

# Software Entropy and Failure

Journal Club 2024-03-01



## Challenge

- We have many software heuristics when it comes to best practices.
- How can we quantify and scientifically improve upon these practices (within the context of avoiding failures)?



Journal article focus



A. E. Hassan, "Predicting faults using the complexity of code changes".

DOI: [10.1109/ICSE.2009.5070510](https://doi.org/10.1109/ICSE.2009.5070510)

## Title definitions

- Faults: *"In document ISO 10303-226, a fault is defined as an abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure."* ([Wikipedia: Fault \(technology\)](#))
- Complexity: *"... un-certainty/randomness/complexity ..."* (article Section 4)



## Complexity

- Complexity as uncertainty or randomness may be understood as *entropy* from information theory.



Complexity (information theory)

*In information theory, the entropy of a random variable is the average level of "information", "surprise", or "uncertainty" inherent to the variable's possible outcomes.*

[Wikipedia: Entropy \(information theory\)](#)



Complexity (information theory)

*Entropy: 1. A measure of the disorder present in a system.*

[Wiktionary: Entropy](#)



Complexity (information theory)

Wait a minute, what even is "information" in this context?





Complexity (information theory)

# The Bell System Technical Journal

*Vol. XXVII*

*July, 1948*

*No. 3*

---

## A Mathematical Theory of Communication

By C. E. SHANNON

$$I(E) = -\log_2(p(E))$$

Information conveyed by an event or message is inversely proportional to its probability. That is, the less probable an event is, the more information it carries when it occurs.



Complexity (information theory)



Imagine a coin toss as a "message" with two equally probable outcomes.



In [3]:

```
import math

# Probability of each outcome for a fair coin toss
probability_heads = 0.5
probability_tails = 0.5

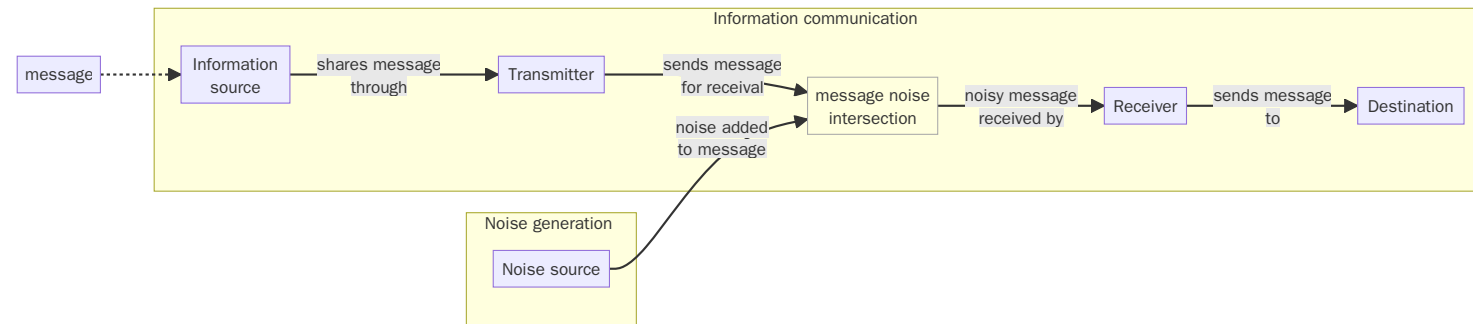
# Calculate Shannon information content for each outcome
information_heads = -math.log2(probability_heads)
information_tails = -math.log2(probability_tails)

print(f"Shannon information content for heads: {information_heads:.4f}")
print(f"Shannon information content for tails: {information_tails:.4f}")
```

```
Shannon information content for heads: 1.0000 bits
Shannon information content for tails: 1.0000 bits
```



## Complexity (information theory)



Entropy:

$$H(X) = - \sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$$

As understood through Shannon's communication system.

```
In [5]: import math

# Probability of each outcome for a fair coin toss
probability_heads = 0.5
probability_tails = 0.5

# Calculate Shannon entropy for the coin toss
entropy = -(
    probability_heads * math.log2(probability_heads)
    + probability_tails * math.log2(probability_tails)
)

print(f"Shannon entropy for the fair coin toss: {entropy:.4f} bits")
print(
    "\nThe result can be understood as 'maximum uncertainty' as it's co
)
```

Shannon entropy for the fair coin toss: 1.0000 bits

The result can be understood as 'maximum uncertainty' as it's completely unpredictable (equal probabilities).



```
In [6]: import math

# Probabilities for the unfair coin toss
probability_heads = 0.3
probability_tails = 0.7

# Information content for heads and tails
information_heads = -math.log2(probability_heads)
information_tails = -math.log2(probability_tails)

# Entropy calculation
entropy = -(
    probability_heads * math.log2(probability_heads)
    + probability_tails * math.log2(probability_tails)
)

print(f"Information content for heads: {information_heads:.4f} bits")
print(f"Information content for tails: {information_tails:.4f} bits")
print(f"Entropy for the unfair coin toss: {entropy:.4f} bits")
print(
    "\nWe have more information from heads as it's less likely. We have
")
```

```
Information content for heads: 1.7370 bits
Information content for tails: 0.5146 bits
Entropy for the unfair coin toss: 0.8813 bits
```

We have more information from heads as it's less likely. We have less information from tails as it's more likely. We have less entropy because the outcome is more predictable.



