

Permission or Forgiveness?

http://www.aleax.it/oscon12_fop.pdf

Alex Martelli
Anna Martelli Ravenscroft

The Google logo is displayed in its characteristic multi-colored font (blue, red, yellow, green, red) with a slight 3D effect.

©2012 Google -- aleax@google.com

A fontly note to my critics

- the proportionally-spaced font used in my slides is NOT **Comic Sans** (as some [nasty?] critics have long alleged)
- it's Apple® **Chalkboard** (goes with the blackboard-background theme)
- it's *this one*, not *this one*!
- if you think Apple's visual designers have no taste, take it up with Cupertino...;-)

Permission or Forgiveness?

- "It's easier to ask forgiveness than permission"
- Rear Admiral Grace Murray Hopper, PhD (Mathematics, Yale); 1906–1992



The Amazing Grace

- Mark I, compilers, COBOL, "debugging", ...
- 1st "CS man (!) of the year", DPMA 1969
- 1st ever American AND 1st ever woman to be a Distinguished Fellow of the British CS
- Defense Distinguished Service Medal
- National Medal of Technology



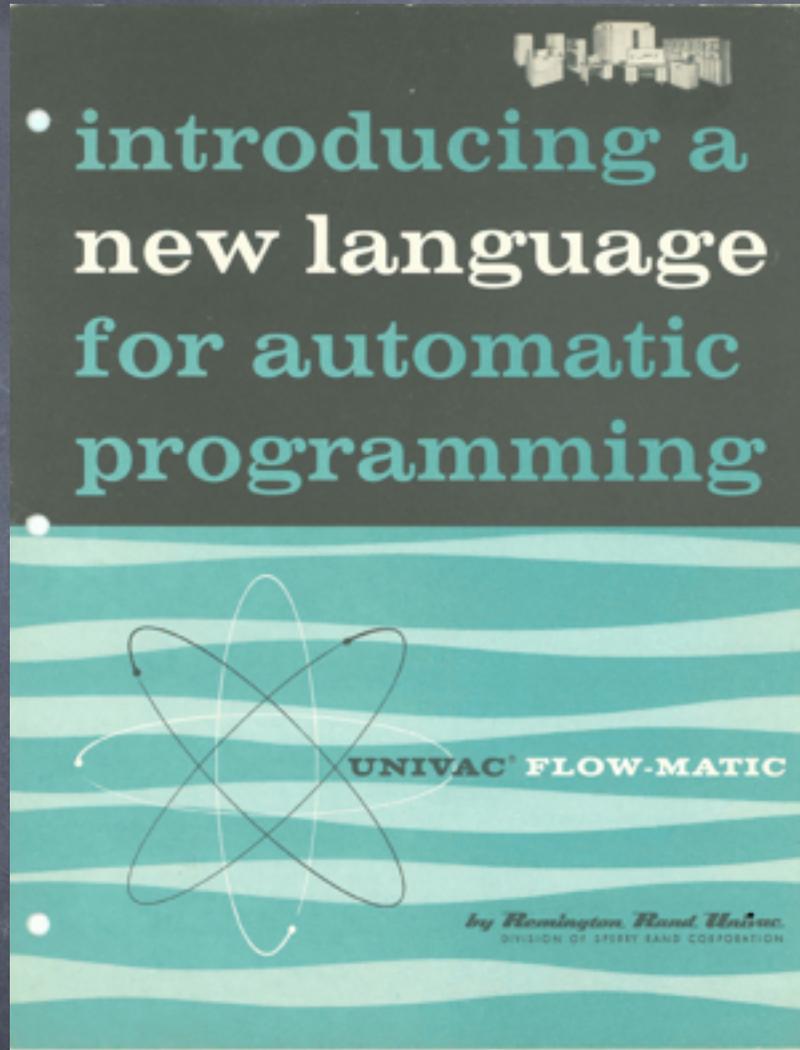
What did EAFP mean?

- GMH specifically referred to innovating from inside a bureaucratic organization
- for Hopper, that would be mostly the Navy, where she served for decades
- but clearly it also applies to large private firms with THEIR bureaucracies (mostly, middle managers...!-)

But, does EAFFP __work__...?

- Clearly, it worked very well for GMH, given:
 - her amazing track record of innovation
 - her promotions all the way to Commodore
 - 3rd party recognition showered on her
- we're not all GMH (we __wish__ we could be that brilliant...!), but, let's ponder..
 - how well did it work for her, again?
 - how/why would it work for us...?

Hopper EAFFP Examples



WHY does it tend to work?

- even the best bureaucrat has incentives to deny permission if and when asked
 - makes life/work more complicated
 - may present a career risk; denial doesn't
- a middle manager's job...:
 - listen carefully to new disruptive ideas...
 - ...then, say NO!
 - it's the org's self-preservation mechanism

WHY do you get forgiven?

- sometimes, the issue doesn't even come up
 - if the unauthorized project's a triumph
- in most cases, the bureaucrat will also, later, have incentives to grant forgiveness
 - (if the issue ever even comes up!-)
 - again: it's the path of least resistance / work / complications / career risk
 - especially for a successful "skunkworks" project

Beyond bureaucracy...?

