Case Studies

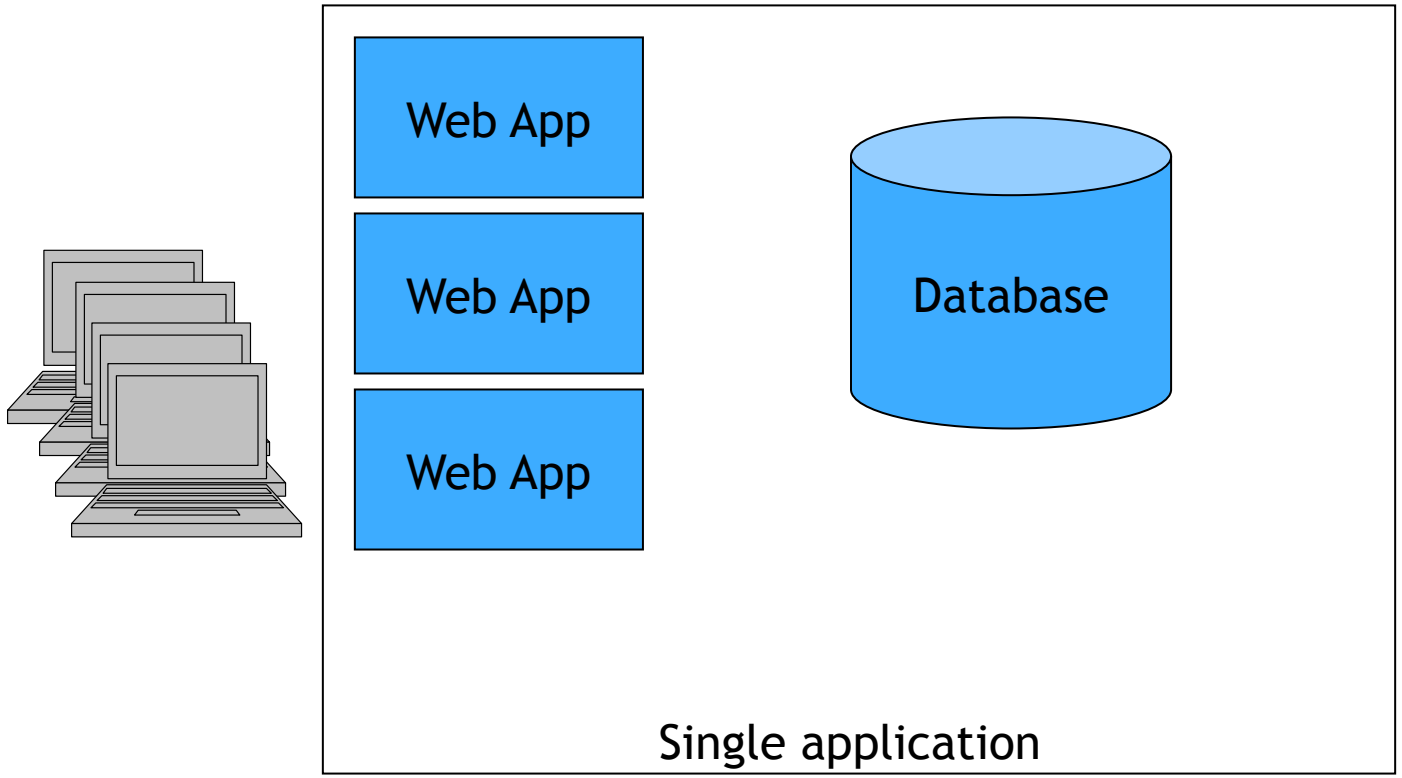
Oxford University

Software Engineering Programme

Dec 2013

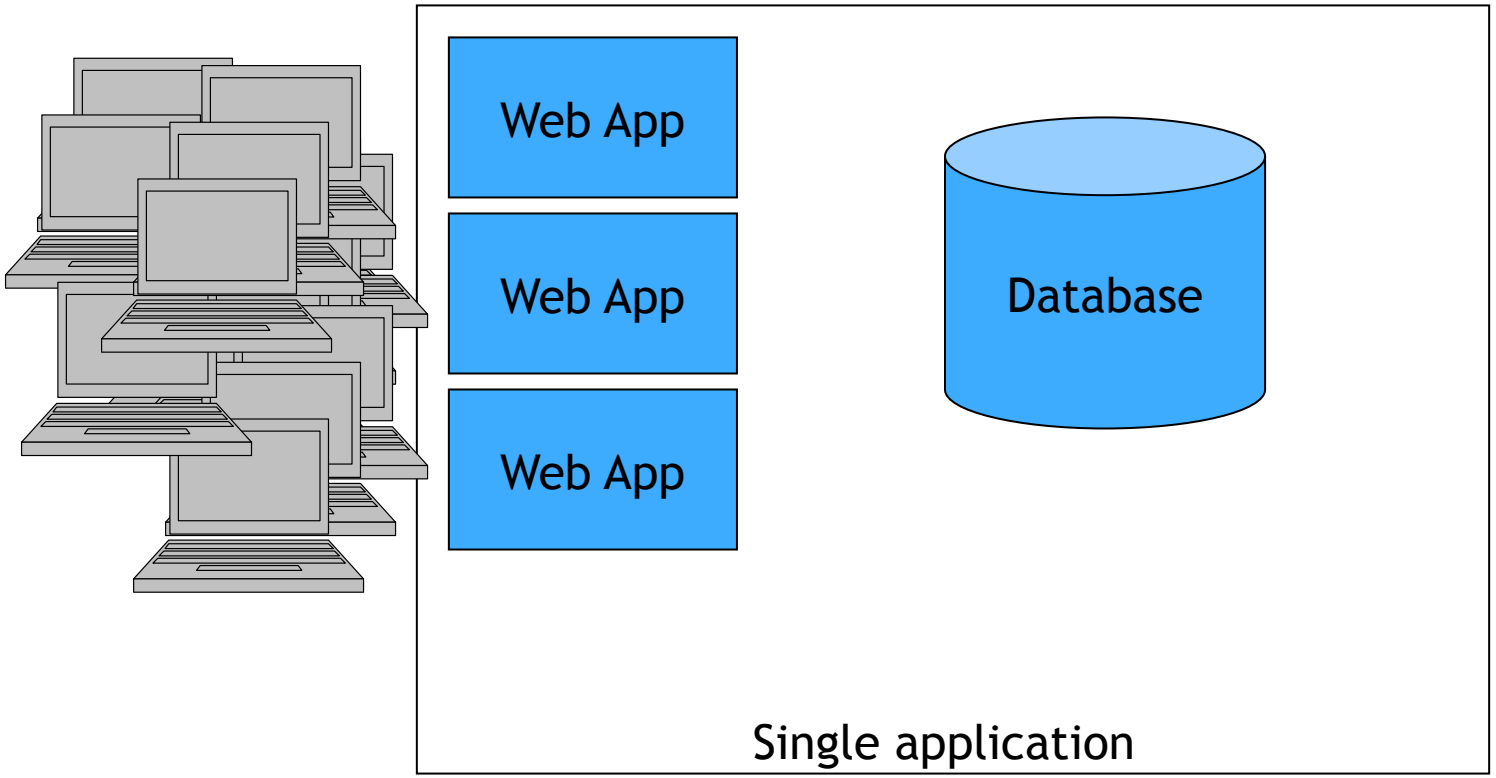
Source: Interview with Werner Vogels, ACM Queue