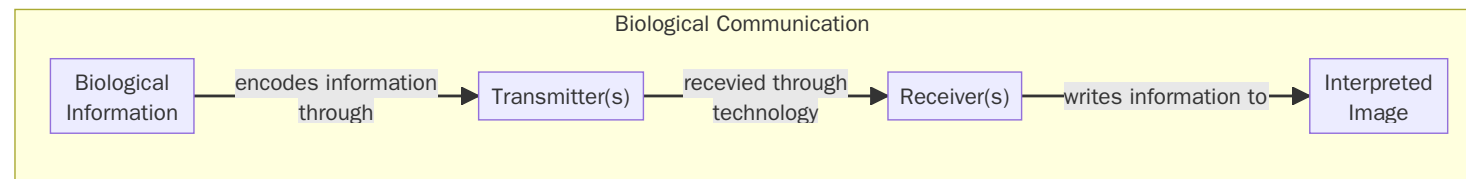
Complexity (information theory)

Questions in the context of software:

- What are the software "messages"?
- What are the software "probabilities"?



## Complexity (information theory)



Why does this matter here? (one rough take!)



## Prior work with Software Failures and Complexity Measures

Mentioned in the article:

- Prior modifications to a file are a good predictor of its fault potential (i.e., the more a file is changed, the more likely it will contain faults).
- Most code complexity metrics highly correlate with LOC (lines of code), a much simpler metric.
- Process metrics outperform code metrics as predictors of future faults.
- Prior faults are good predictors of future faults.

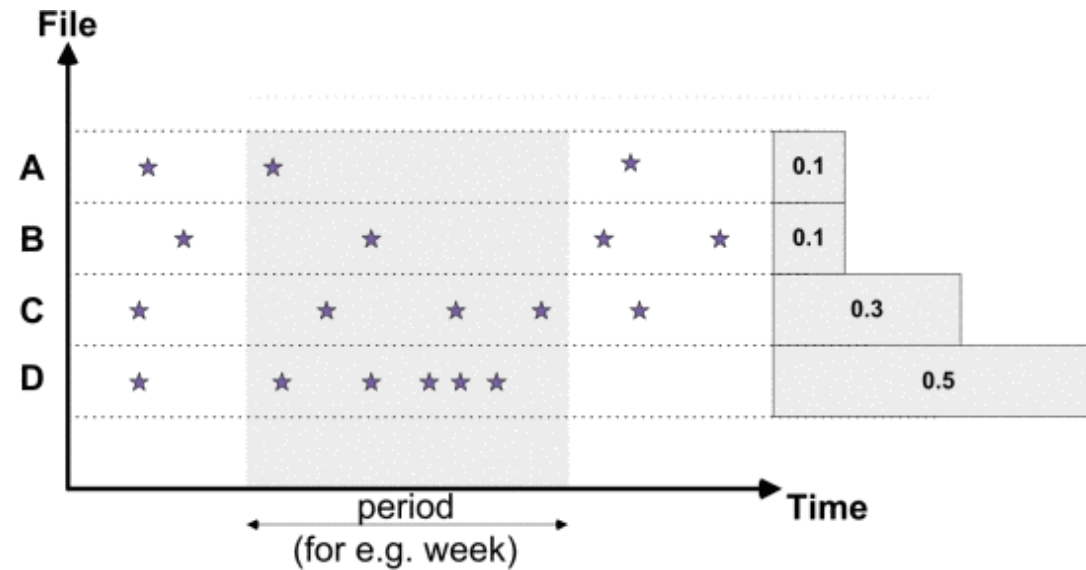


"Predicting faults using the complexity of code changes" Conjecture

- *A complex code change process negatively affects its product, the software system.*
- *The more complex changes to a file, the higher the chance the file will contain faults.*