- Python (exceptions vs checks)
- concurrency (optimistic vs locks)
- source-code control systems (!)
- networking (CD vs CA)
- "do it right the first time" vs "launch and iterate" and "fail, but fail FAST"
- when DOESN'T it work?
 - on SENSIBLE rules, principles, orgs
 - but especially: when it breaks Kant's Categorical Imperative...

"Permission"

```
def with_perm(filepath, default=None):  
    if os.access(filepath, os.R_OK):  
        with open(filepath) as f:  
            return f.read()  
    else:  
        return default
```

A.K.A "LBYL"

What's wrong w/this code? A lot -- it's even explicitly discouraged at [docs.python.org!](https://docs.python.org/)—)

"Forgiveness"

```
def with_forg(filepath, default=None):  
    try:  
        f = open(filepath)  
    except IOError:  
        return default  
    else:  
        with f:  
            return f.read()
```

A.K.A "EAFP"

Real vs effective UID -- but, setuid scripts are not usually a good/secure idea anyway;-).

Types: P or F? (1)

Once there was a similar choice...:

```
if isinstance(x, (int, float)):
```

```
    return x + 1
```

```
else:
```

```
    return default
```

...vs good ol' "duck typing"...

```
try:
```

```
    return x + 1
```

```
except TypeError:
```

```
    return default
```



Types: P or F? (2)

But, today...:

```
if isinstance(x, numbers.Number):  
    return x + 1  
else:  
    return default
```

...is an idiomatic, well-supported way to perform "typeclass"-checking... (for the relatively few "typeclasses" supported as ABCs in the **collections** and **numbers** standard library modules; a framework might add some more).

Write a new typeclass ABC?

- if it captures an important, well-recognized abstraction (e.g "polygon", "image", "sound", ...)
- capable of multiple implementations
 - warranted for important performance / memory-footprint trade-offs
- a core concept in the framework's field
- ideally with the ability to supply useful auxiliary methods (though "pure interfaces" may be OK too, esp. within a "family")
- maybe subclasses some existing ABC...
- beware of the "guy with a hammer" syndrome!-)
 - not ALL problems are nails...

Defaults: even better!

```
if hasattr(x, 'foo'): ...
```

VS

```
getattr(x, 'foo', 'bar')
```

```
if key in d:
```

VS

```
d.get(key, 'default')
```

Neither Permission nor Forgiveness,
but rather: a useful "Plan B"!-)

Optimistic Concurrency

- traditional "permission" approach:
 - acquire locks / mutexes guarding all needed resources, THEN perform the desired set of operations
- modern, speedy "forgiveness" approach:
 - perform the desired set of operations within a "transaction"
 - must be able to detect and reject rare transactions which suffered "collisions"
 - retry if needed (rarely... one hopes!-)
- OCC, STM, ...

E.g: source-code control

- bad old "permission" way:
 - "check out" all files you need to change
 - blocks everybody else's access to them
 - develop on your WS: change, test, &c
 - "commit" the changeset
 - releases the files
- much better, popular "forgiveness" way:
 - change, test, &c, on local file copies
 - "commit" the changeset
 - detects conflicts, forces reconciliation

Networking

- "permission": e.g "token ring" - only the node with the token can put a packet on the wire, then (or instead) passes the token; avoids collisions
- "forgiveness": e.g "Ethernet" - just "start talking" (if the wire's not busy) - detect collisions, "back off" & retry later
- again: can be much faster (except under unbearable overload conditions where it "thrashes") -- sometimes more robust, but that depends on many other details

Launching a Product

- "slow but safe", permission-ish approach:
 - studies, focus groups, &c
 - find out what consumers "want"
 - top-down design and development
 - and finally the Big Launch
- "agile", forgiveness-like approach:
 - launch ("beta"!) early, iterate often
 - based on real-world feedback
 - "fail, but fail FAST"
- varies by product type, innovation, cost, ...