“Obidos”

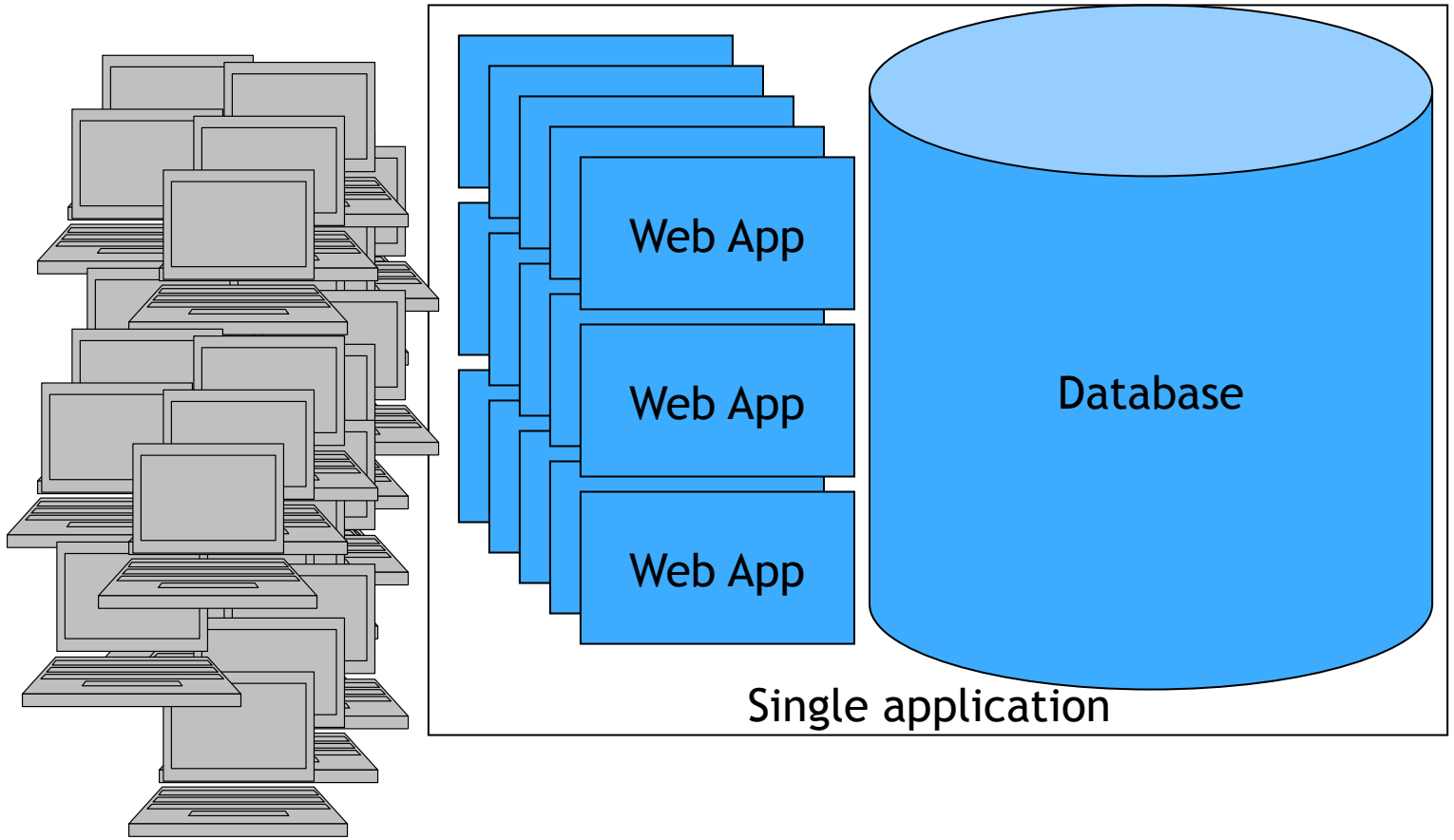


But it was Successful!

