Technical documentation

is a back-up

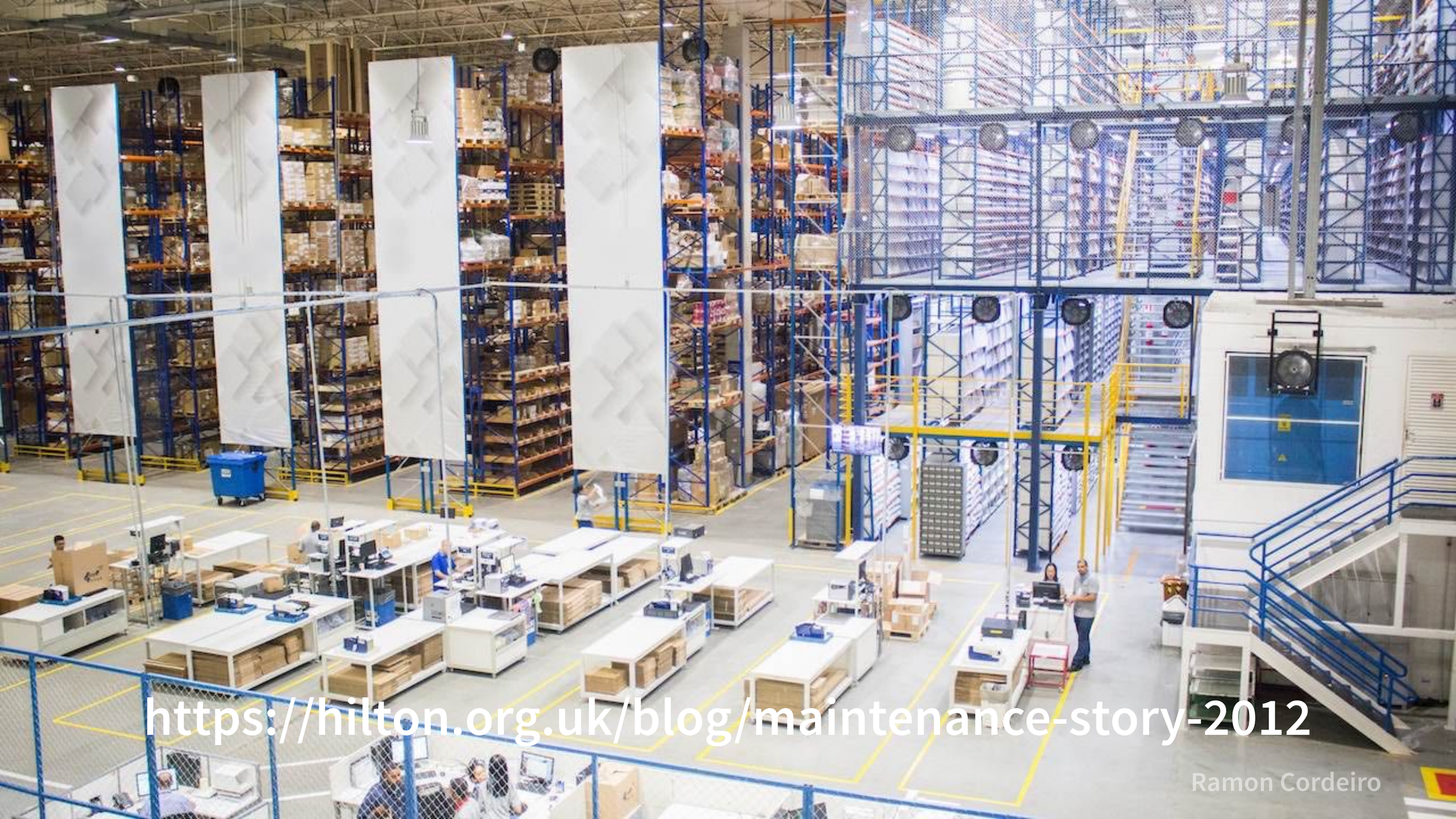
(so make sure it works)

@PeterHilton

http://hilton.org.uk/

https://hilton.org.uk/blog/maintenance-story-1997

https://hilton.org.uk/blog/maintenance-story-2006



Ramon Cordeiro

Three kinds of legacy code

Good code

but…

old technology

Code written by…

someone else

(a.k.a. bad code)

Complex code

because of…

10 years' built-up business rules

Three kinds of technical documentation

Extensive tech documentation.