## Section 4: Basic Code Change Model (BCC)



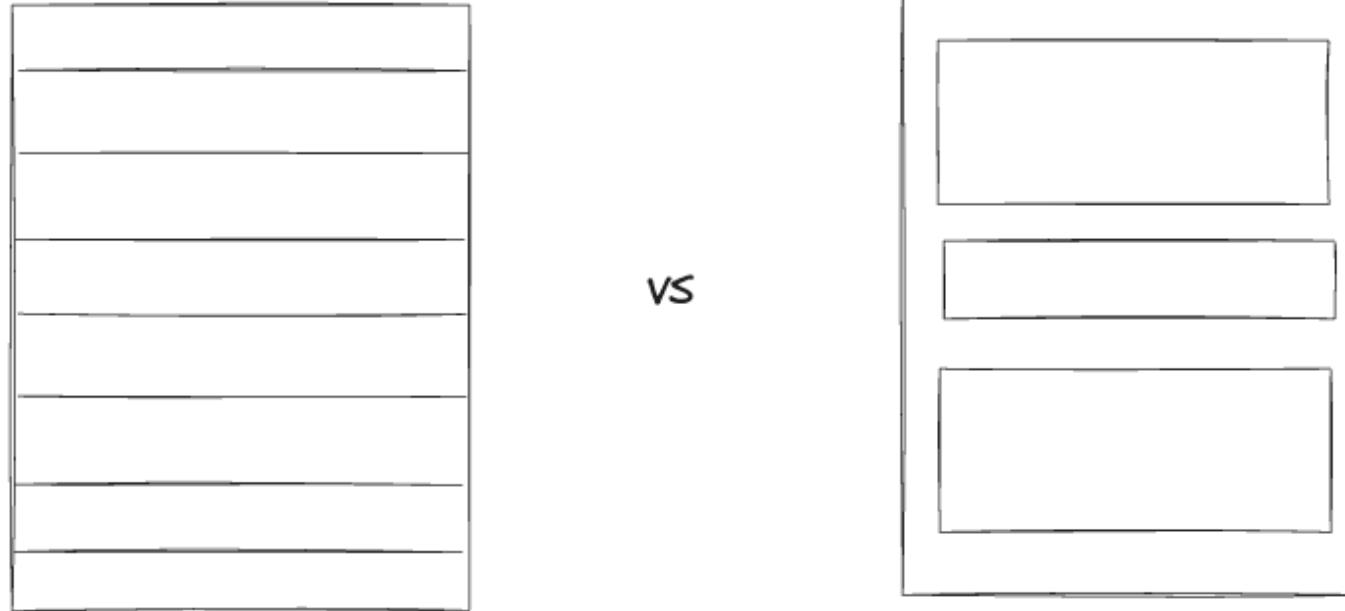
- FI modifications context (file interactions?)
- Files A, B, C, D are part of a software system.
- Stars are when changes occur to the files over a certain time period (such as a week).
- P gives the probability that a file is changed in a period.

## Section 4: Basic Code Change Model (BCC)

+3 -1 ■■■■ ■

- Instead of solely using the number of changes to the file, lines added or removed are summed to help build additional detail for understanding the modifications.
- In the above example, we'd have a total of 4 concerning the lines modified for a single file.

