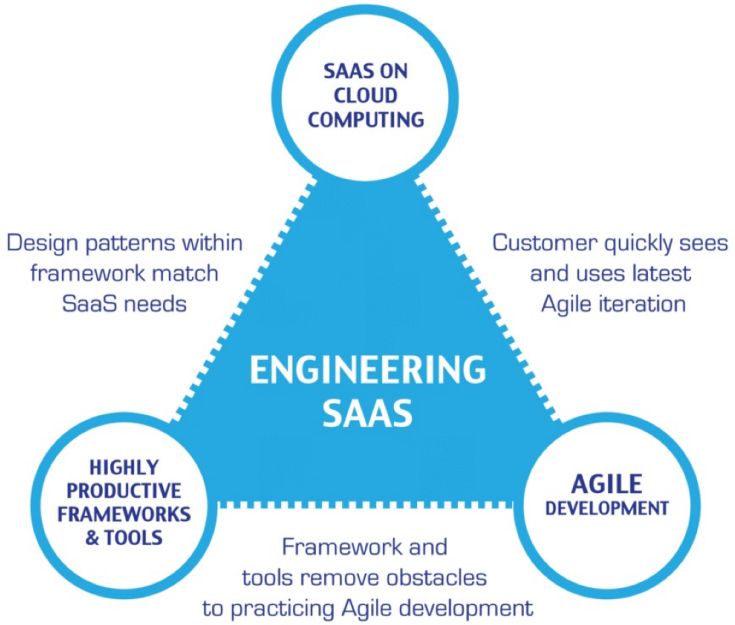
AUT UNIVERSITY COMPUTER + MATHEMATICAL SCIENCES




# Week 2: Cloud Computing and SOA

## 409232 Software Development Methods

Jim Buchan



Software Engineering Research Laboratory

# Today's Goals

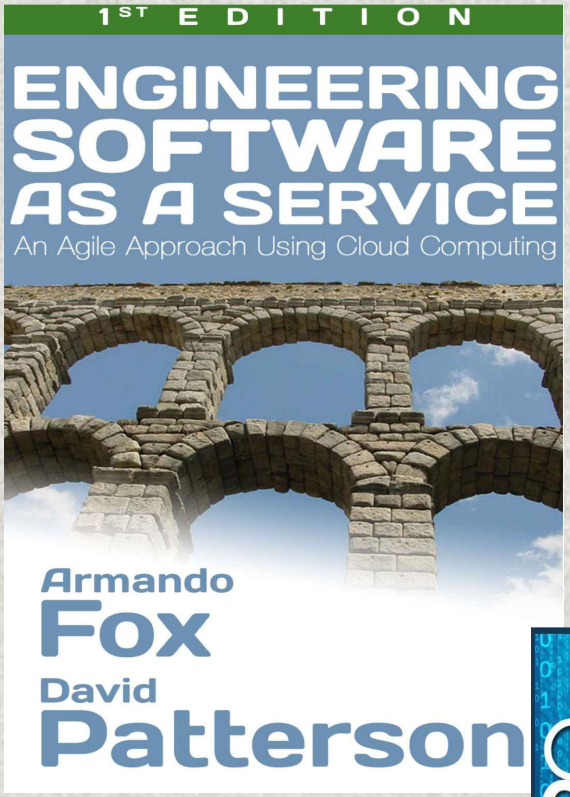
(20 min)
Review of what you should know from last week
Chapter 1 of the Textbook

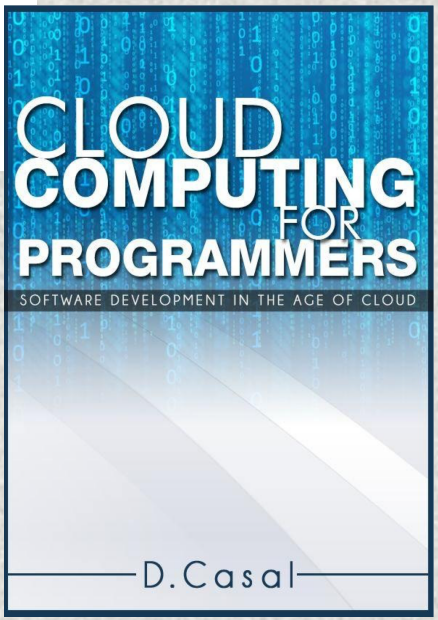
(40 min)
Service Oriented Architecture
Client-server
RESTful services
Application Programming Interfaces (APIs)