Internet Scale Up



… to bursting point



Problems

- Too many complex pieces of software in a single system
- No evolution possible
- Need to scale independently
  - Parts sharing resources with other unknown code paths
- No isolation
- No clear ***ownership***

Database scaling

- Databases a shared resource
- Hard to scale-out
- Front-end and backend shared by
  - Too many teams
  - Too many processes

A new model

- In 2001 decided on a new approach
- SOA based – even before the term was in common usage
- Encapsulating the data with the business logic that operates on the data
- Only access through a published service interface
- No direct database access is allowed from outside the service
- No data sharing among the services.

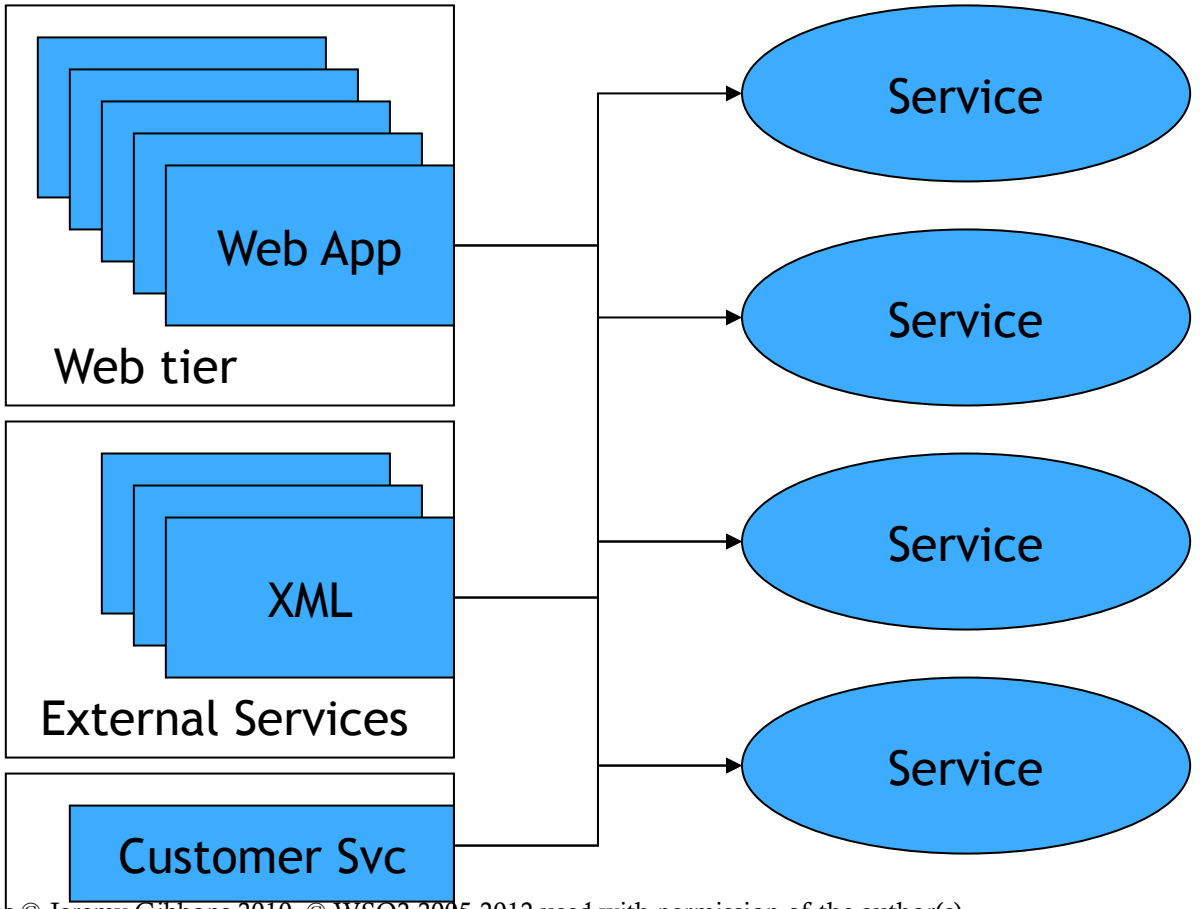
Growth

- Amazon services in the hundreds
- A typical visit to the homepage may include calls to 100 services
- Caching reduces the actual network traffic
- Fully distributed, decentralized
- The web servers are just one client into the service fabric

Matched by business growth

- Amazon is supporting many new businesses
- Books, CDs, Electronics, Toys, Tools and Hardware,…
- Plus millions of independent retailers sharing the Amazon platform

New architecture



Lessons learnt

- Isolation
  - Service Orientation promotes ownership and control
- Scalability
  - By preventing direct database access, can scale the services without affecting clients
- Need a common service-access mechanism
  - Aggregation
  - Routing
  - Tracking

Organization

- "Each service has a team associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it… ***You build it, you run it***"
  Werner Vogels, CTO, Amazon

