Lecture 5:
Designing smart systems

Using statistical methods to anticipate user needs and actions with Bayesian strategies

Overview of the course

- Theory driven approaches to HCI
- Design of visual displays
- Goal-oriented interaction
- Designing efficient systems
- **Designing smart systems**
- Designing meaningful systems (guest lecturer)
- Evaluating interactive system designs
- Designing complex systems

Uniform text entry

Apple UK

Information gain per key press

$$h(x_i) = \log \frac{1}{p(x_i)}$$

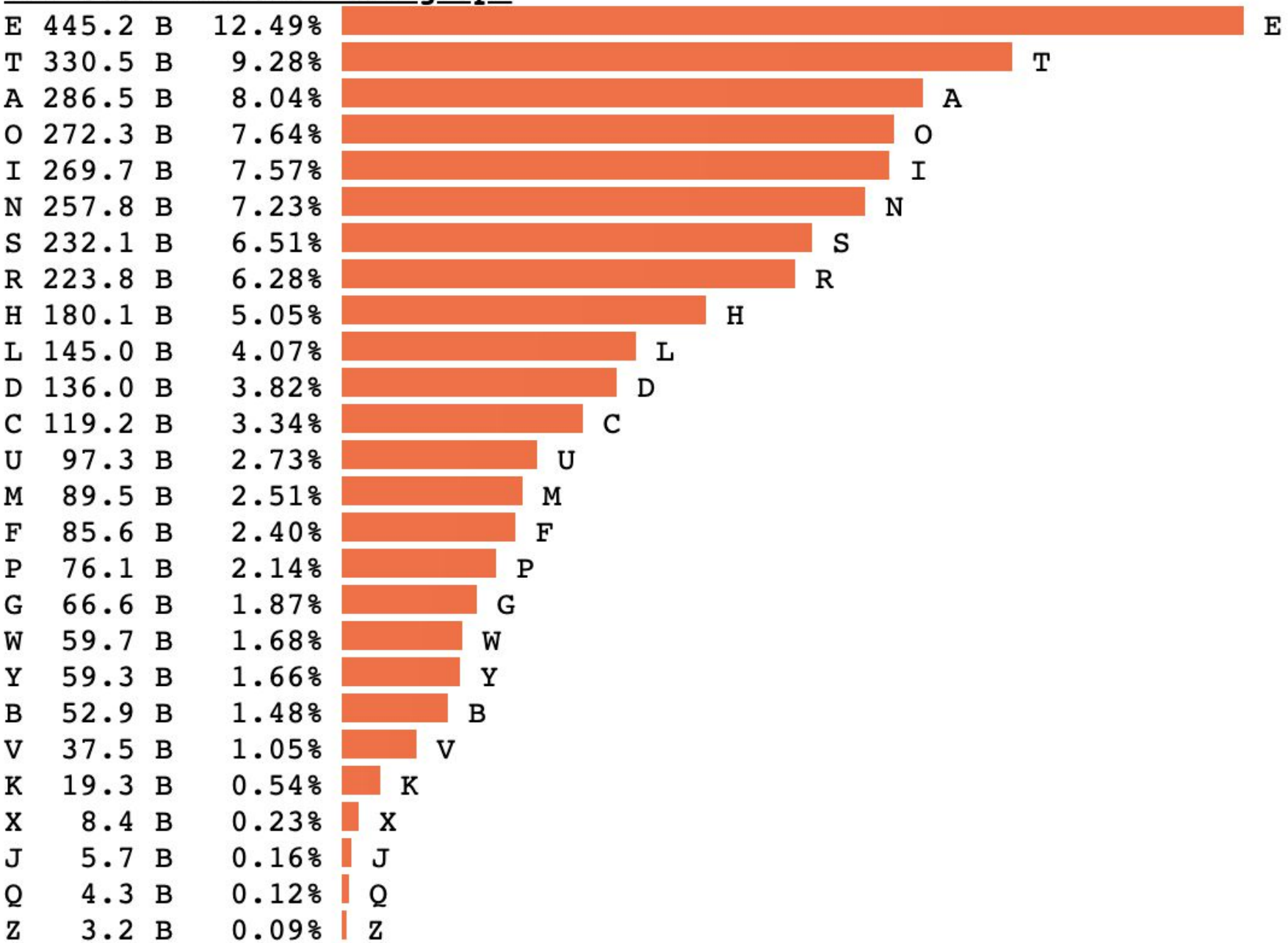
The q?