Quality system records, e.g. code review forms

No documentation at all. None.

(not even code comments 😢)

Minimal docs: data dictionary, operations guides

(and code comments 😀)

Three lessons learned

Generous budget

Documentation was a backup

Bad code, original team not available

No backup was a problem.

Planning for maintainability

Minimum viable docs FTW!

Lessons learned

We don’t like software documentation when it’s:

1. Time-consuming
2. Hard
3. Wrong
4. Messy
5. Too long
6. Missing
7. Unversioned
8. Incomplete
9. Neglected
10. Boring

Technical documentation has many failure modes

… but no automated tests.

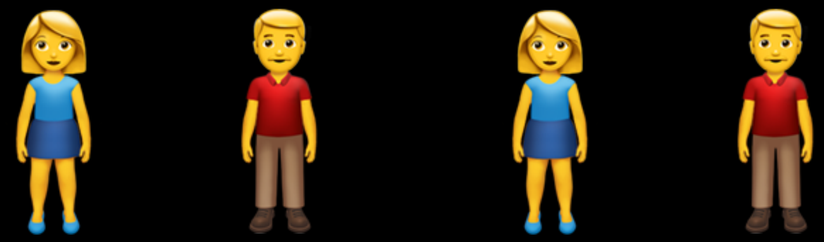
‘There is no documentation fairy’

Allan Kelly

Anthony Tran

Technical design docs don’t help maintainers

Software design
(a.k.a. development)

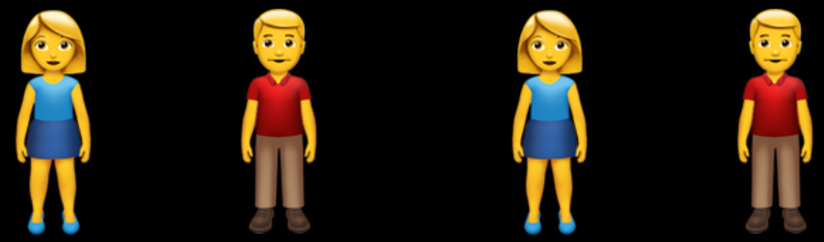


‘Technical specifications’

=

design choices and plans

Maintenance
(a.k.a. development)



? ? ?

=

how to understand and modify the code

Three rules for backups

Rule 1

You need backups for important things

(things go wrong)

Rule 2

Not using backups regularly might be a sign that things are going well

(but also a risk)

Rule 3

If you haven't tested your backups then they don't work

(things go wrong)

Three rules for documenting legacy systems

I had to invent a word for legacy systems…


YOUR LEGACY SYSTEM IS
UNDOCUMENTABLE!

Three rules for documenting legacy systems

Rule 1

Bad code:
needs all the docs
you can get
(like clues in a
murder mystery)

Rule 2

There's always
too much code

(so focus on
getting started)

Rule 3

Some legacy
systems are
undocumentable
(document the
process instead)