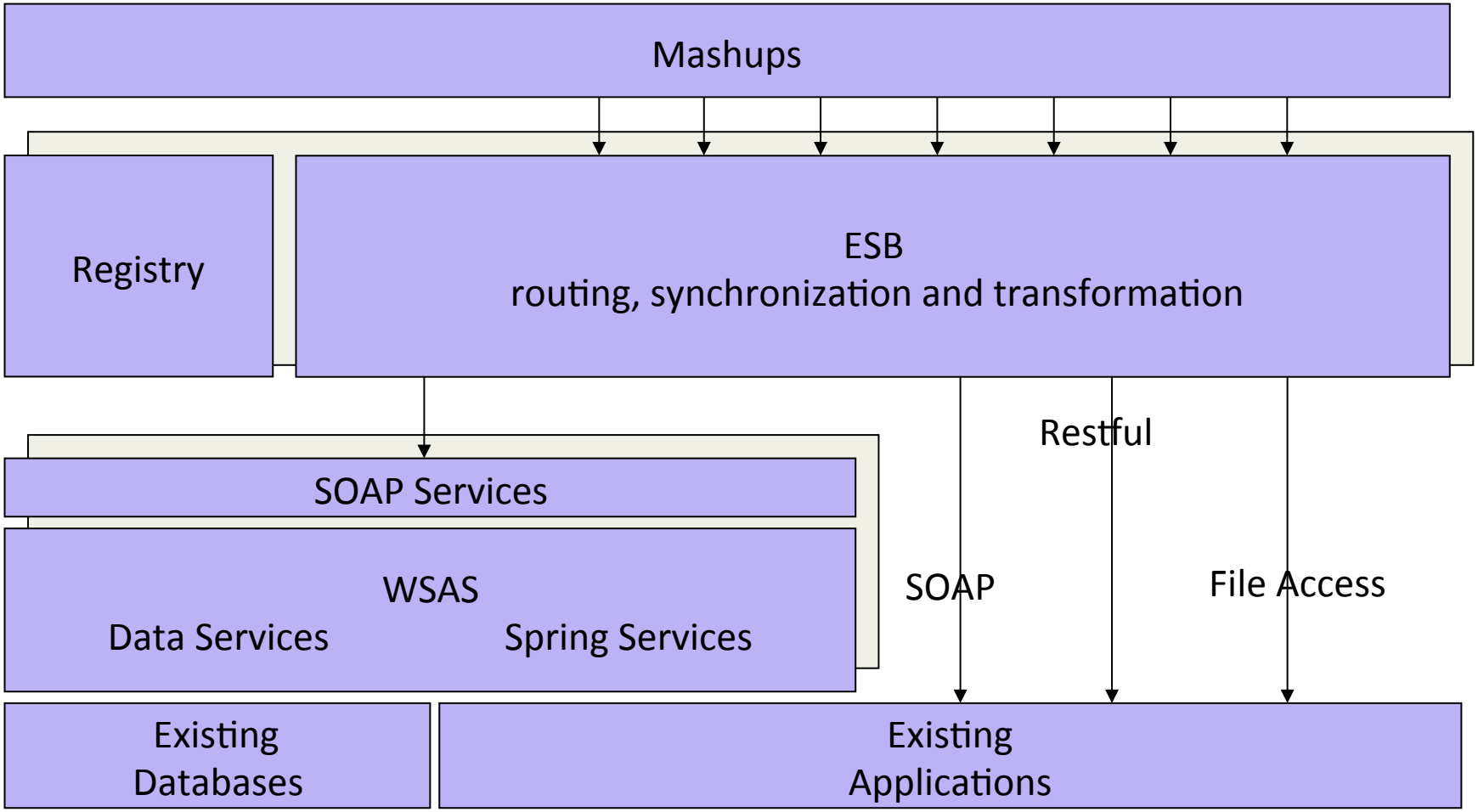
  - Promotes Customer Focus and Innovation
  - Gives developers direct access to customers
  - And experience of how their code performs

Integration at the glass

Concur

- Concur is an online expense management company
  - >$200m revenue
  - Multiple legacy systems:
    - Customer Relationship Management
    - ERP
    - Sales Force Automation
    - In house HR employee application
  - Main requirement – enable better reporting across applications
    - Internal project only – not in the direct flow of external customer systems
  - Needed an approach that supported:
    - Iterative development
    - Support changes to the underlying systems
    - Flexible

Architecture

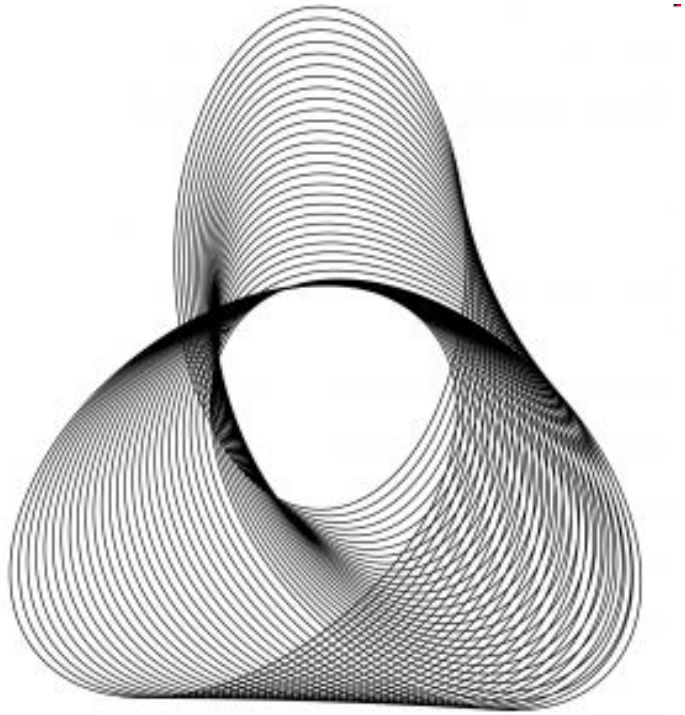


Bug Tracking / ITIL Ticket / CRM / SFA / HR / (10 systems in all and growing)