Information gain per key press

“As you are aware, E is the most common letter in the English alphabet, and it predominates to so marked an extent that even in a short sentence one would expect to find it most often”

The Adventure of the Dancing Men,

Sir Arthur Conan Doyle

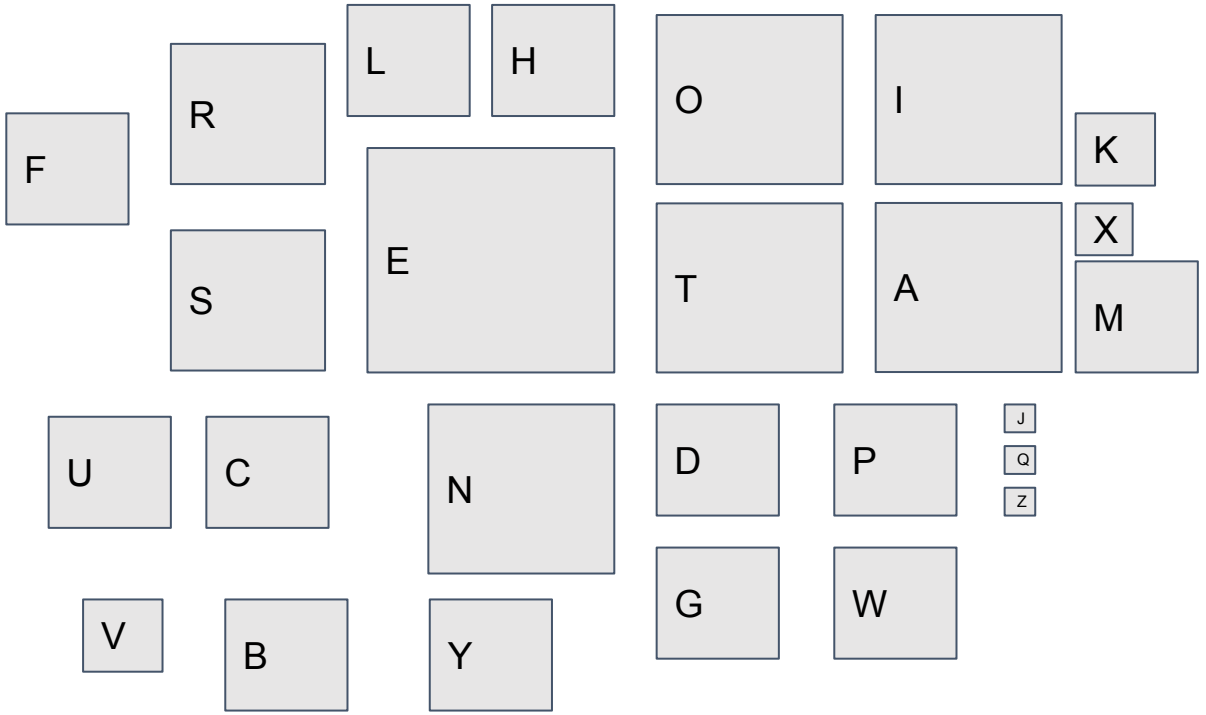


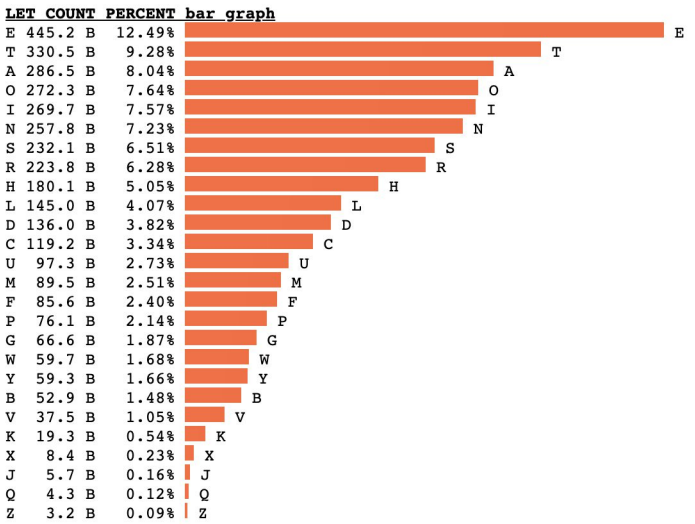
https://norvig.com/mayzner.html

Hacking Fitt’s Law: “semantic pointing”

Renaud Blanch, Yves Guiard and Michel Beaudouin-Lafon. **Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation.** In *Proceedings of CHI 2004*, pages 519-526, Vienna - Austria, April 2004.