## Section 4: Basic Code Change Model (BCC)

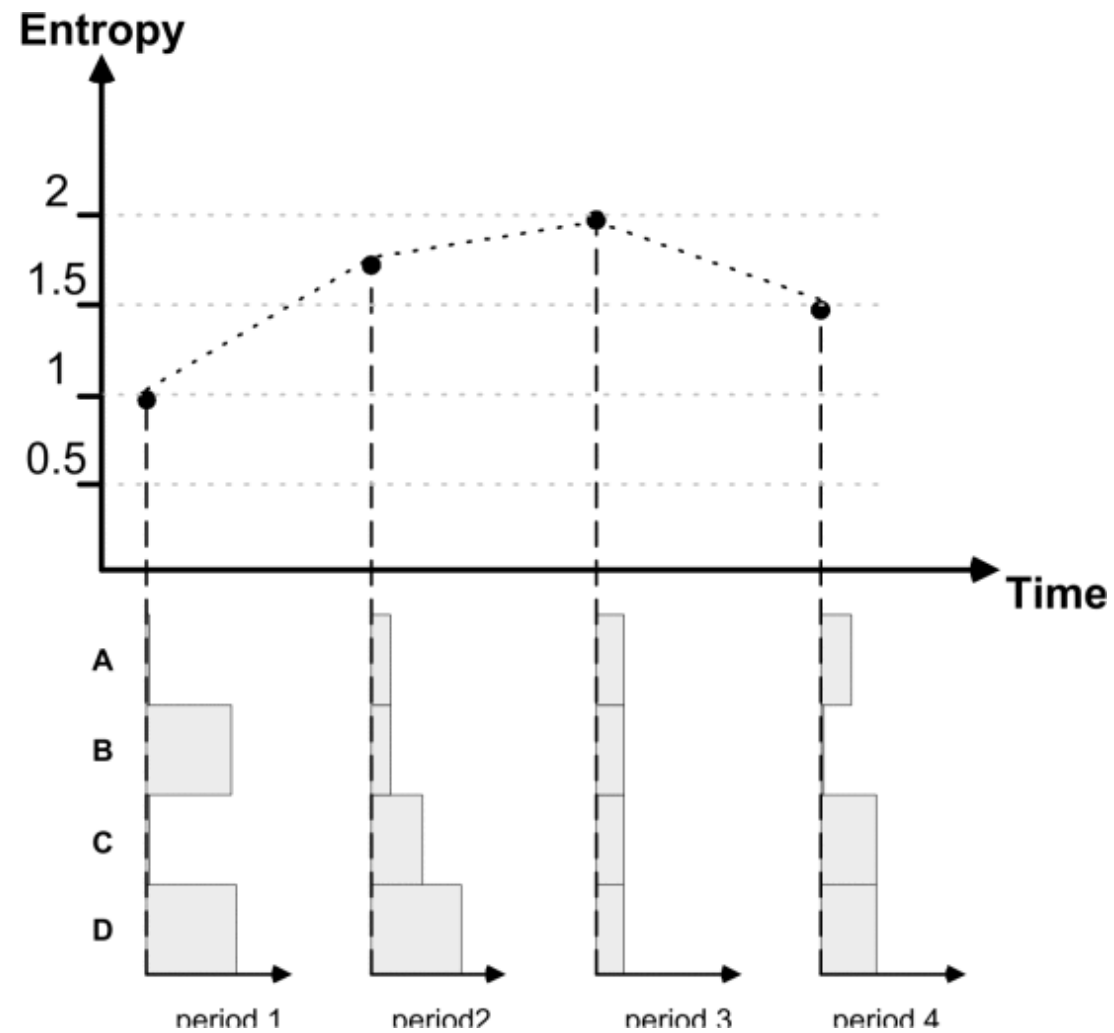


### Files vs internal modularity

- *"... choice of files is based on the belief that a file is a conceptual unit of development where developers tend to group related entities such as functions and data types"* (functions and other in-file modularity change based on language or style).



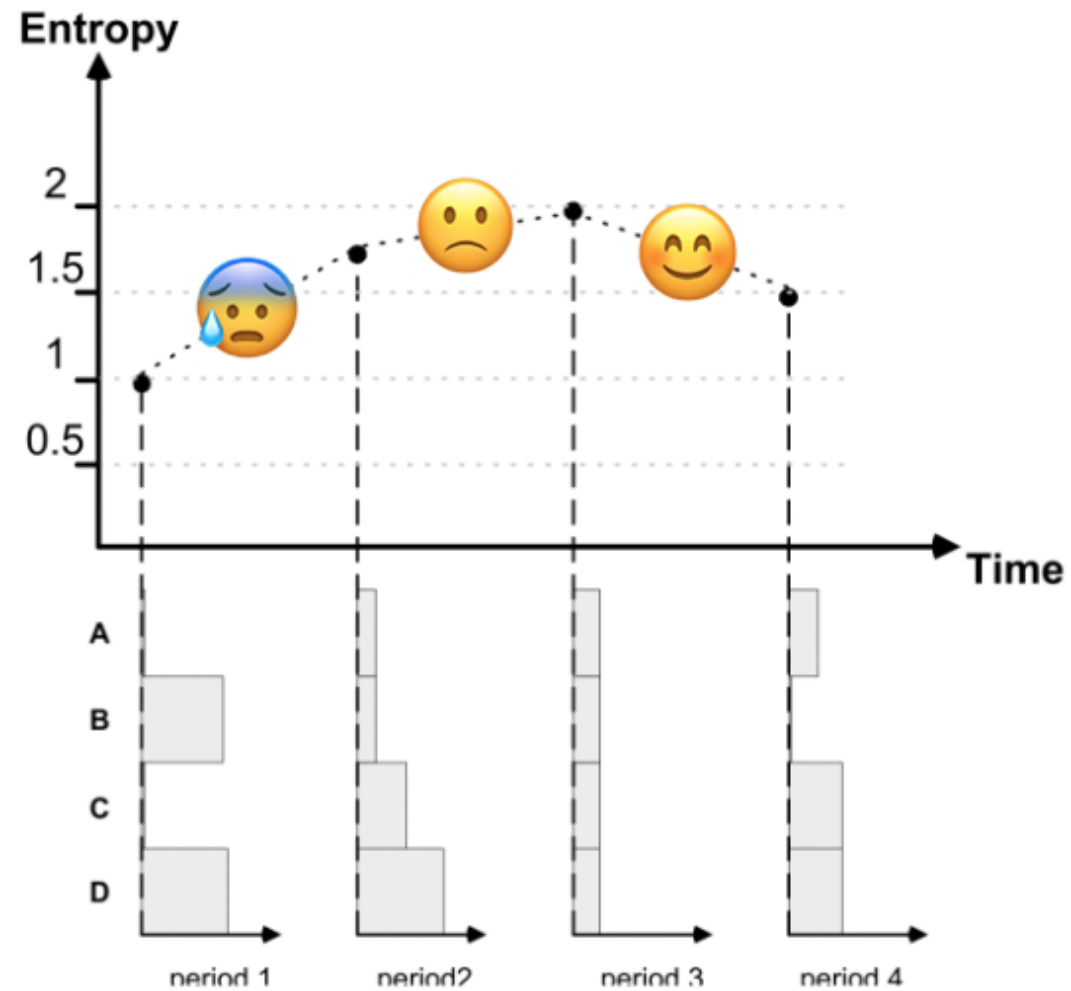
## Section 4: Basic Code Change Model (BCC)