Technical details

- Everything deployed on Windows 2003 running on VMWare
- Internal systems so limited security
  - Basic authentication
  - Some use of digital signature
- Running in a blade server to simplify test and scaling
  - Currently Hot/Cold but moving to Hot/Hot
- ~75,000 transactions a day
  - 95% SOAP, 5% Restful at this point
- WSDLs and Schema's stored in WSO2 Registry
  - Embedded in the ESB
- Currently 18 services across 10 backends with 120 operations
  - Growing
- Looking at moving to a more event-based approach in the future

Iterative development



Project Approach

- Planned for iterative development over phases
- Staff self-educated on SOA and looked at Open Source systems before talking to vendors
- One week "kickstart" education and POC session
  - Built a data synchronization application
- Proof to the business:
  - Concur built a prototype that offered real value to executives:
    - Single customer view mashup – pulled open CRM tickets, ERP and CRM data.
    - The demo was an "instant hit" – gaining an executive sponsor
- Team identified re-usable services
  - Put extra effort into the design
- Several refactoring iterations

Benefits

- Lower cost of licenses/users on SaaS systems
  - Previously were using licenses for occasional users
- Intermittent users were being trained on systems that they rarely used – the new mashups replaced this requirement
- The SOA design has allowed incremental replacement of some legacy systems
  - Existing test plans for Sarbanes-Oxley could be re-used
- Open source meant that a POC could prove the benefits to the business without upfront expenditure

Lessons Learnt

- Keep it Simple
- In-house expertise has paid off
  - Steeper learning curve but
  - Better technology selection
  - Lower overall cost
  - More agility
- Use of open source projects has
  - Reduced cost
  - Been more flexible
  - Given better access to the community and developers

Business to Government

OIO SOI

OIO SOI

- Danish Government wanted to simplify electronic business
  - Especially for Business-to-Government (B2G)
- Potential savings of 630m Euros by digitalizing business
- Requirements
  - Reliable delivery
  - Secure – encrypted and signed messages
  - Support small businesses

OIO SOI

- Several aspects
  - A registry for service lookup
  - A profile of transport protocols
  - Open Source toolkits for Java and .NET
  - A reference implementation of a message handler
  - A legal framework
- Some existing framework
  - A nationwide digital certificate framework
  - A standard XML syntax for invoices and orders (UBL2)

Registry

- A profile of OASIS UDDI v3.0
- A central registry run by the Danish Government
  - https://publish.uddi.ehandel.gov.dk:12443/registry/uddi/web
- Designed to be used by electronic clients
  - Not to be browsed by humans!
- Requires a Danish Certified Certificate to publish

RASP
Reliable Asynchronous Secure Profile

- A profile of
  - SOAP 1.2
  - WS-Security 1.1
  - WS-ReliableMessaging 1.0
  - WS-Addressing
- Two bindings: HTTP and SMTP

- Why SMTP?
  - To allow small businesses to communicate
  - No requirement to host a web server
    - No 24x7 operation
    - No firewall configuration
  - Only an email address

RASP capabilities

- Authentication
- Confidentiality
- Integrity
- Non-repudiation / proof of delivery
- Support for intermediaries
- Asynchronisity

Interoperability

- RASP includes libraries for both
  - .NET – based on WCF 3.0
  - Java – based on Apache Axis2
- Defined a set of tests and run using a continuous test environment
- Biggest problems were found with
  - WSRM and SMTP