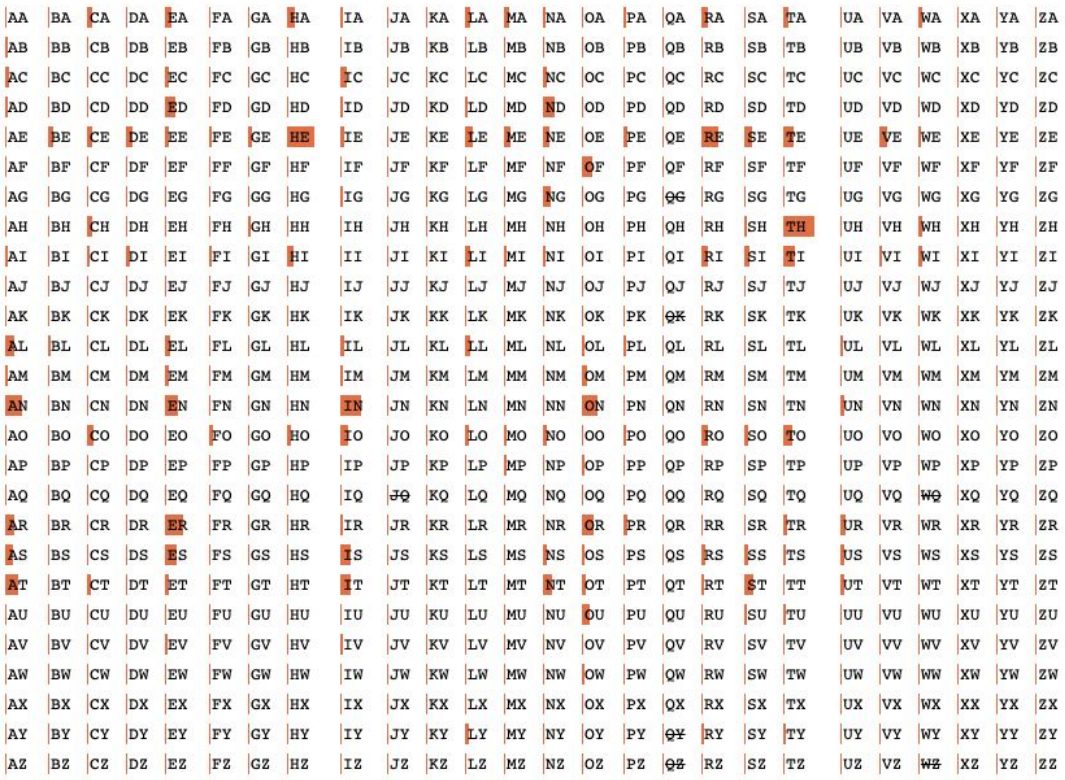
Simple application of Fitts Law





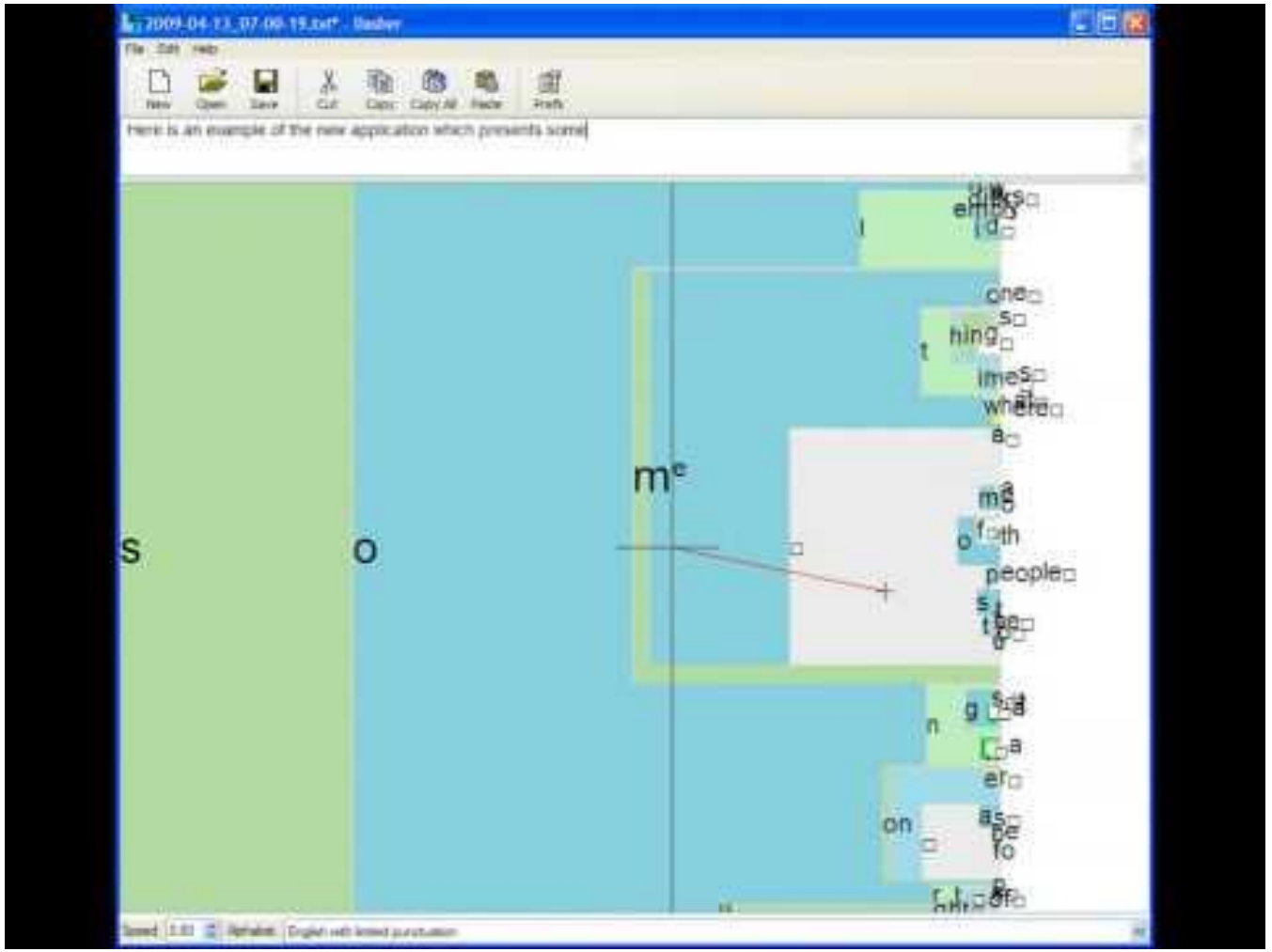
What's wrong with this?

Bigrams



Increasing the depth of the language allows for a further separation...

Building a system based on relative frequencies



Dasher
(https://www.youtube.com/watch?v=FLalNywdHxU)

Some lessons from Dasher

- Turning an information theoretic model into a user interface requires a lot of creativity

  => Interaction with Machine learning course

- In many cases simple models (nGrams + smoothing) are as - or more - effective than complex ones (neural nets)

- Supporting even famous software, useful for marginalised groups is hard

It guesses your thoughts, then types

DAVID MACKAY SET OUT to invent a better way of entering text on devices such as digital assistants and mobile phones. His creation, which he calls "Dasher", is a little like an arcade game: *Attack of the Killer Alphabets*, perhaps.

