coding {the} architecture

Modern Legacy Systems

Robert Annett - 2055

What is legacy?

Old, unstable, unsupported, Non-maintained, supplanted, monolithic, complex, obsolete, bad?

It's not well defined

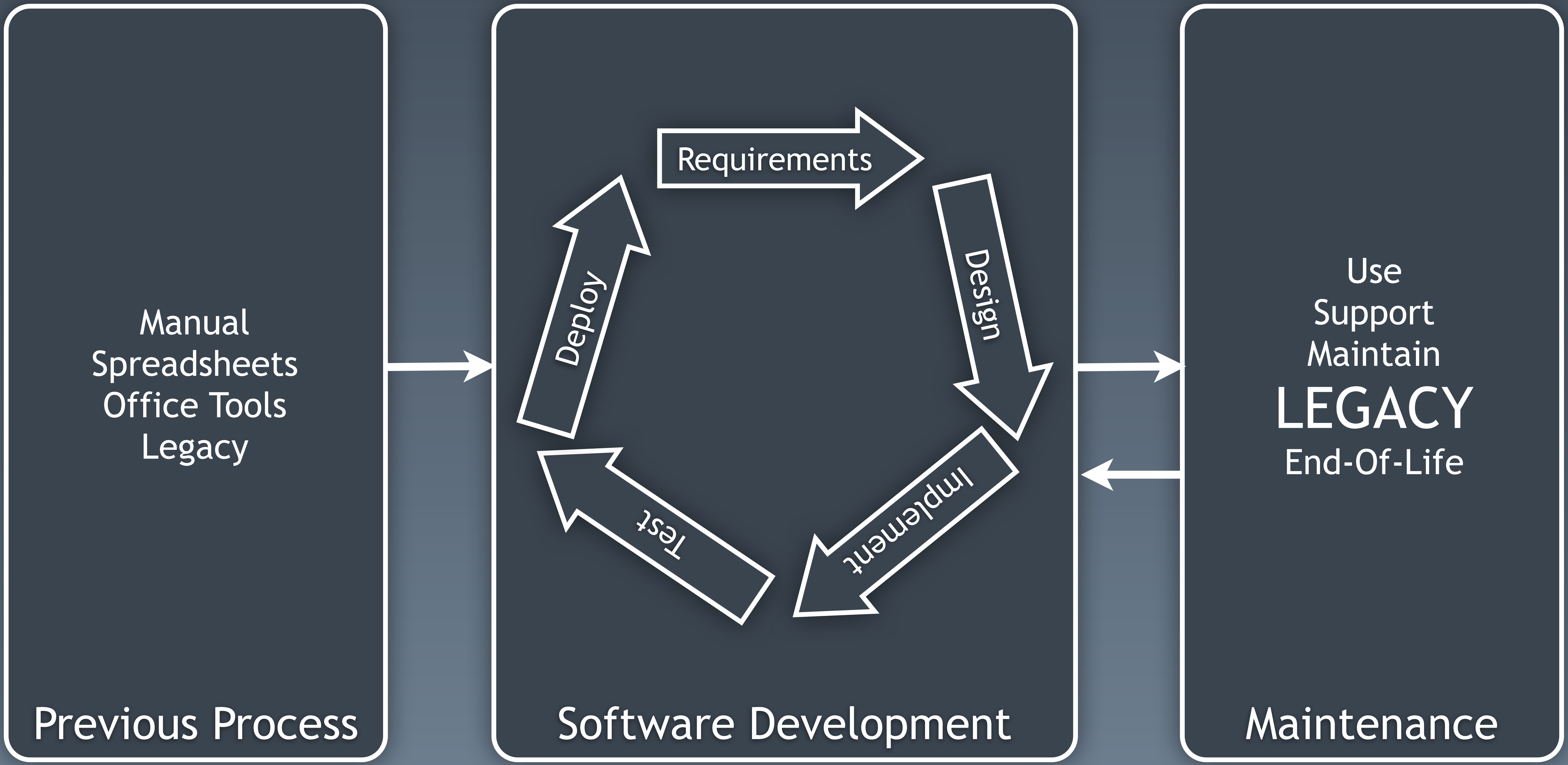
Often used as an insult

What is legacy?

"Legacy code" often differs from its suggested alternative by actually working and scaling.