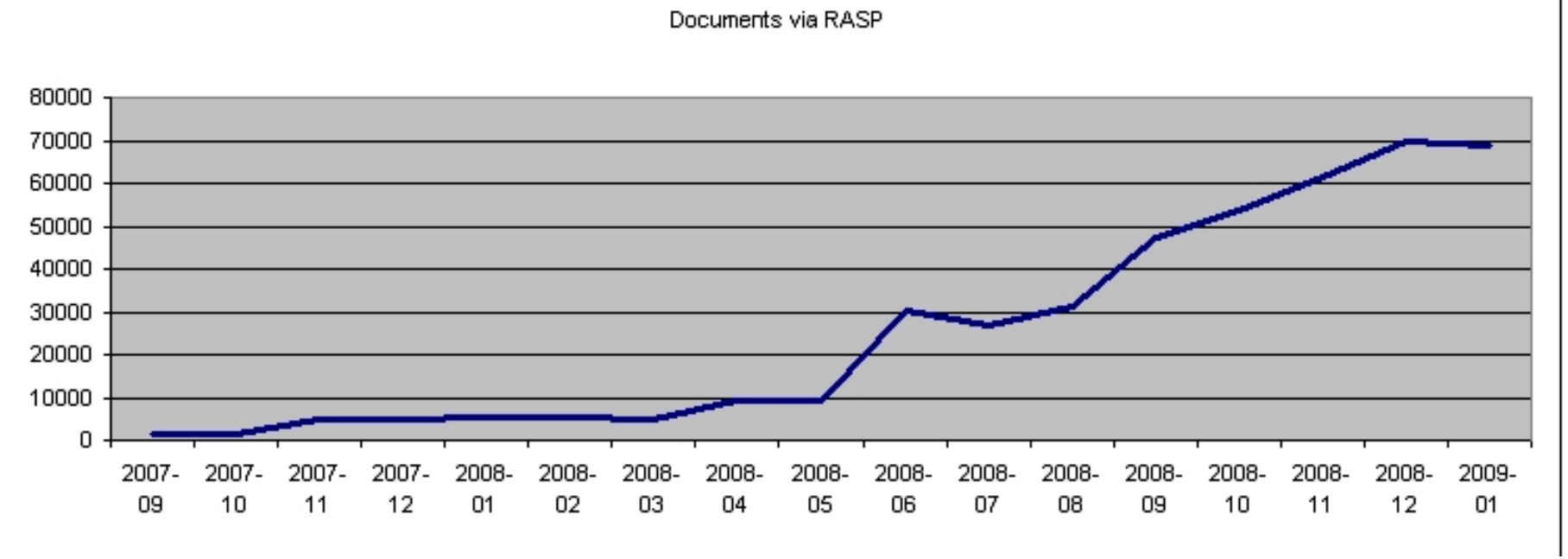
NITA Interop

No RM, No Sec		HTTP		SMTP	
Scenario	Description	Axis2->.NET	.NET->Axis2	Axis2->.NET	.NET->Axis2
1	Basic success	Yes	Yes	Yes	Yes
2	Resending	NA	NA	NA	NA
3	Timeout	NA	NA	NA	NA
4	Incomplete stack fault	NA	NA	NA	NA
5	Clock Skew	NA	NA	NA	NA
6	Custom Headers	Yes	Yes	Yes	Yes
7	Mail Binding validity	NA	NA		
RM Only		**HTTP**		**SMTP**	
Scenario	Description	Axis2->.NET	.NET->Axis2	Axis2->.NET	.NET->Axis2
1	Basic success	Yes	Yes	Yes	Yes
2	Resending	Yes	Yes	Yes	Yes
3	Timeout	Yes	Yes	Yes	Yes
4	Incomplete stack fault	Yes	Yes	Yes	Yes
5	Clock Skew	NA	NA	NA	NA
6	Custom Headers	Yes	Yes	Yes	Yes
7	Mail Binding validity	NA	NA		
Sec only		**HTTP**		**SMTP**	
Scenario	Description	Axis2->.NET	.NET->Axis2	Axis2->.NET	.NET->Axis2
1	Basic success	Yes	Yes	Yes	Yes
2	Resending	NA	NA	NA	NA
3	Timeout	NA	NA	NA	NA
4	Incomplete stack fault	Yes	Yes	Yes	Yes
5	Clock Skew	Yes	Yes	Yes	Yes
6	Custom Headers	Yes	Yes	Yes	Yes
7	Mail Binding validity	NA	NA		
RM+Sec		**HTTP**		**SMTP**	
Scenario	Description	Axis2->.NET	.NET->Axis2	Axis2->.NET	.NET->Axis2
1	Basic success	Yes	Yes	Yes	Yes
2	Resending	Yes	Yes	Yes	Yes
3	Timeout	Yes	Yes	Yes	Yes
4	Incomplete stack fault	Yes	Yes	Yes	Yes
5	Clock Skew	Yes	Yes	Yes	Yes
6	Custom Headers	Yes	Yes	Yes	Yes
7	Mail Binding validity	NA	NA		

Logical architecture

- This is logically a complete peer-to-peer architecture
  - With only a central registry
- Any company can talk to any other company
- Even those with only mail accounts
- Cannot track all the requests!

Results



18,500 companies sending invoices via RASP
Mandatory to send invoices to all government agencies
Scanning companies and a web gateway allow bridging