A reader in physics at Cambridge, he used his knowledge of probability to devise a system where the letters appear to flow – on the screen – towards the writer's pen or cursor. As the letters flood by, the shape of your word appears as if by magic, stretching out into the alphabet soup like a character in a colour blindness test.

It's smart maths rather than magic: the system guesses the word you are trying to write and flows the next character towards the cursor. It also learns the kinds of words you use.

Only minute movements of pen or cursor are needed, making Dasher a prime candidate for use by both able-bodied and disabled. It could, for example, be driven by a device which tracks eye movement.

MacKay is a co-founder of Transversal, a commercial venture which, again, exploits probability theory to make the interrogation of computer databases simpler. MacKay believes in sharing software and Dasher is free to download from the web, much to his Transversal colleagues' horror. Get your copy before they shrink wrap it.

www.inference.phy.cam.ac.uk/djw30/dasher/download.html

(The Financial Times, February 2002)

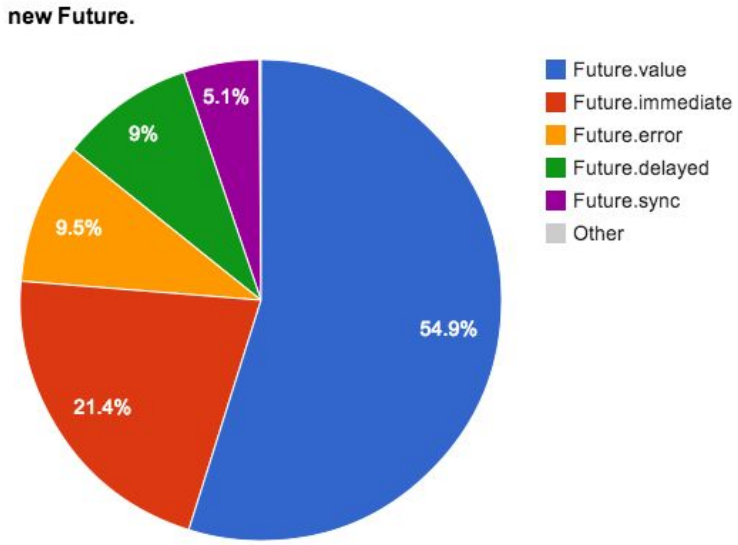
Artificial languages

```
new Future.?
```

```
New Future.aaaaa()
New Future.aaaab()
New Future.aaaac()
New Future.error()
…
New Future.aaaad()
New Future.aaaae()
New Future.value()
```

Artificial languages

```
new Future.?
```



Liveness becomes Entelechy, Church et al

Ordering code completion suggestions

A simple scheme for predicting code completions:

```
void            main()            {
     Stopwatch sw = new Stopwatch();
      sw. // <--- What goes here?
}
```

```
elapsed
elapsedMicroseconds
elapsedMilliseconds
elapsedTicks
Frequency
hashCode
isRunning
noSuchMethod
Reset
runtimeType
Start
Stop
toString
```

Ordering code completion suggestions

We calculate:

P(completion = “reset” | *context* = “void main() { Stopwatch sw = new Stopwatch(); sw.”*)*
P(completion = “start” | *context* = “void main() { Stopwatch sw = new Stopwatch(); sw.”*)*

…

And the usual:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

Ordering code completion suggestions

$$P(completion = ? \mid context = \text{“...”}) \propto P(context = \text{“...”} \mid completion = ?)\ P(completion = ?)$$

Feature vector

Completion c	Count of seen completions	P(completion)
start	10	0.5
reset	5	0.25
elapsed	5	0.25