Bjarne Stroustrup

Business Process Lifecycle



Manual
Spreadsheets
Office Tools
Legacy

Previous Process

Requirements

Design

Implement

Test

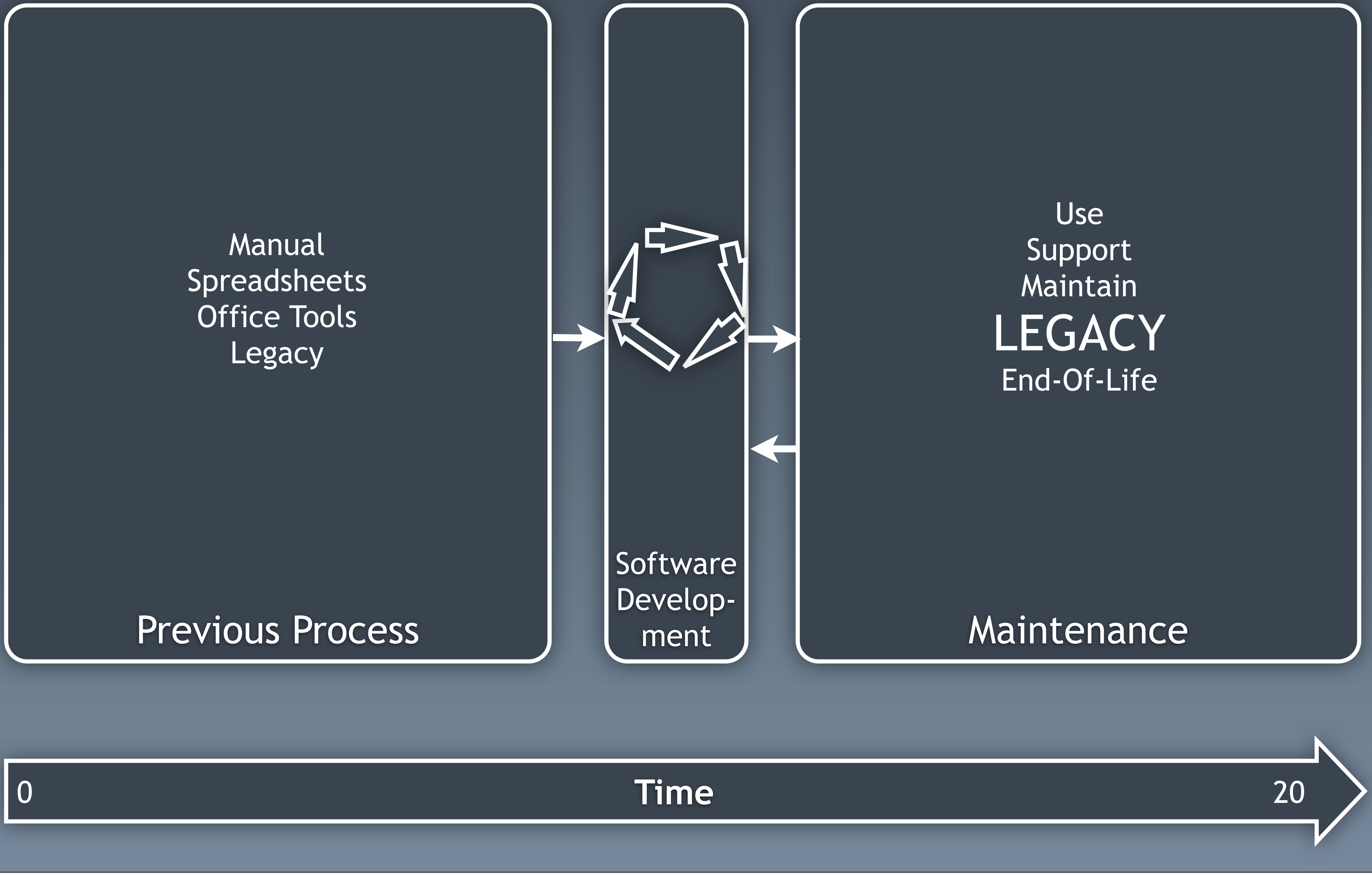
Deploy

Software Development

Use
Support
Maintain
LEGACY
End-Of-Life

Maintenance

Business Process Lifecycle



Manual
Spreadsheets
Office Tools
Legacy

Previous Process

Software Development

Use
Support
Maintain
LEGACY
End-Of-Life

Maintenance

0

Time

20

Is this legacy?

©Dave Ross

…no it's a museum piece

some real modern legacy systems

- Java 1.2 running under Solaris 8 on an Ultra Sparc III server using Oracle 8i, or...
- C# 1.0, Windows server 2000 using SQL Server 2000
- Objective C on NeXT server? J++/J#? VB6? FoxPro? Brokat?
- A combination of manual operations and 'office' tools

Scenario

- You work for a furniture company
- You get a promotion!
- You are now responsible for the warehouse inventory system
- No one has touch it for.... a while.
- It was written in 2003
- It’s a 3-tier architecture... etc etc...
- It basically works although everyone moans about it.
- What do you do?

Is this the kind of system you've seen?

What are the `common` issues?

No Documentation

usernames/passwor

licenses!

tion of static data

rking

over

Last release branch?