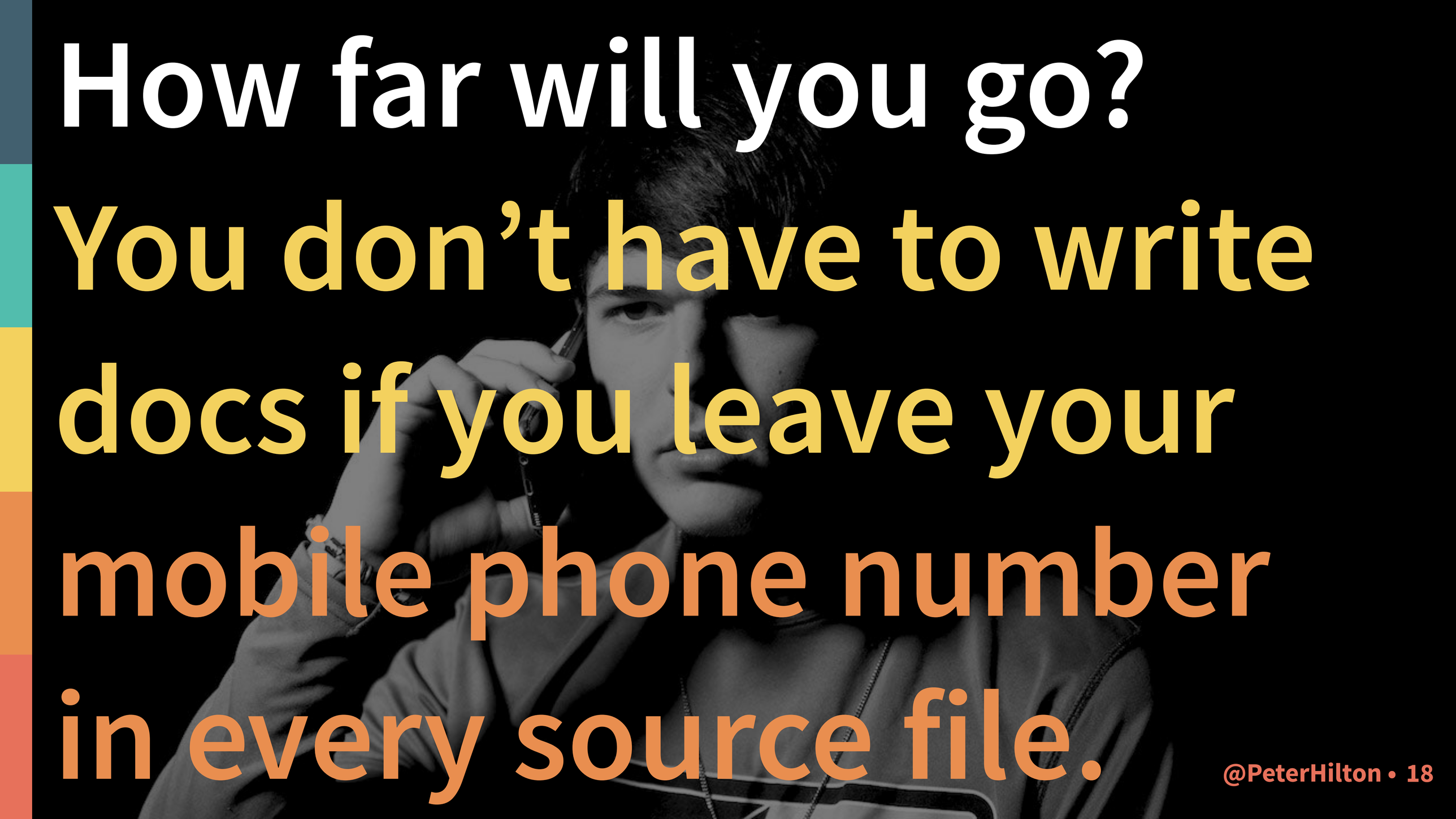
How to cheat at docs

How far will you go?

Chris Moore

How far will you go?

You don’t have to write docs if you leave your mobile phone number in every source file.