- Entropy as measured through 4 time periods for multiple files in a software system



## Section 4: Basic Code Change Model (BCC)



## Section 4: Basic Code Change Model (BCC)

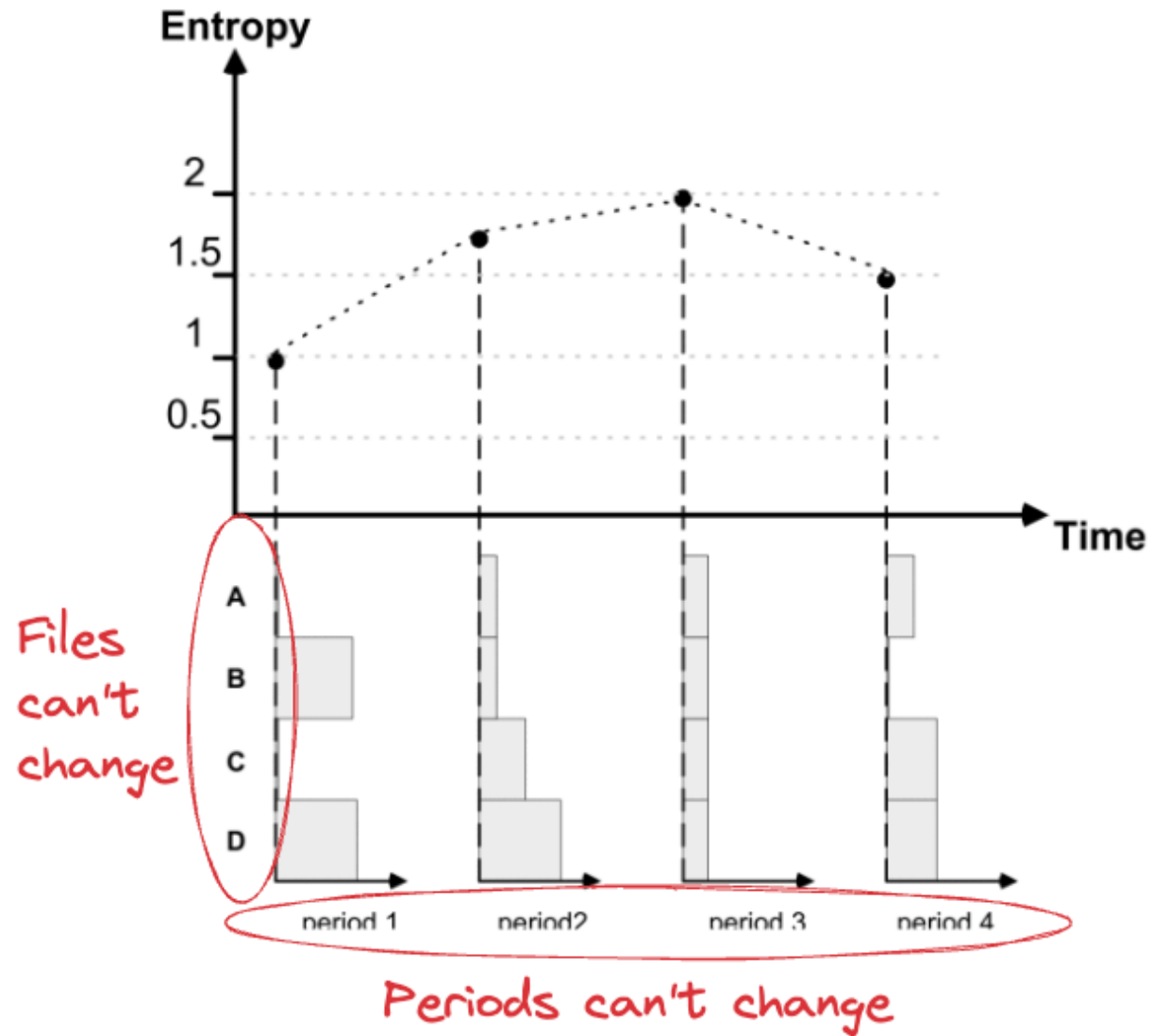
### Suggested conclusions

- Monitoring for unexpected spikes in entropy and investigating the reasons behind them would let developers plan ahead and be ready for future problems.
- Would expect the entropy to remain high for a limited time period then to drop as the refactoring eases future modifications to the code
- Complex code base may cause a consistent rise in entropy over an extended period of time, until the issues causing the rise in change entropy/complexity are addressed and resolved