User lists

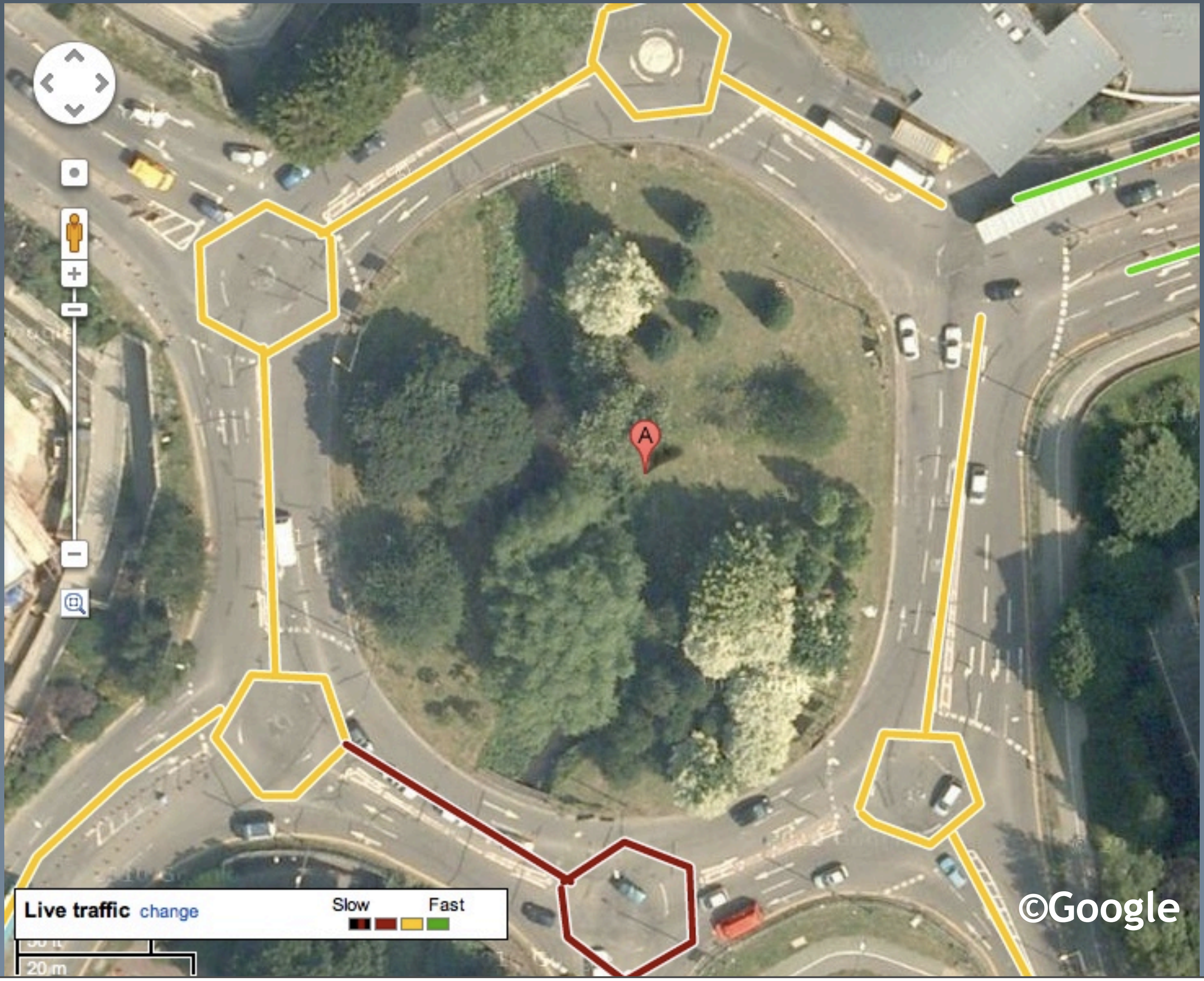
Network information

communications protocols

Manifesto for Agile Software Development, 2001

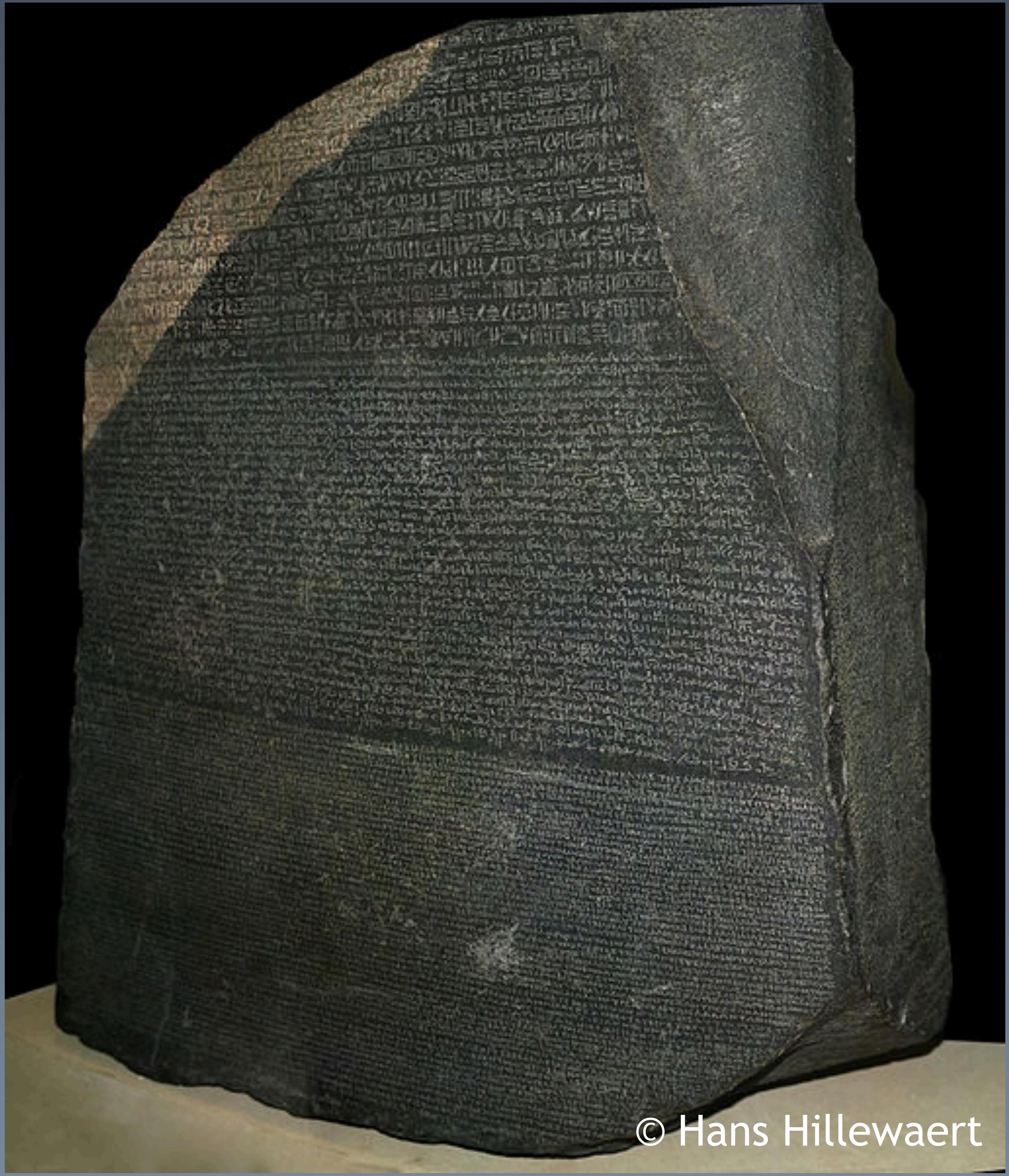
No Overarching Design

- Deliberate/Agile Architecture
- Organic Growth



©Google

Lost Knowledge



© Hans Hillewaert

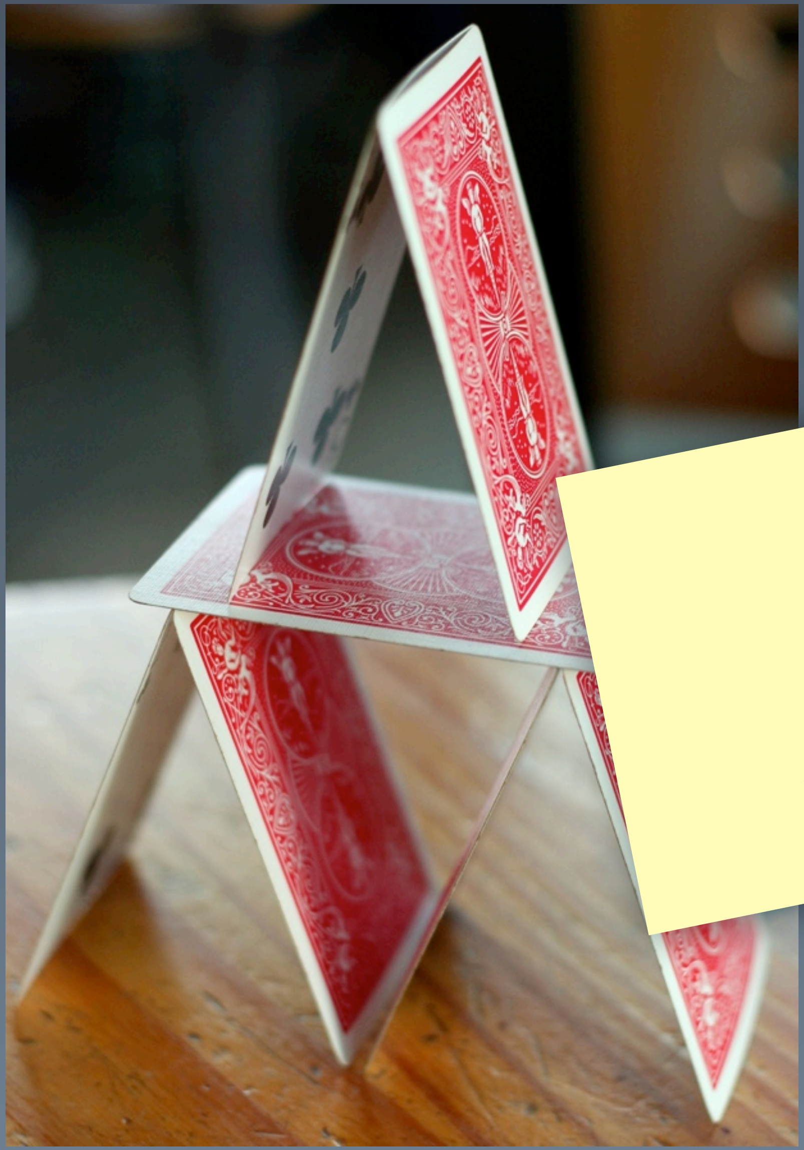
Hidden Knowledge

Application Voodoo!

Unused Functionality

©Ellin Beltz

Fragility



Real or imagined?

Coupling

tight

loose

Cohesion!

Zombie Technologies

- Unsupported or...
- dead and buried
- Major changes in APIs
- or just changes in style and fashion

Licensing

> If a company has SQLServer on VMware, licensed on a per core basis that can be dynamically scheduled across hosts - all machines need licenses.

Sean Robinson, License Dashboard quoted in Computing

Regulation

Personal data processed for any purpose or purposes shall not be kept for longer than is necessary for that purpose or those purposes.

Data Protection Act and Information Commissioner

Politics

- Who owns the project?
- Who uses the project?
- Who pays for the project?
- Who gets fired if it fails?
- Whose job is at risk... if successful?
- No-one likes changes

Except developers!

It’s not all bad!

- A legacy system is a successful system
- You can have a large impact
- You have real users you can talk to
- You can learn a lot about the business
- It’s important but not trendy so you’ll probably get paid well...
- The bar may be low!

what are my strategies?

Ignore it?

Investigation?

Maintenance

Sweat those assets!

Upgrade

Don't upgrade and abandon.

Migrate

The business has NFRs as well!

Incremental Improvement

- Functional additions
- Probably combined with upgrading
- …or “just stick it in”…

Beware technical debt

Replacement/Implementation

- Completely re-write the system
- Introduce a new architecture and technology
- You still need to maintain your legacy system while doing a re-write!
- Your users AND operations staff will require a lot of retraining

Is it better or just cooler?

you can *mix-and-match* the strategies!