Improve the code instead

Better naming and cleaner code

(*fewer* code comments)

Automated builds

(*shorter* installation instructions)

Standard architectures

(no system diagrams)

The ideal documentation system

1. Works with your favourite IDE
2. Integrated with source code
3. Structured content
4. Version control
5. Minimal dependencies
6. Doesn’t require writing one yourself when you should be working

```
            beq scroll

            lda speed
            eor #$01
            sta speed

; ---------------------------------------
; Ordinary scroll... (he, he - beat me
; it is probably the shortest 1x1 scroll
; routine on the world ;)

scroll      lda roll
            sec
            sbc speed
            bpl nzero
            and #$07
            sta roll

            ldy #$00
rewrite     lda scrline+1,y
            sta scrline,y
            iny
            cpy #$27
            bne rewrite
```

Code comments 😬

Comments feature in almost all programming languages,
but remain an almost taboo topic.
Developers will go to bizarre lengths to avoid comments
and usually fail to write code so good they don’t need any.

Comments are not the enemy:
meetings are the enemy!

How to write good code comments

1. Try to write good code first.
2. Write a one-sentence comment.
3. Refactor the code (make it easier to understand).
4. Delete unnecessary comments.
5. Rewrite bad comments

   (all good writing requires rewriting)
6. Add detail where needed.
7. GOTO 1

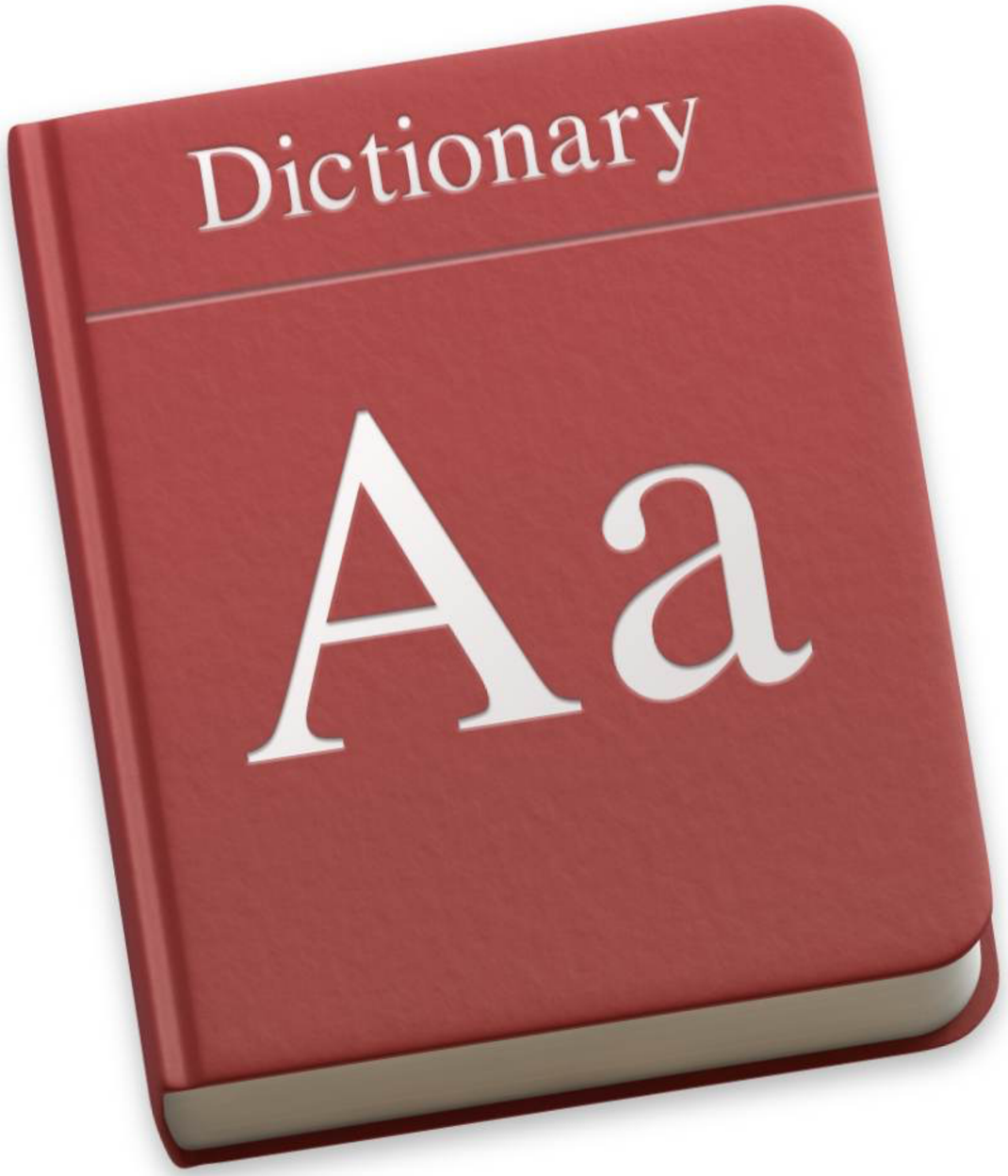
Play for Scala
(Manning, 2014)

