (-01%)	-69.9 (-18%)	+19.2 (+05%)	-54.1 (-15%)

- $E_m \geq E_f \geq E_{HCM}$ in most cases
  - Predictors based on faults are better than modifications, and **complexity models are better than faults or modifications**.

Related Work

- Barry et al. (2003)
  - Studied a retail software's code modification records
  - Identified evolution patterns in the software system

- Adb-El-Hafiz (2001)
  - Quantified the complexity of the **source code** to measure entropy.

- → This paper's model uses code modification records to quantify the complexity of the **code change process.**

Conclusion

- Verified the conjectures:
  - Complex code change process negatively affects the software system.
  - The more complex changes to a file, the higher the chance the file will contain faults.

- Contributions
  - Computed the complexity of the **code change process**
    - instead of just that of the source **code**
  - Presented a **better predictor** of future faults
    - Help managers plan ahead and be ready for future

Discussion

- Limitations
  - Did not consider unrepaired faults
  - Generalization limited to large open source systems
  - Results do not show a causality relation
- Future work
  - Study commercial systems
  - Build a richer and detailed metric for complexity

Thank You .

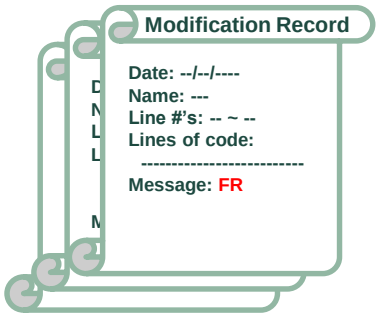
KAIST SE LAB
KAIST Software Engineering Laboratory

Appendix (1/6)

- Code Change Process (cont'd)
  - Modification Types
    - Fault Repairing modifications (FR)
      - Fixing a fault
      - Not used in calculating the complexity of the change process
      - Used to count fault in case study for validation
    - General Maintenance modifications (GM)
      - Bookkeeping modifications such as copyright update
      - Not used in analysis
    - **Feature Introduction modifications (FI)**
      - **Adding or enhancing features**
      - **Used to calculating the code change process complexity**

Appendix (2/6)

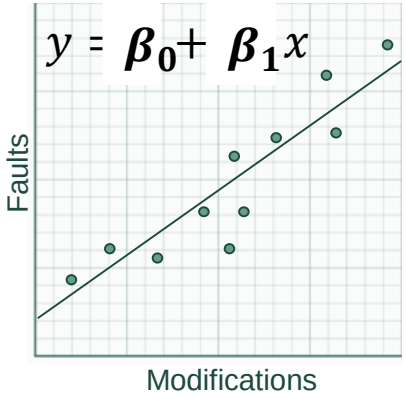
- SLR example for **x = number of modifications**
  - β_0 and β_1 estimated using fault data from **2nd and 3rd** years



Automatic Lexical Analysis

App #1

	Subsystem #1	Subsystem #2	...	Subsystem #n
x Modification Count	5	20	...	7
y Actual Fault Count	5	19	...	10



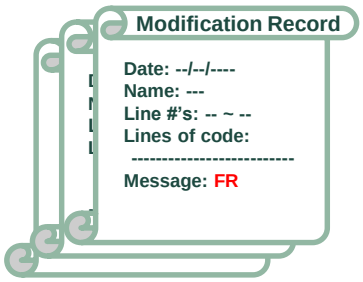
R^2 value

App	R_f^2	R_m^2	R_{1s}^2	R_{2s}^2	R_{3s}^2	R_{1d}^2
NetBSD	0.57	0.55	0.54	0.53	0.61	0.71
FreeBSD	0.65	0.48	0.57	0.58	0.59	0.65
OpenBSD	0.45	0.44	0.54	0.55	0.54	0.57
Postgres	0.57	0.36	0.49	0.51	0.60	0.61
KDE	0.31	0.26	0.28	0.29	0.36	0.57
KOffice	0.30	0.27	0.33	0.33	0.27	0.41

Appendix (3/6)

Prediction error example

- Predict faults in the application in the **4th and 5th** years



Automatic Lexical Analysis

$$\hat{y} = \boldsymbol{\beta_0} + \boldsymbol{\beta_1} x_i$$

		App #1 Subsystem #1	Subsystem #2	…	Subsystem #n
x_i	Modification Count	5	20	…	7
y	Actual Fault Count	5	19	…	10
$\hat{y}$	Predicted Fault Count	6	13	…	15
		e_1	e_2		e_n

$$\boldsymbol{E} = \sum_{i=1}^{n} e_i^2$$

Appendix (4/6)

- Modification vs. Faults

App	$E_m - E_f$ (%)	$P(H_0\ holds)$
NetBSD	+11.7 (+04%)	0.67
FreeBSD	+71.2 (+48%)	0.00
OpenBSD	+03.7 (+02%)	0.84
Postgres	+47.2 (+49%)	0.02
KDE	+26.3 (+07%)	0.32
KOffice	+26.3 (+04%)	0.51

  - Prior faults should be used to predict faults instead of prior modifications.

Appendix (5/6)

- Modifications vs. Entropy

App	$E_{HCM3s} - E_m$ (%)	$P(H_0\ holds)$	$E_{HCM1d} - E_m$ (%)	$P(H_0\ holds)$
NetBSD	-39.8 (-14%)	0.03	-106.5 (-36%)	0.00
FreeBSD	-47.4 (-22%)	0.02	-72.0 (-33%)	0.00
OpenBSD	-40.4 (-18%)	0.01	-53.8 (-23%)	0.00
Postgres	-52.7 (-37%)	0.04	-56.9 (-40%)	0.03
KDE	-52.1 (-13%)	0.01	-165.2 (-42%)]	0.00
KOffice	+03.3 (+01%)	0.83	-69.9 (-18%)	0.01

  - Both HCM based models are likely to outperform prior modifications

Appendix (6/6)

- Faults vs. Entropy

App	$E_{HCM3s} - E_f$ (%)	$P(H_0\ holds)$	$E_{HCM1d} - E_f$ (%)	$P(H_0\ holds)$
NetBSD	-28.14 (-10%)	0.26	-94.84 (-34%)	0.00
FreeBSD	+23.81 (+16%)	0.30	-00.79 (-01%)	0.97
OpenBSD	-36.59 (-16%)	0.02	-50.05 (-22%)	0.01
Postgres	-05.53 (-06%)	0.71	-09.71 (-10%)	0.55
KDE	-25.72 (-07%)	0.32	-138.87 (-38%)	0.01
KOffice	+19.20 (+05%)	0.34	-54.07 (-15%)	0.04

  - Models based on entropy are as good as (or even better) predictors of faults in comparison to prior faults for most studied software systems.