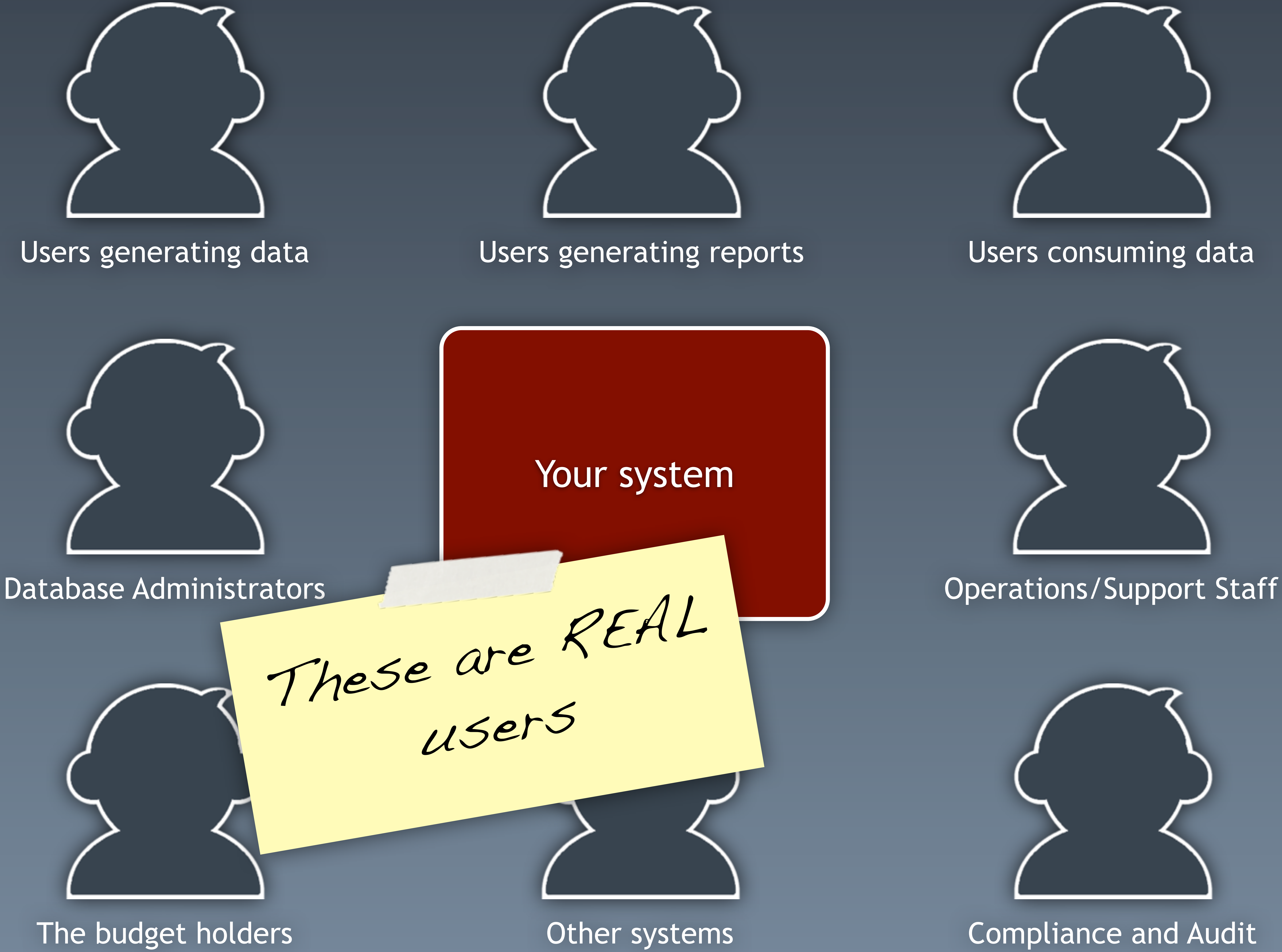
Don't just default to re-write

executing your

Strategy

locate the

stakeholders



Users generating data

Users generating reports

Users consuming data

Database Administrators

Your system

Operations/Support Staff

The budget holders

Other systems

Compliance and Audit

These are REAL users

draw a diagram of the system

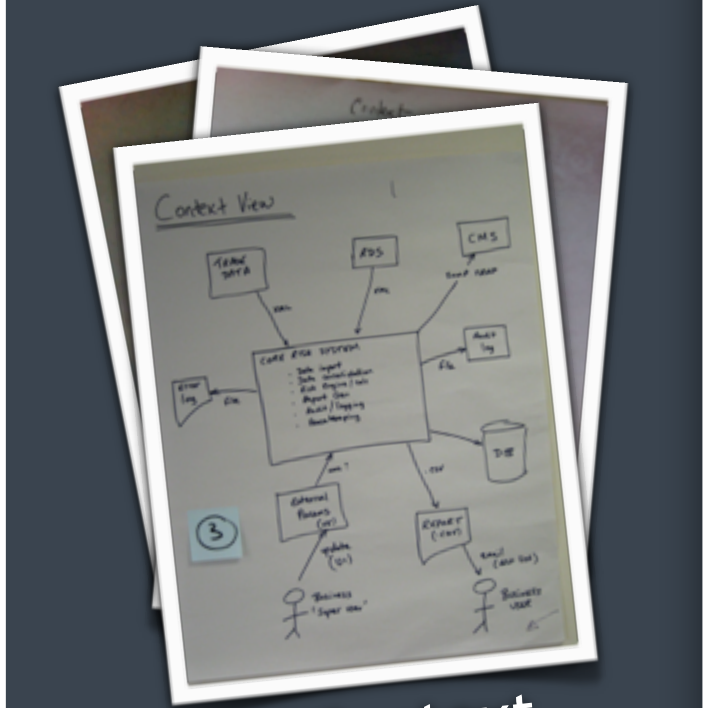
coding (the) architecture
Software Architecture for Developers
A practical and pragmatic guide to software architecture
Technical leadership
Collaboration, consistency and preventing chaos
Communication
Effective sketches and lightweight documentation
Just enough
Structure, shared vision, risks and firm foundations
Simon Brown

Published incrementally

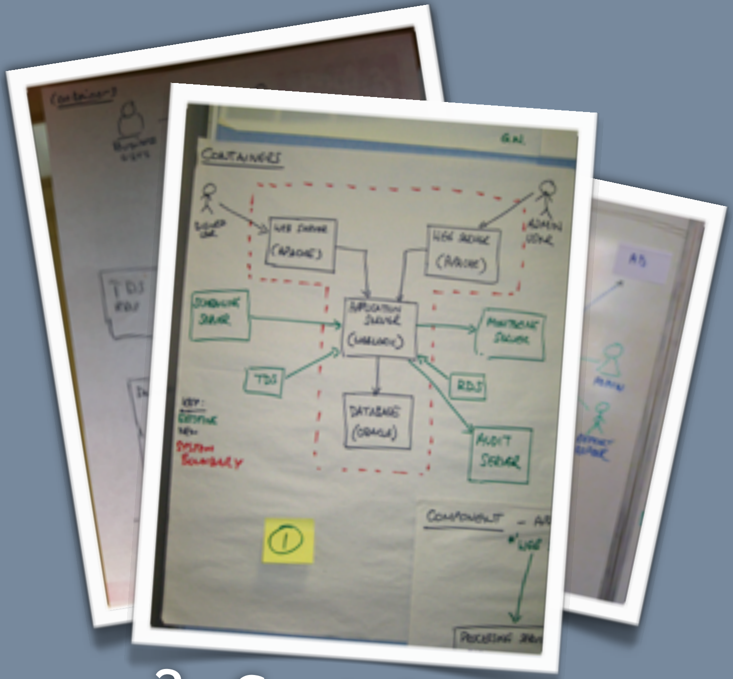
Variable pricing

Buy now and get free updates

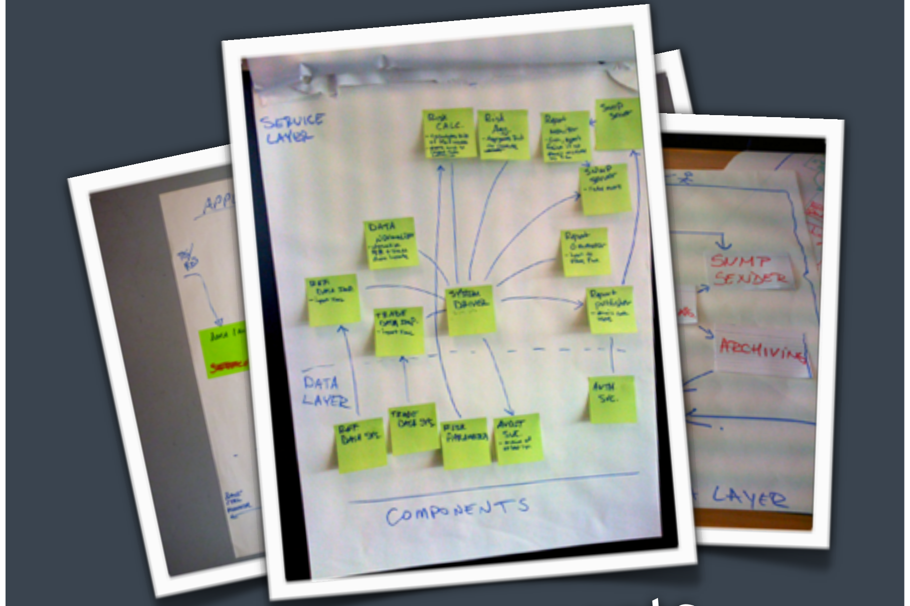
leanpub.com



1. Context



2. Containers



3. Components

This only covers the static structure (runtime, infrastructure, deployment, etc are also important)

… and, optionally,
4. Classes

C4

- Context
- Containers
- Components
- Classes

Thinking inside the box

Context

- What exists in the system?
- Who is using it? (users, actors, roles, personas, etc)
- How does it fit into the existing IT environment?