## Section 5: Extended Code Change Model (ECC)

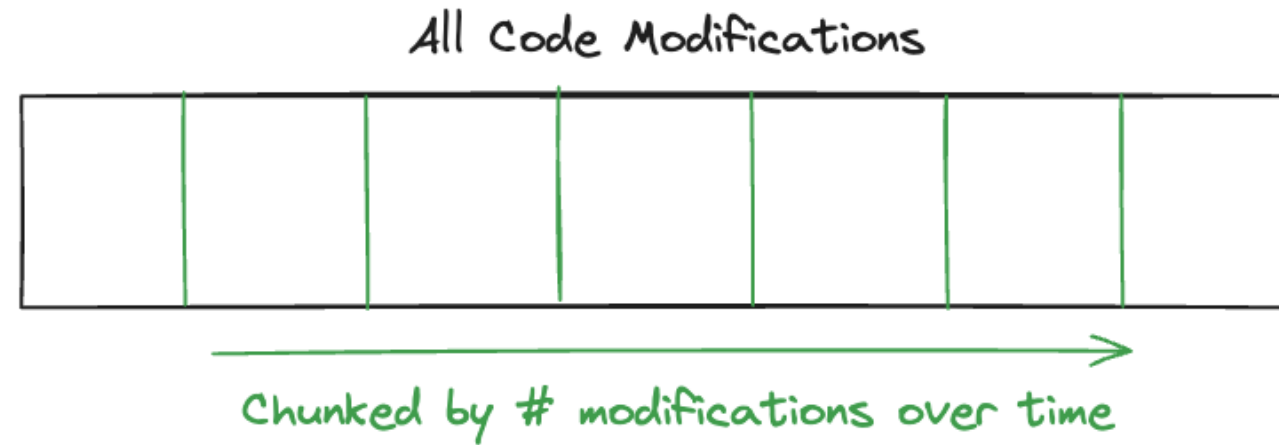


## Section 5: Extended Code Change Model (ECC)

- The "basic" BCC model has limitations through static file count and time periods.
- As a result, we need to "extend" on the basic model.