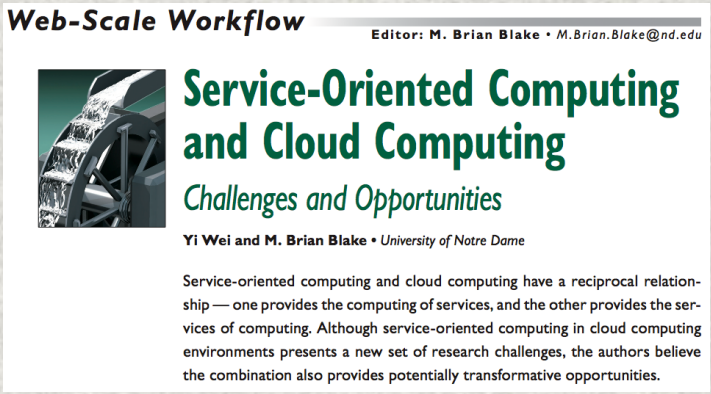
(30min)
Review of VM setup
GitHub
SSH
Heroku

(20min)
Assignments 1 & 2

# The Textbook

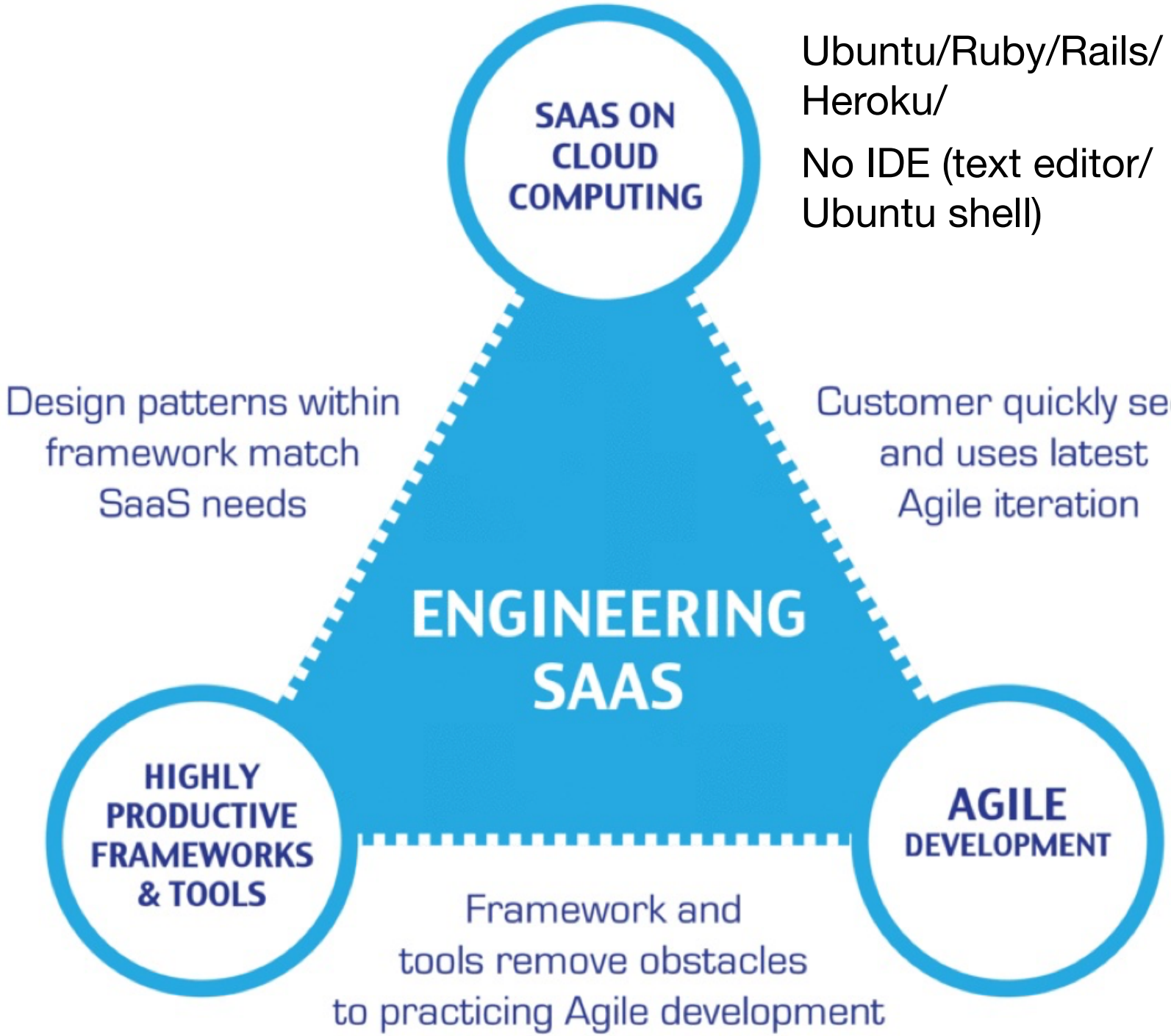



Internet computing, IEEE, 2010


Computer, IEEE, October 2011


SAAS ON CLOUD COMPUTING
Ubuntu/Ruby/Rails/ Heroku/
No IDE (text editor/ Ubuntu shell)
Design patterns within framework match SaaS needs
Customer quickly sees and uses latest Agile iteration
ENGINEERING SAAS
HIGHLY PRODUCTIVE FRAMEWORKS & TOOLS
AGILE DEVELOPMENT
Framework and tools remove obstacles to practicing Agile development

# Quizz

# The document-and-plan approach to software development has been superseded by agile approaches such as XP or Scrum

| | Question: A no answer suggests Agile; a yes suggests Plan and Document |
|---|---|
| 1 | Is specification required? |
| 2 | Are customers unavailable? |
| 3 | Is the system to be built large? |
| 4 | Is the system to be built complex (e.g., real time)? |
| 5 | Will it have a long product lifetime? |
| 6 | Are you using poor software tools? |
| 7 | Is the project team geographically distributed? |