Peter Hilton
Erik Bakker
Francisco Canedo

http://bit.ly/playscala2p

OPD

OTHER PEOPLE'S DOCUMENTATION


Dictionary
Aa

Use standards and frameworks

Standards/dictionaries are *other peoples' documentation.*

Frameworks are *other people's code*
and the successful ones have *good documentation.*

Not a perfect solution, because standards and frameworks are dependencies that you might not want.

MVD

MINIMUM VIABLE DOCUMENTATION

README.md

The minimal system is a README per component,
the short version of complete system documentation:

1. What it is
2. Purpose
3. Features
4. Usage/examples
5. Installation
6. Asking questions
7. Building from source
8. Authors/maintainers
9. How it works
10. Who it’s for

Special-purpose documentation for specific needs

There are many kinds of documentation that you usually don’t need. But sometimes you do.

- API reference
- Architecture diagram
- Contributor’s guide
- UML diagram
- Component inventory
- Entity relationship diagram
- Data dictionary
- Process model
- Business rules
- Architecture Decision Record

Hire technical writers

Docs as Code (Eric Holscher)

1. Use the same tools for docs that you use for code:
   issue tracker, version control, mark-up, review, tests, etc.
2. Use the development team's workflows.
3. Integrate docs development with the product team.
4. Adopt shared ownership of documentation.
5. Allow any team member to write the first draft.
   (Relevant for user documentation as well as system docs)

http://www.writethedocs.org/guide/docs-as-code/

Documentation techniques wish list

Architecture Decision Records (Michael Nygard)

1. Concisely record each architecturally significant decision.
2. Use sequential numbering and consistent titles, e.g.
   ‘ADR 1: Deployment on Ruby on Rails 3.0.10’
3. Document context, decision, status, and consequences.
4. Write-once: add new records to replace old ones.
5. Read all records to ‘load’ the current state
   (it’s like *event sourcing* for documentation)

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

Four kinds of documentation (Daniele Procida)

‘There isn’t one thing called documentation, there are four’

1. Tutorials
2. How-to guides
3. Explanation
4. Technical reference

‘… each of these kinds of documentation has only one job’

(it’s like *design patterns* for documentation)

https://www.divio.com/blog/documentation/

A manifesto for error reporting (David R. MacIver)

‘a manifesto for how errors should be reported by software to technical people whose responsibility it is to work with said software’

https://www.drmaciver.com/2013/03/a-rewritten-manifesto-for-error-reporting/

Summary

Technical documentation

1. Technical documentation is a backup for knowledge
   (and the rules for backups apply)
2. We need technical system documentation,
   (but we don’t usually need very much)
3. There are many new practices to replace old approaches;
   these practices have principles in common.

Technical documentation

Technical documentation is a backup for the team's knowledge (and the rules for backups apply)

For maintenance, we need technical system documentation (but we usually don't need very much)

There are many new practices to replace old approaches (these practices have principles in common)

more like this…

How to write maintainable code (training course)

Documentation for developers (training course)

@PeterHilton

http://hilton.org.uk/