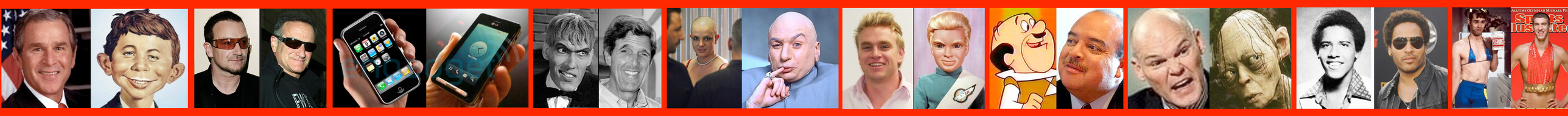
SEPARATED AT BIRTH

Related implementations ... reunited!



marius nita & david notkin university of washington

Related implementations are **coexisting**, syntactically **different**, **interchangeable** pieces of code. Some examples:

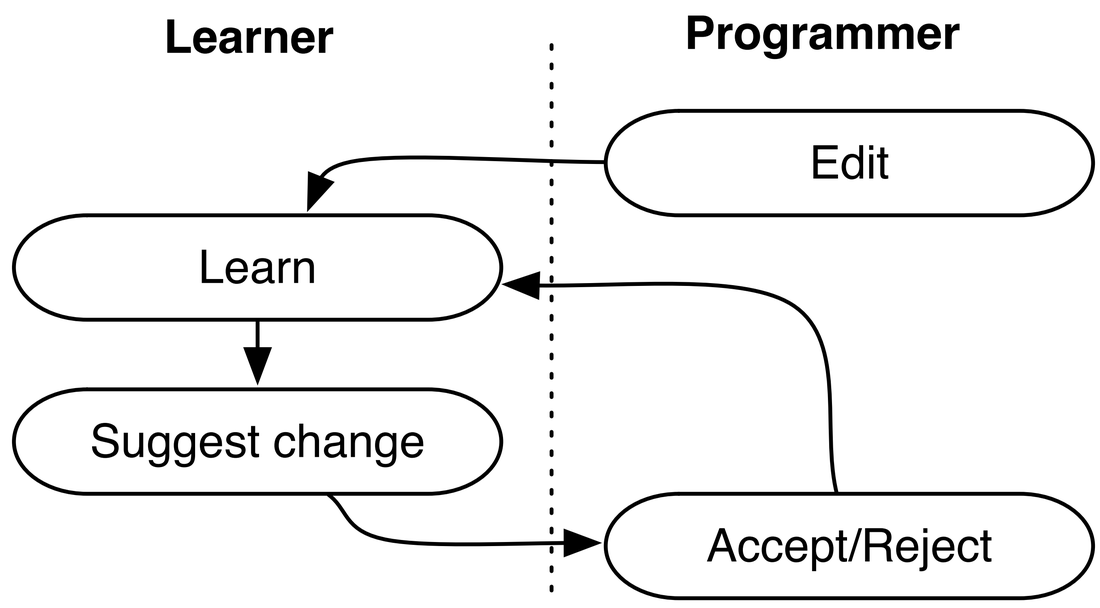
Implementations of an **interface** in OOP.
(E.g., list and array variants of a sequence.)

Interchangeable software **configurations**.
(E.g., Cocoa vs. Win32 GUI targets.)

Related implementations are **different**: they target different underlying data structures, hardware platforms, or third party APIs.

Related implementations are **similar**: they implement similar functionality and are indeed swappable: they respect a common contract.

```
pop()                          pop()
  if cursor == 0       <-->      if list == null
    throw Empty;                   throw Empty;
  r = arr[cursor-1]    <-->      r = list.data;
  cursor--;            <-->      list = list.next;
  return r;                      return r;
```



We propose a way to **track** the **underlying relationships** among related implementations:

Programmers explicitly **map** related code **elements together** during the creation or editing of an implementation.

A **learner** inputs the programmer's actions and builds a **mapping** of how the **original** implementation **relates to** the **new** one.

```
cursor == 0     <=>   list == null
arr[cursor-1]   <=>   list.data
cursor--        <=>   list = list.next
```

We **store** the mapping long-term.

Uses of the **mapping**:

Help the programmer in the editing process by incrementally **suggesting future mappings** and **changes** based on previous ones.

Determine whether a **code change** has **corresponding changes** in the other implementations.

Determine whether a new **test** has **corresponding tests** in the other implementations.

Suggest what the corresponding **changes** and tests should **look like**.

marius@cs.washington.edu