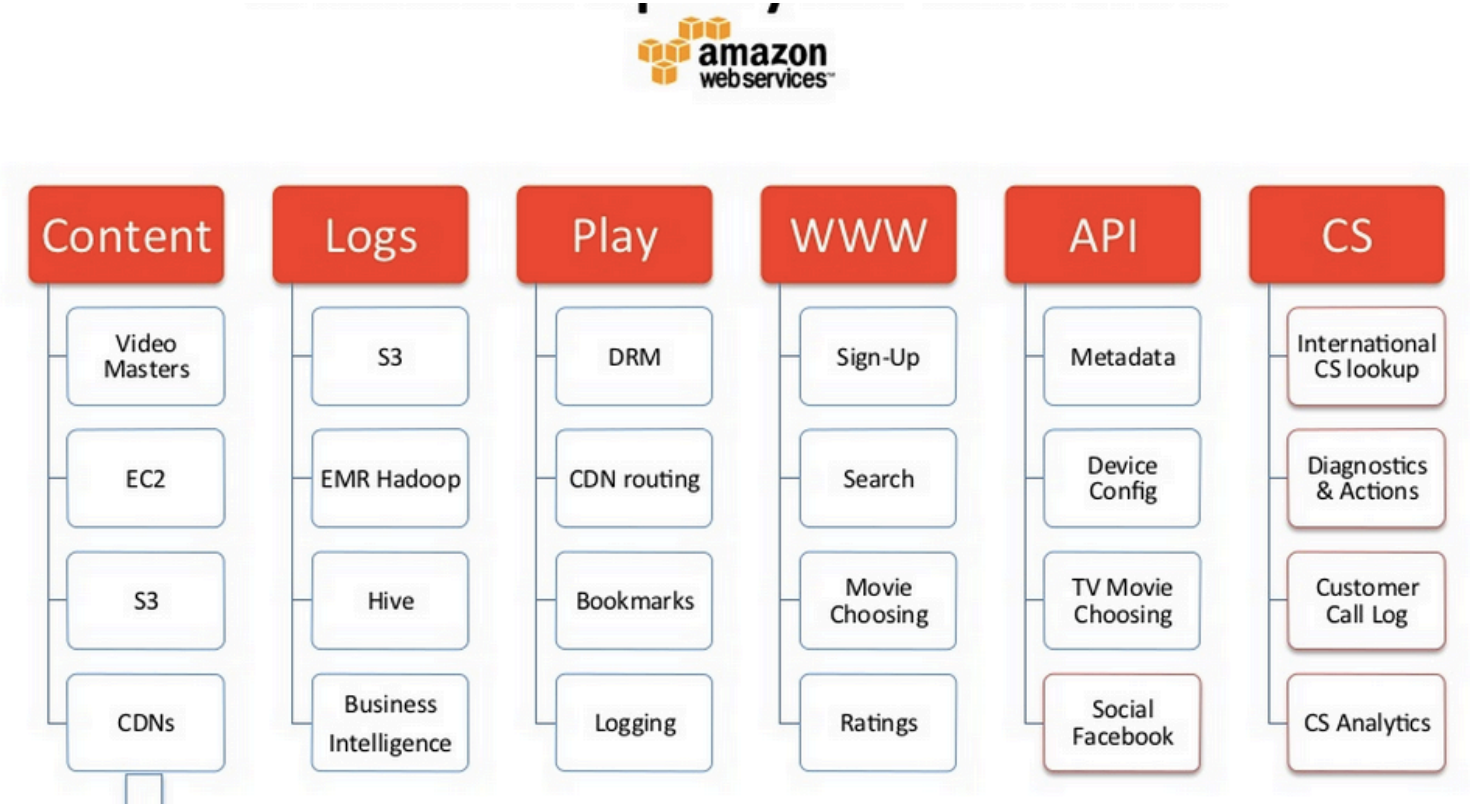
Lessons learnt

- SMTP in the real world is tricky
  - Spam filters can modify or drop messages
  - Our email accounts got shut down for “spamming”
    - i.e. sending many messages in a short time
  - Timeouts were too long for the RM system
  - We made mistakes layering SMTP and WS-Addressing
- Publishing interoperable reference implementations was a big win
  - Proved interoperability
  - Formed the basis for other implementations to test against
- The RASP team is now working on a European initiative:
  - PEPPOL http://peppol.eu
  - Trying to bring the same results across Europe

Netflix

- A REST and Cloud based SOA approach
- Continuous Delivery
- 100% Based in the cloud
- See excellent presentations from Adrian Cockcroft
  - e.g. http://www.slideshare.net/adrianco/global-netflix-platform

Netflix Deployed on AWS



Platform Services

- Discovery – service registry for “applications”
- Introspection – Entrypoints
- Cryptex – Dynamic security key management
- Geo – Geographic IP lookup
- Platformservice – Dynamic property configuration
- Localization – manage and lookup local translations
- Evcache – eccentric volatile (mem)cached
- Cassandra – Persistence
- Zookeeper - Coordination
- Various proxies – access to old datacenter stuff

The (in)famous Chaos Monkey

- Randomly kills machines
- Yes, production systems
- Proves that the system is resilient

Twitter Architecture

- Open Sourced their technology:
  - Finagle
  - http://twitter.github.io/finagle/
  - Called an RPC system, but completely asynchronous
  - Based on “Services”

http://monkey.org/~marius/talks/twittersystems/#4

Late 2012 architecture

Many **open source** components

- Memcache, redis, MySQL, etc.
- Necessarily heterogeneous

Organized around **services**

- Distinct responsibilities
- Isolated from each other
- Distributed computation and data
- RPC between systems

Multiplexing HTTP frontend

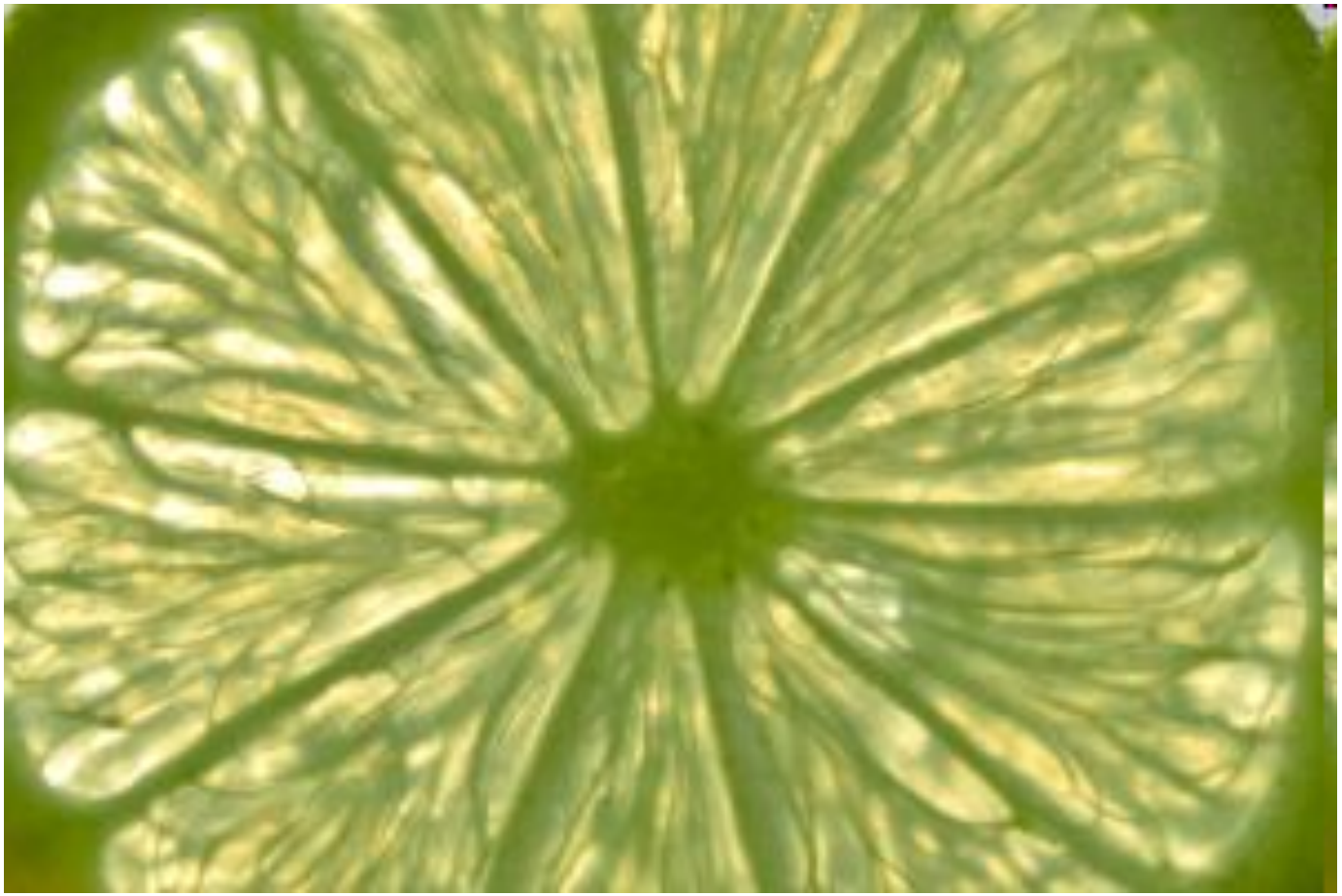
- Crucial for modularity, load balancing