| 8 | Is team part of a documentation-oriented culture? |
| 9 | Does the team have poor programming skills? |
| 10 | Is the system to be built subject to regulation? |

# Agile Principles

**Organizational**

1 Put the customer at the center.

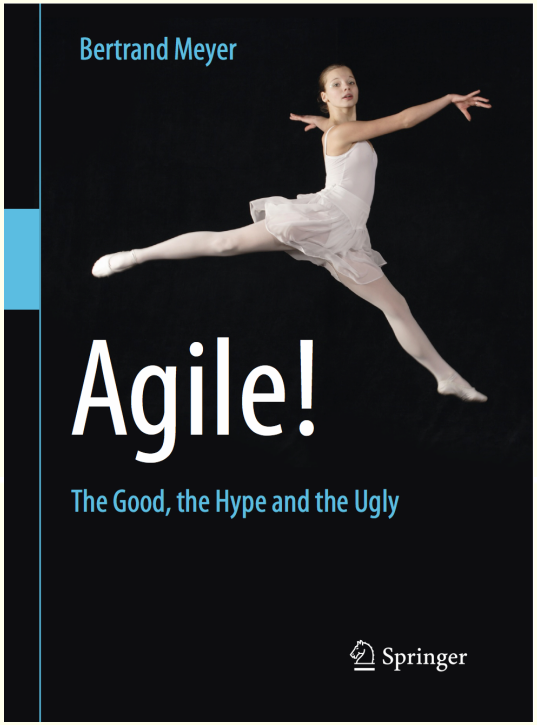
2 Let the team self-organize.

3 Work at a sustainable pace.

4 Develop minimal software:

- 4.1 Produce minimal functionality.
- 4.2 Produce only the product requested.
- 4.3 Develop only code and tests.

5 Accept change.




**Technical**

6 Develop iteratively:

- 6.1 Produce frequent working iterations.
- 6.2 Freeze requirements during iterations.

7 Treat tests as a key resource:

- 7.1 Do not start any new development until all tests pass.
- 7.2 Test first.

8 Express requirements through scenarios.

# Agile Principles

**Organizational**

1 Put the customer at the center.
2 Let the team self-organize.
3 Work at a sustainable pace.
4 Develop minimal software:
  4.1 Produce minimal functionality.
  4.2 Produce only the product requested.
  4.3 Develop only code and tests.
5 Accept change.