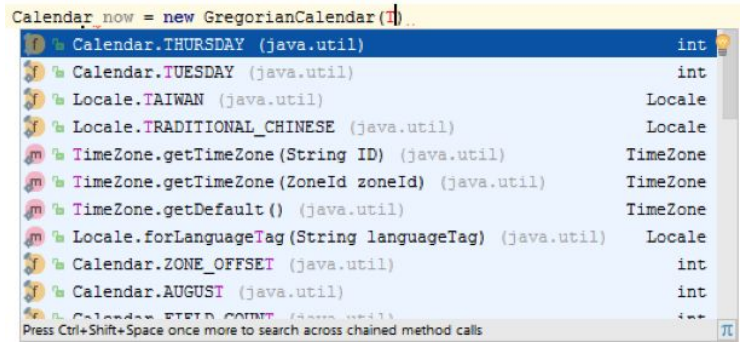
Ordering code completion suggestions

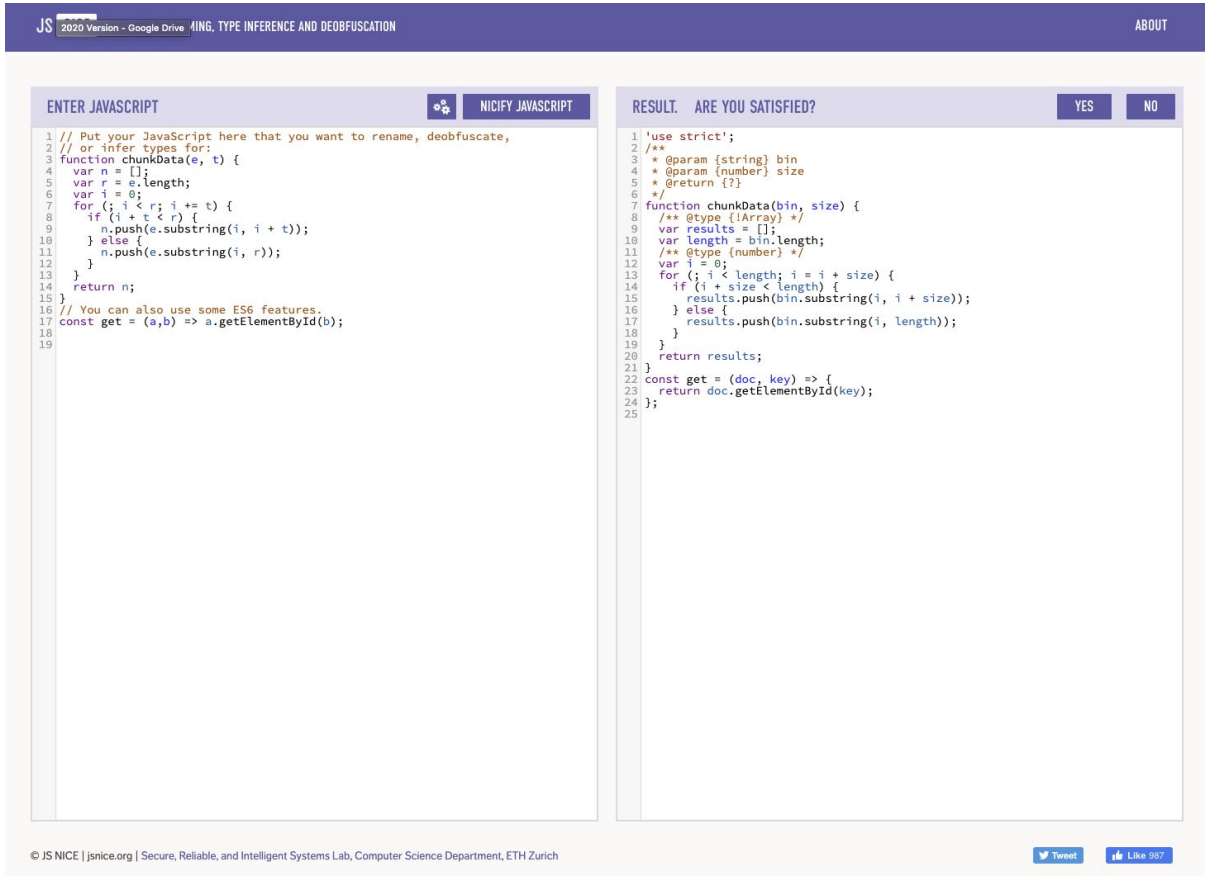
$$P(completion = c \mid context = \text{“...”}) \propto P(context = \text{“...”} \mid completion = c)\, P(completion = c)$$