Containers

- What are the high-level technology decisions?
- How do containers communicate with one another?
- As a developer, where do I need to write code?

Components

- What components/services is the system made up of?
- Is it clear how the system works at a high-level?
- Do all components have a home (a container)?

a quick example

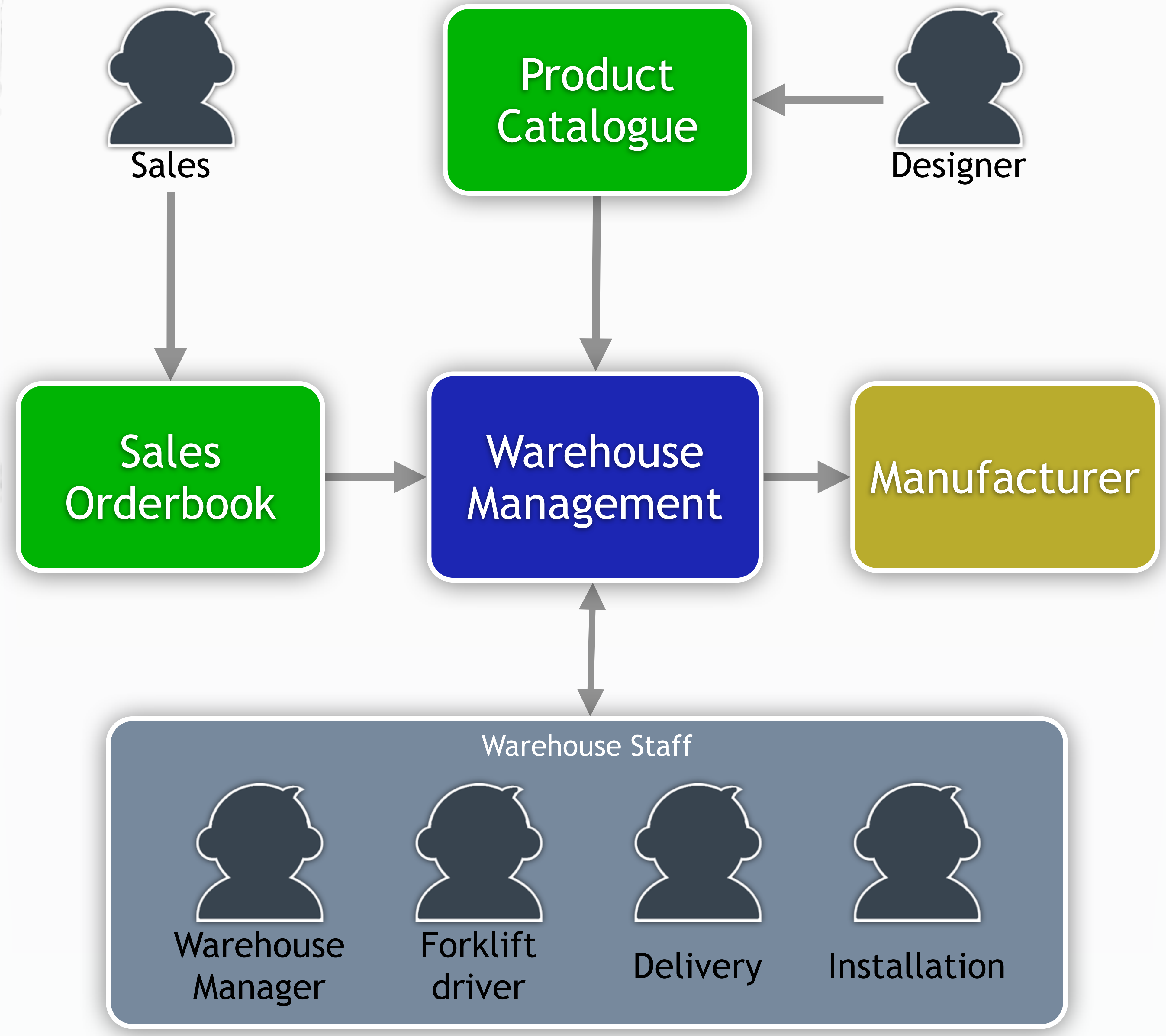
Context:

What

do we have?