Anti-patterns

- Use a full waterfall model
- Don't budget time for integration test
  - Assume that standard coding unit test->integration test will work
- Build unit tests that don't test interoperability
  - E.g. Simulate XML request/response inside the calling system rather than calling a remote system
- Wait until all the systems are ready before starting any integration test
  - A delay to one system will hold up testing all the others
- Don't bother with continuous build and test
  - Even better build by hand
  - **Even better** test by hand too
- Have a nice complex process to hand over from development to test
  - That way each defect will take a long time
- Wait until the project is failing to find out your team doesn't have the skills

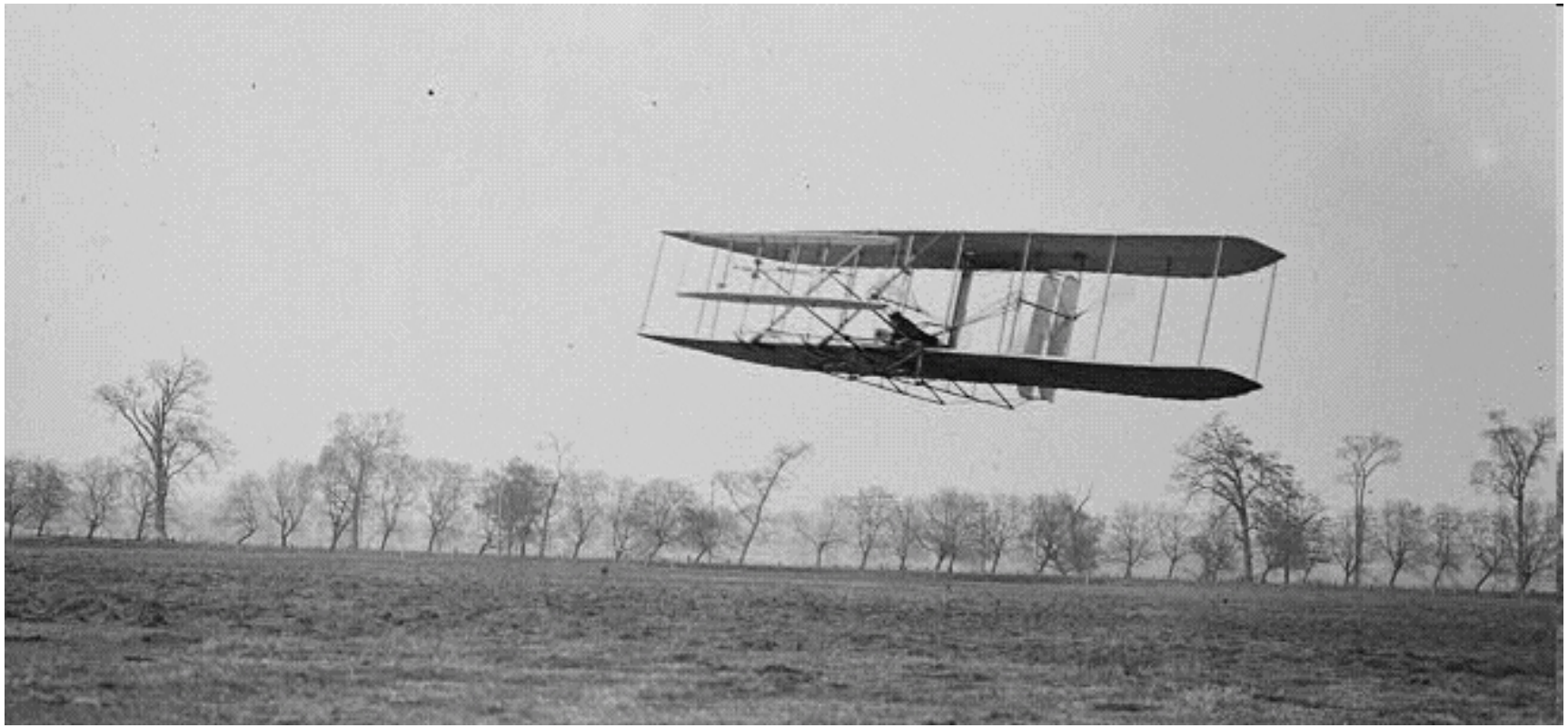
Conclusions

Thin slice prototyping is always a good idea



Iterative project plans are essential

Prove the concept to the business



KISS

Questions?