Completion c	P(completion==c | context) ∝	Order
start	0.9 * 0.5 = 0.45	0
reset	0.4 * 0.25 = 0.1	1
elapsed	0.2 * 0.25 = 0.06	2

Completion c	Feature	Feature value	Count
start	“First-Use”	true	9
		false	1
reset	“First-Use”	true	2
		false	3
elapsed	“First-Use”	true	1
		false	4

Some progress in information efficient IDEs

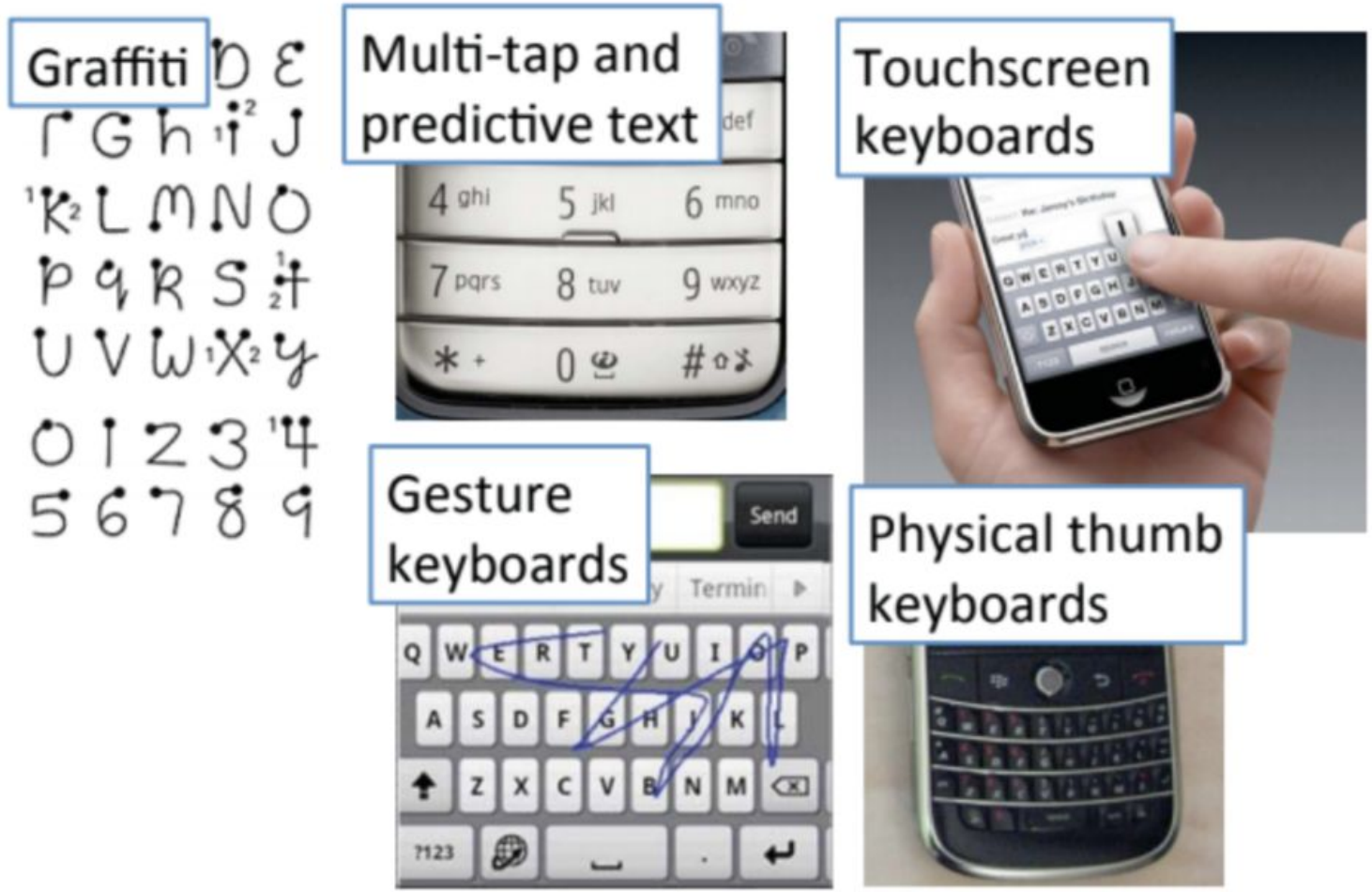




IntelliJ (Jetbrains)
JSNICE (ETH Zurich)

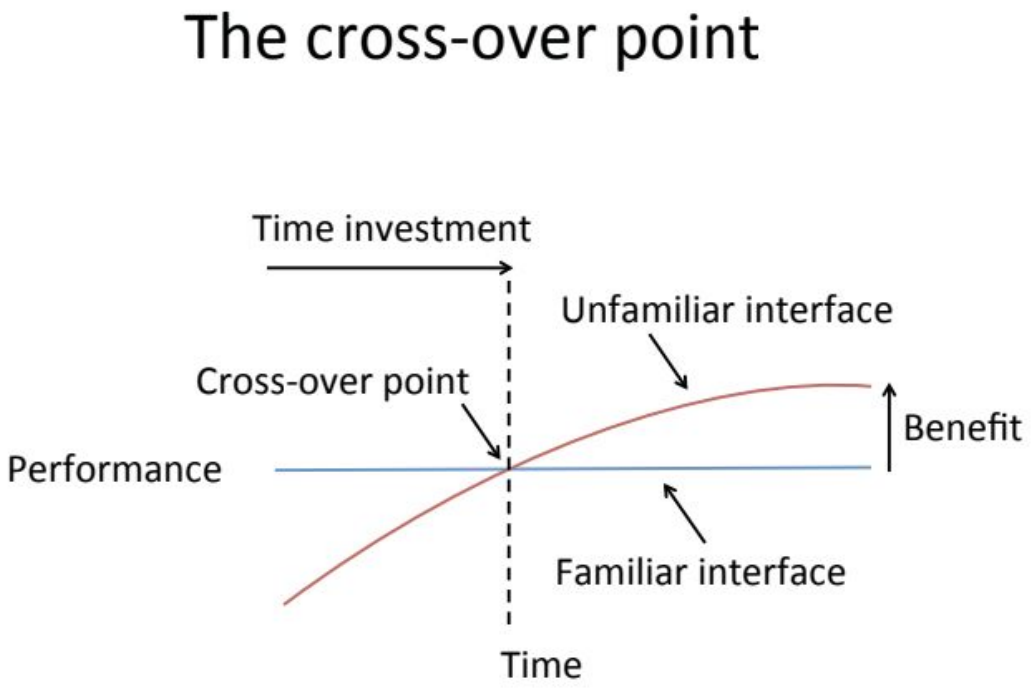
Building user interfaces

(from Per Ola's material)



Building user interfaces
(from Per Ola's material)

- Entry and error rate
- Learning curve, familiarity and immediate efficacy
- Form factor, presentation, time and comfort
- User engagement
- Visual attention and cognitive resources
- Privacy
- Single vs Multi-character entry
- Specification vs Navigation
- One/Two handed
- Task integration
- Robustness
- Device independence
- Computational demands
- Manufacturing and support cost
- Localisation
- Market acceptance



Building user interfaces: Solution principles
(from Per Ola's material)

- From closed to open-loop
  - Avoid the need for a visual feedback loop
- Continuous novice-to-expert transition
  - Avoid explicit learning
- Path dependency
  - Avoid redesign the interaction layer
- Flexibility
  - Enable users to compose and edit in a variety of styles without explicit mode switching
- Efficiency
  - Let users' creativity by the bottle-neck