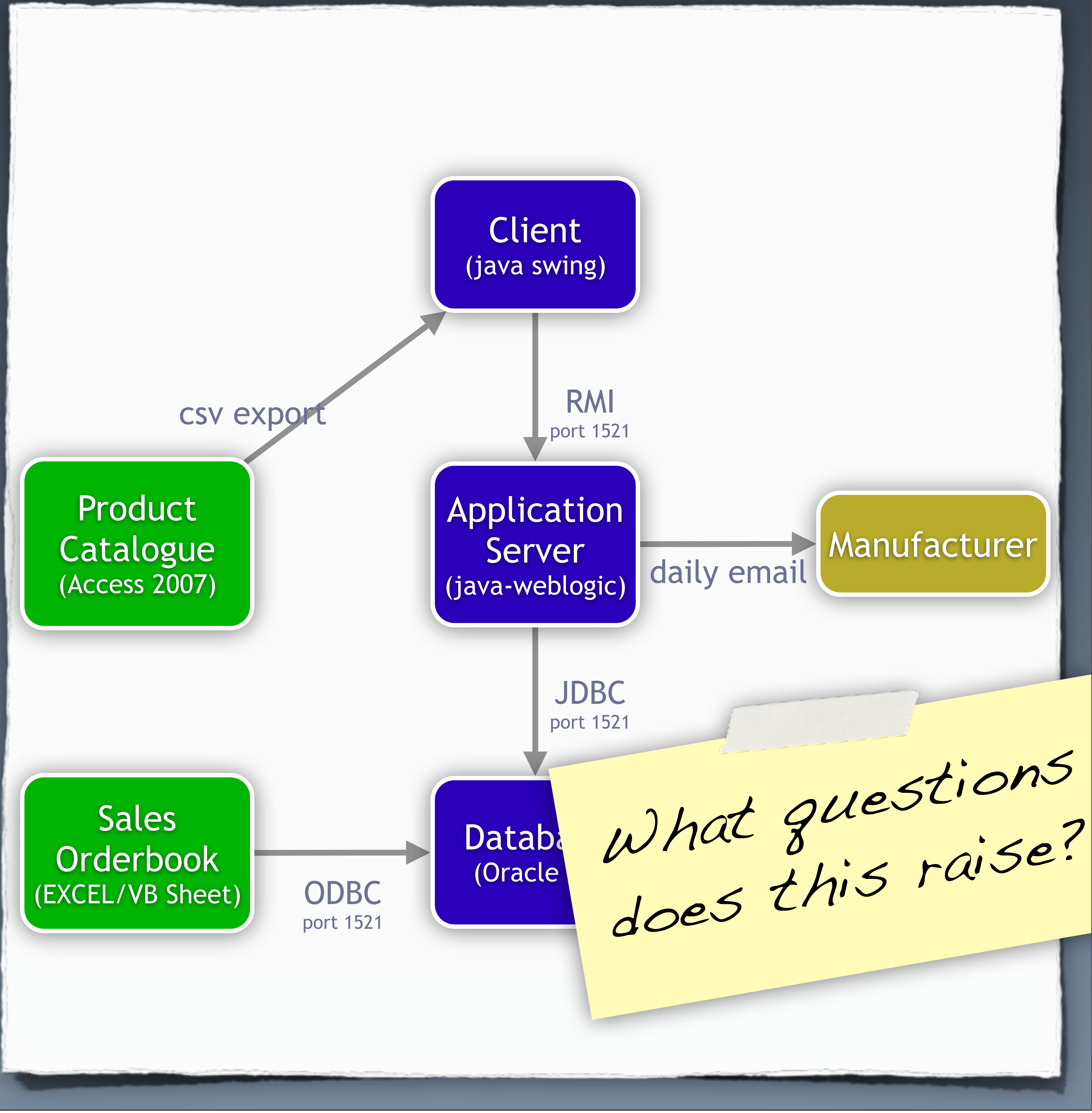
Who

is using it?



What **containers** is the system made up of?

How do they communicate?



What questions does this raise?

deeper analysis

Deeper Analysis (suggestions)

- Component Diagrams
- Class Diagrams (possibly)
- DB Schemas
- Configuration
- Stored data sizings
- Network settings and traffic
- The source code

Beware that what is running may differ from your source control...

beware the

Dark Side

of the source

we now know what

we've got

let's safely make some

changes

virtualisation

is your friend

Preparation

- Re-create your PRODUCTION system as a test system.
- Run tests and metrics for the copy
- Make sure you can swap in and out the virtualised containers at a high level
- Try to build and redeploy the components within containers.

Be confident you can deploy and rollback.

Preparation

- Create some systems tests
  - Make them realistic uses of the system.
  - Report runs you can compare
  - Consider ‘test’ data in the real system...
- Build, Virtualise and Deploy Production

It’s paranoid but we can now deploy without fear at last we can

modify

the system!

Stabilisation

- Worth doing even if replacing
- Data Cleanse
- Data Archive
- Remove unused components
- Shift resources
- Tune applications/DBs
- Code changes if required

A stable system proves your competence!