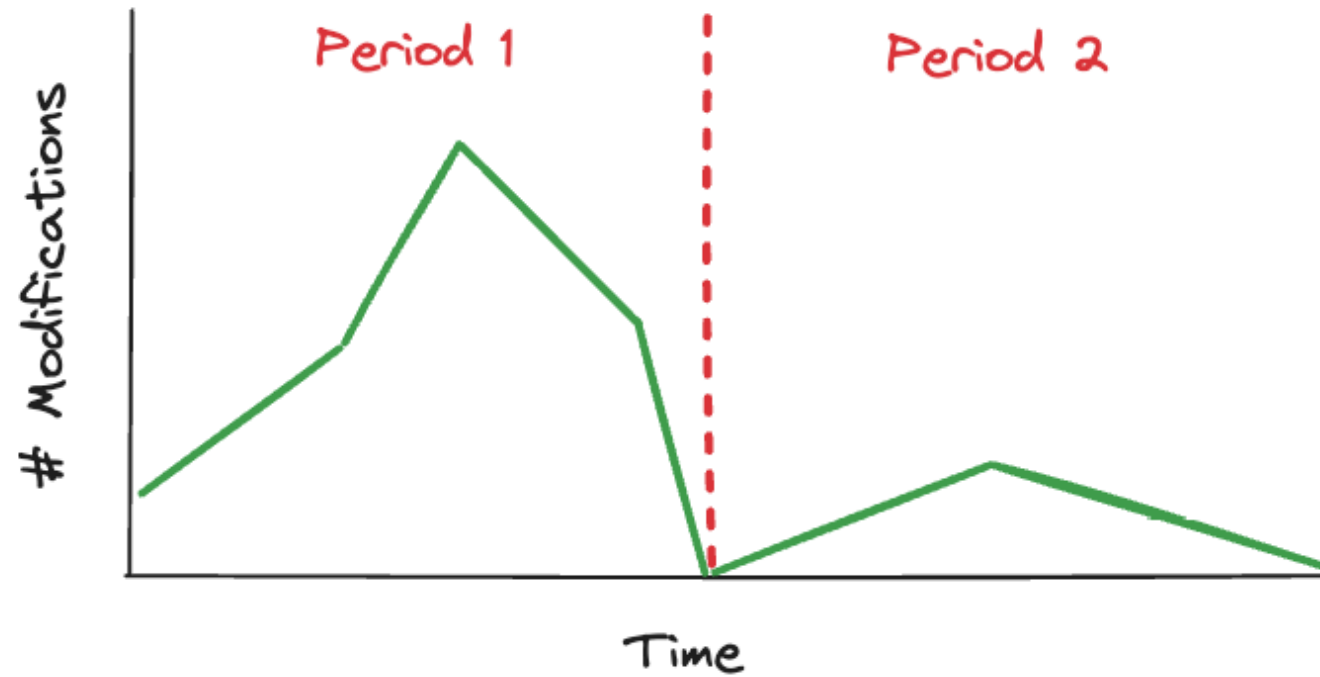


## Section 5: Extended Code Change Model (ECC)



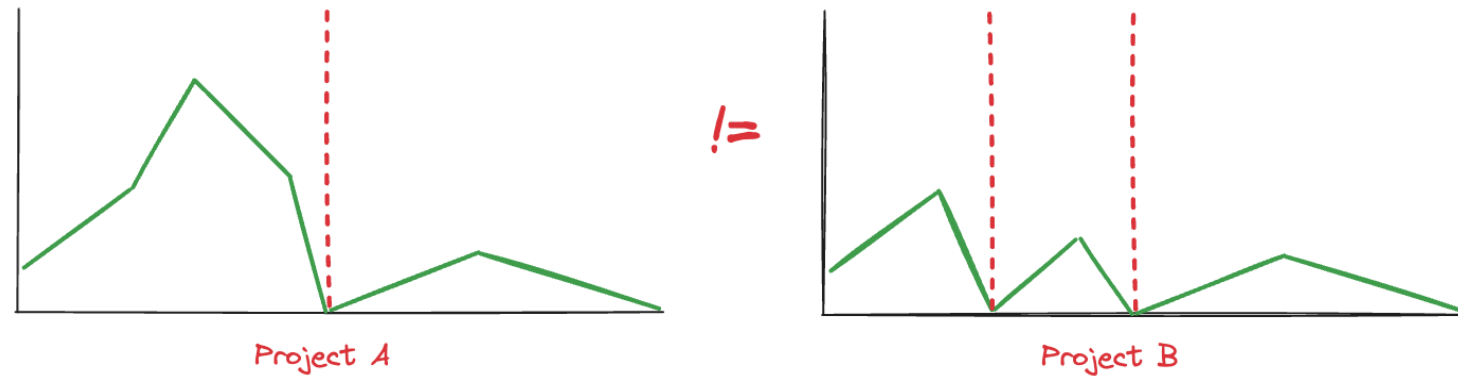
- Time "Evolution Periods" Modification limit based periods.
- Uses a certain number as a way to "chunk" by modifications over total time for the project.

## Section 5: Extended Code Change Model (ECC)



- Time "Evolution Periods" Burst based periods.
- Calculates dynamic periods based on modifications and time, segmenting by periods of modification vs none.

## Section 5: Extended Code Change Model (ECC)



- We now have a way to use more dynamic time periods for code changes.
- However, Shannon entropy calculations don't inherently allow for comparisons between distributions of different sizes (each software project could have a different number of distributions or in this case time periods).






## Section 5: Extended Code Change Model (ECC)

$$H(P) = - \sum_{k=1}^n (p_k * \log_n p_k)$$

- A normalized static entropy is introduced to help with varying distribution sizes to enable software project cross-comparisons.



## Section 5: Extended Code Change Model (ECC)

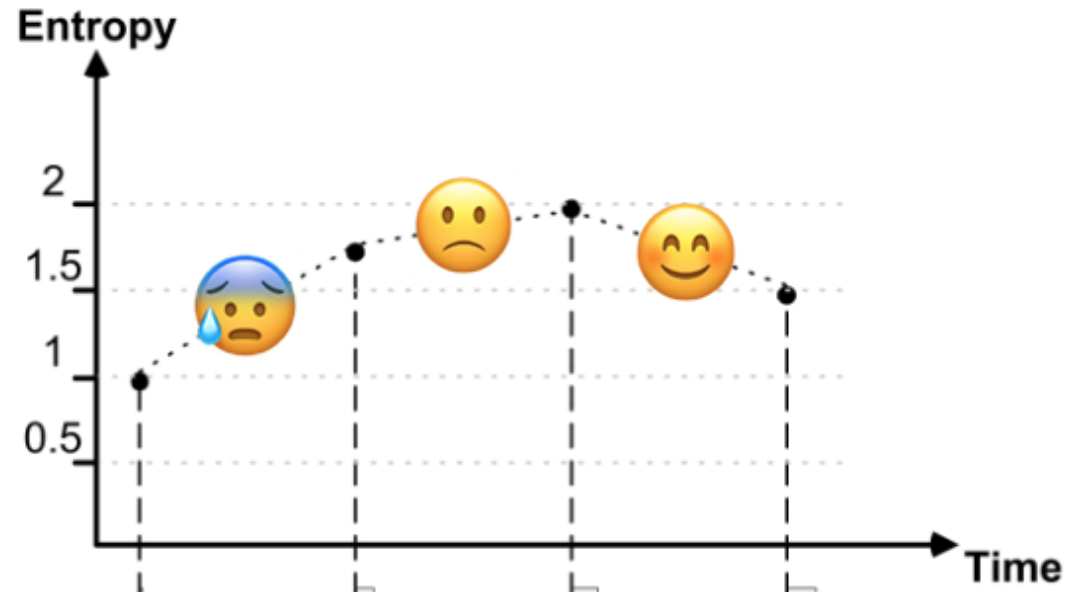
 Makefile	update	3 years ago	✗
 README.md	Update Readme wit...	2 months ago	✓
 make.bat	update	3 years ago	✗
 poetry.lock	Bump aiohttp from 3...	last month	✓
 pyproject.toml	docs(changelog): ad...	2 weeks ago	✓

Example: files modified within the last 6 months.