Products & Forgiveness

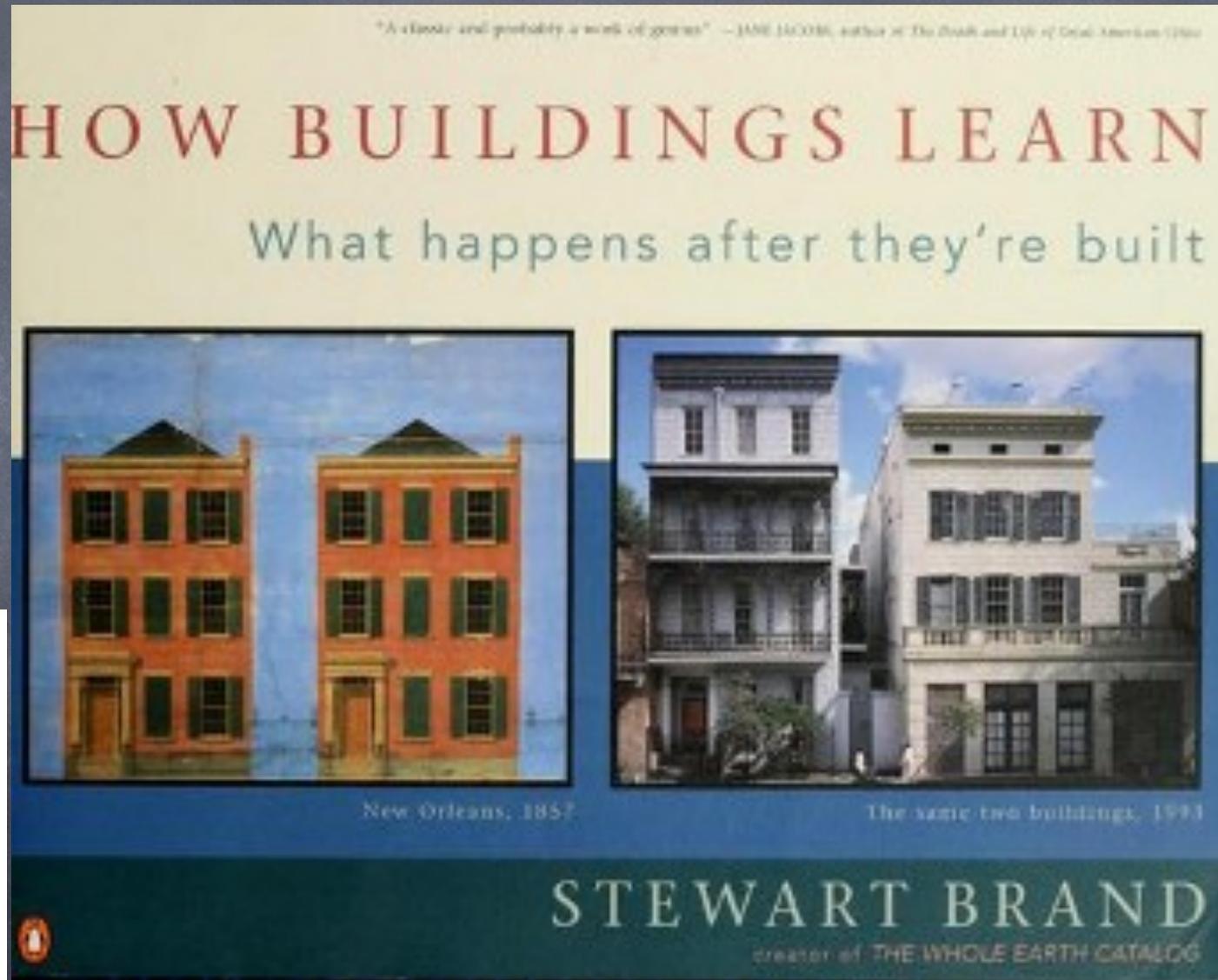
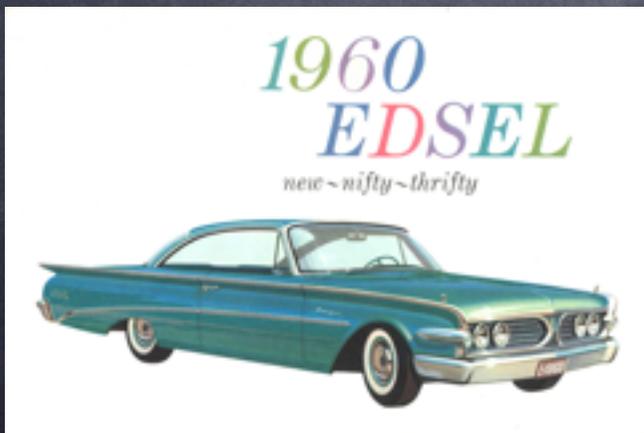


Products & Forgiveness

1960
EDSEL
new~nifty~thrifty



Products & Forgiveness



When NOT a good idea (1)

- when there are sensible, appropriate rules and principles in place,
 - AND a sensible, appropriate process to work with them
- e.g: mandatory code review before commit to the reference repository is allowed
 - pair programming not a good alternative
- "working with The System" may sound "too mainstream" but it can most often be a good idea (in the right environment)!

When NOT a good idea (2)

- one of the strongest examples...:
- mandatory preliminary reviews of product plans and architectures by security experts to spot privacy/security risks
 - security cannot be "an afterthought"!
- procedural arrangement becomes crucial
 - "pre-coding" architectural review
 - "post-coding" security/privacy review
- the difficult part: not too hot, nor too cold

When NOT a good idea (3)

- in a lot of common human interactions
- Kant's "Grundlegung zur Metaphysik der Sitten": "Act in such a way that you treat humanity, whether in your own person or in the person of another, always at the same time as an end, and never simply as a means".
- example: you don't just plagiarize somebody else's thesis counting on being able to apologize if caught... it's not about it working or not, it's just WRONG!

Q & A

[http://www.aleax.it/
oscon12_fop.pdf](http://www.aleax.it/oscon12_fop.pdf)