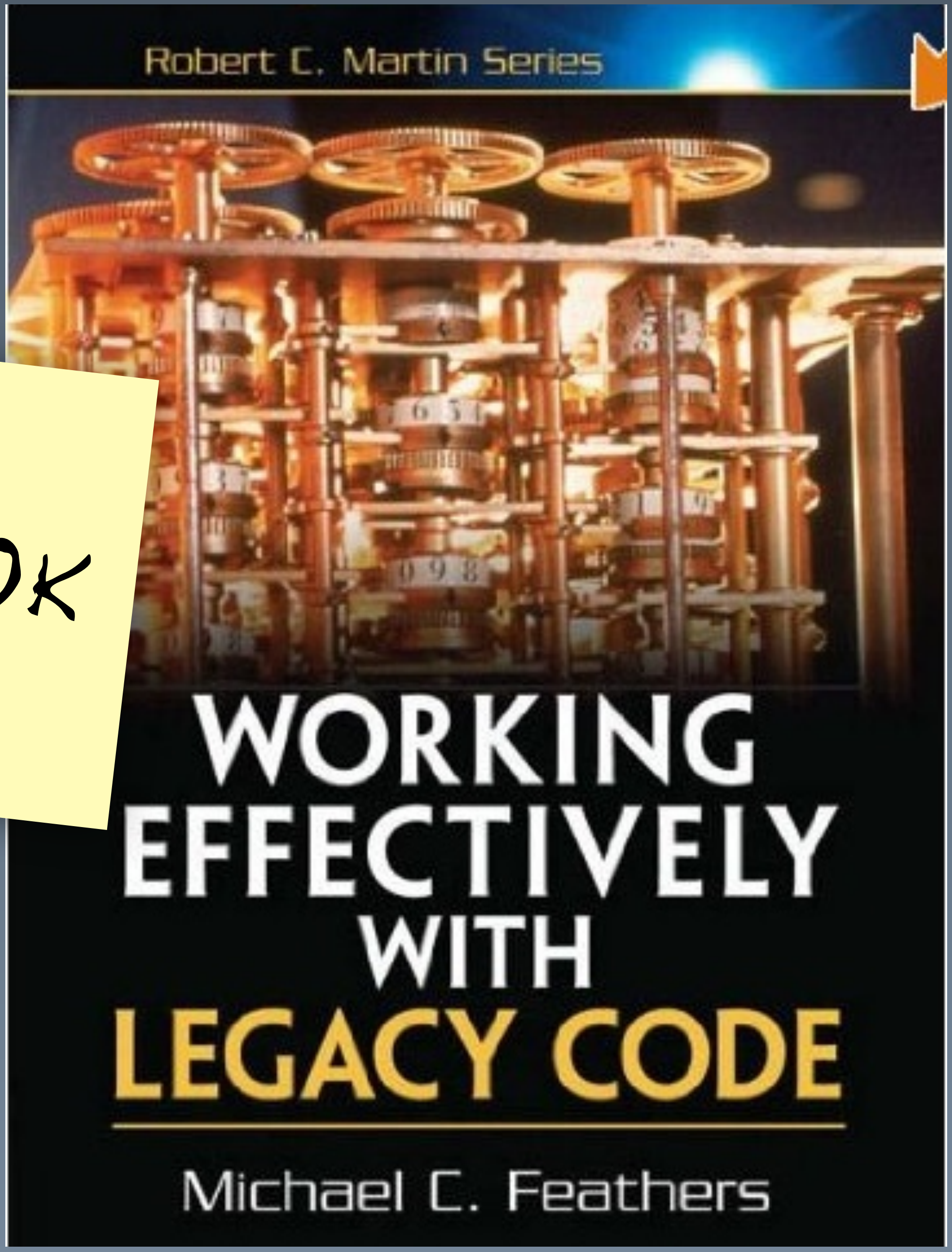
Upgrading/Migration Third Party

- Third party products are ALWAYS harder to upgrade/migrate than the sales pitch says.
- Functional and non-functional behaviour WILL change.
- Be prepared to rollback if and when necessary.
- This should be methodical after all the preparation

Legacy Code Modification

- Formatting
  - Create a new baseline
- Refactor as you go
  - Break dependencies... and...

BUY THIS BOOK



Strategic rewrites take longer to implement than planned. Be prepared to maintain the legacy for a while.

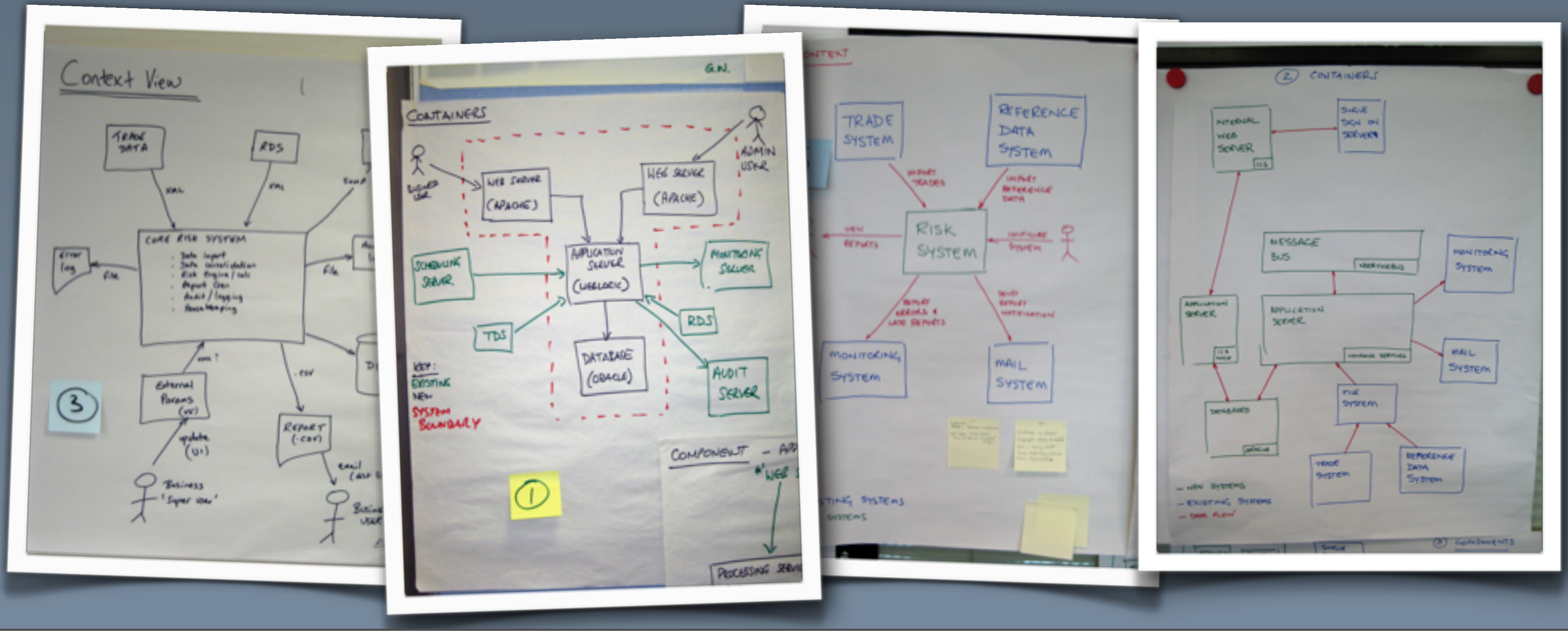
how to leave a

good

legacy

Documentation is not evil!

- Basic access information
- Build and deploy instructions
- Code AND config
- Asset Register
- BCP/Failover/Backup Plans
- ARCHITECTURE DIAGRAM



Tests

- System
- Integration
- Functional
- Non-functional
- UAT
- Accessibility...
- Automate and leave documentation

There's more to testing than xUnit

System Design

- High level design
- Architecture doesn’t necessitate a 200 page document

best practice
and
right tool for the job
but…

Technology Consistency and Scope

Careful of new technologies

It's about trade offs...

Thanks

and leave a good legacy!

robert.annett@codingthearchitecture.com

@robert_annett on Twitter

robert.annett@codingthearchitecture.com

@robert_annett on Twitter

simon.brown@codingthearchitecture.com

@simonbrown on Twitter

Coding the Architecture

Search

Software architecture for developers

coding {the} architecture

Home | Blog | Book | Presentations | Videos | Training | Coaching & Consulting | Contact

Software Architecture for Developers
Find out what the role of a software architect is all about in our 2-day software architecture training course...

The Frustrated Architect

The slides and video from my 'The Frustrated Architect' talk at Skills Matter in November 2011 are available to view online.

50+ essays

Read our (in-progress) book; a collection of essays that together form a practical and pragmatic guide to software architecture.

Speaking

Here are some of the events that we spoke at in 2011 and are speaking at in 2012. The slides are available to download.

Writing

Software development plus training and consulting