- Additionally, adaptive sizing entropy ( $H'$ ) is introduced.
- Adaptive sizing entropy includes only *recently* modified files (relative to the project) to avoid discrepancies caused by rarely modified files.
- Recently can mean:
  - Time: "modified within the last x months."
  - Previous periods: "modified within the last x periods."



## Section 6: File Code Change Model



- History Complexity Metric (HCM) is introduced.
- *"We believe that files that are modified during periods of high change complexity, as determined by our ECC Model, will have a higher tendency to contain faults."*



### History Complexity Metric (HCM)

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

- HCM measures how files which have been modified during periods of high complexity/entropy will tend to be more prone to faults.
- HCM assigns to a file the effect of the change complexity of a period, as calculated by the ECC model.
- History Complexity Period Factor ( $HCPF_i$ ) is calculated for each file per time period.



## History Complexity Period Factor (HCPFi)

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

- History Complexity Period Factor ( $HCPF_i$ ) is calculated for each file per time period.
- $c_{ij}$  is the contribution of entropy for period  $i(H_i)$  assigned to file  $j$ .
- $c_{ij}$  can be defined as follows:
  1.  $HCM^{1s}$   $c_{ij} = 1$ : assumes full complexity value to every file modified during a period.
  2.  $HCM^{2s}$ ,  $c_{ij} = p_j$ : measures percentage of complexity per file within a period. Assumes that the frequency of modification effects how a file should be measured.
  3.  $HCM^{3s}$ ,  $c_{ij} = \frac{1}{|F_i|}$ : distributes the complexity evenly among all files for a period.
  4.  $HCM^{1d}$ : which leverages  $HCM^{1s}$  with a decay model  $e^{\phi * (T_i - Current\ Time)}$



## Section 7: Case Studies (Linear Regression Models)

Application Name	Application Type	Start Date	Subsystem Count ( <i>low level directories</i> )	Prog. Lang.
NetBSD	OS	March 1993	235	C
FreeBSD	OS	June 1993	152	C
OpenBSD	OS	Oct 1995	265	C
Postgres	DBMS	July 1996	280	C
KDE	Windowing System	April 1997	108	C++
KOffice	Productivity Suite	April 1998	158	C++

- Linear regression models were created to measure prediction capabilities of future faults.
- Faults are defined as: *"the number of Fault Repairing (FR) modifications which occurred in the subsystem during the fourth and fifth years."*
  1. Modifications vs. Faults (Are modifications or prior faults a better predictor of future faults?)
  2. Modifications vs. (HCM) Entropy (Are modifications or HCM entropy a better predictor of future faults?)
  3. Faults vs. (HCM) Entropy (Are prior faults or HCM entropy a better predictor of future faults?)



## Section 7: Case Studies (Linear Regression Models)

SLR Model	Value of $x$
$Model_m$	number of modifications for a subsystem.
$Model_f$	number of faults for a subsystem.
$Model_{HCM_{1s}}$	$HCM_{1s}$ value for a subsystem.
$Model_{HCM_{2s}}$	$HCM_{2s}$ value for a subsystem.
$Model_{HCM_{3s}}$	$HCM_{3s}$ value for a subsystem.
$Model_{HCM_{1d}}$	$HCM_{1d}$ value for a subsystem.

- All HCM models use the ECC bursty model with one hour quiet times between bursts.
- $HCM_{1d}$  uses decay ( $\phi$ ) of 10.



## Section 7: Case Studies (Linear Regression Models)

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

- $R^2$  (coefficient of determination) shows the quality of the model fit (0 is no correlation, 1 is maximum correlation).
- $HCM_{1d}$  (HCM entropy with decay model) performed best.



## Section 7: Case Studies (Linear Regression Models)

Key takeaways they found:

1. *"Prior faults are better predictors of future faults than the prior modifications."*
2. *"The HCM based predictors are better predictors of future faults than prior modifications or prior faults."*



## Reflections



- HCM entropy appears to be important when it comes to avoiding failure.
- However, less entropy would mean no growth, so we have to find balance.
- Perhaps [decision fatigue](#) is important here with regards to the people involved.
- [Innovation](#) requires an implicit risk of failure (more unpredictable) and learning.

# Thanks!

Questions / comments?