**Technical**

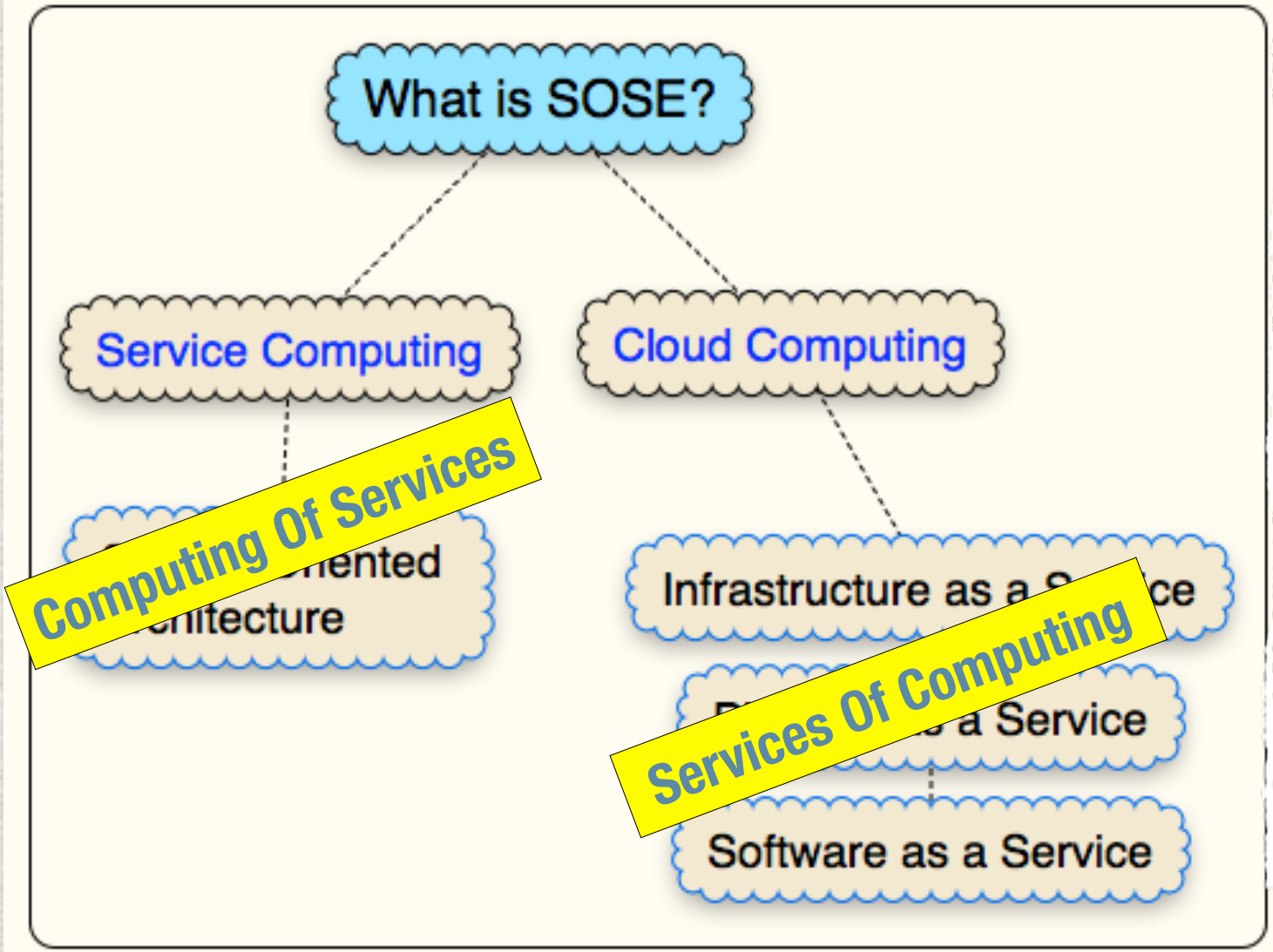
6 Develop iteratively:
  6.1 Produce frequent working iterations.
  6.2 Freeze requirements during iterations.
7 Treat tests as a key resource:
  7.1 Do not start any new development until all tests pass.
  7.2 Test first.
8 Express requirements through scenarios.

- Scrum
- Extreme Programming (XP)
  - Pair programming
  - Story Boards
- Kanban
- Behaviour Driven Development (BDD)
- Test Driven Development (TDD)
- Continuous Integration
- Continuous Delivery

# The document-and-plan approach to software development has been superseded by agile approaches such as XP or Scrum

| | Question: A no answer suggests Agile; a yes suggests Plan and Document |
|---|---|
| 1 | Is specification required? |
| 2 | Are customers unavailable? |
| 3 | Is the system to be built large? |
| 4 | Is the system to be built complex (e.g., real time)? |
| 5 | Will it have a long product lifetime? |
| 6 | Are you using poor software tools? |
| 7 | Is the project team geographically distributed? |
| 8 | Is team part of a documentation-oriented culture? |
| 9 | Does the team have poor programming skills? |
| 10 | Is the system to be built subject to regulation? |

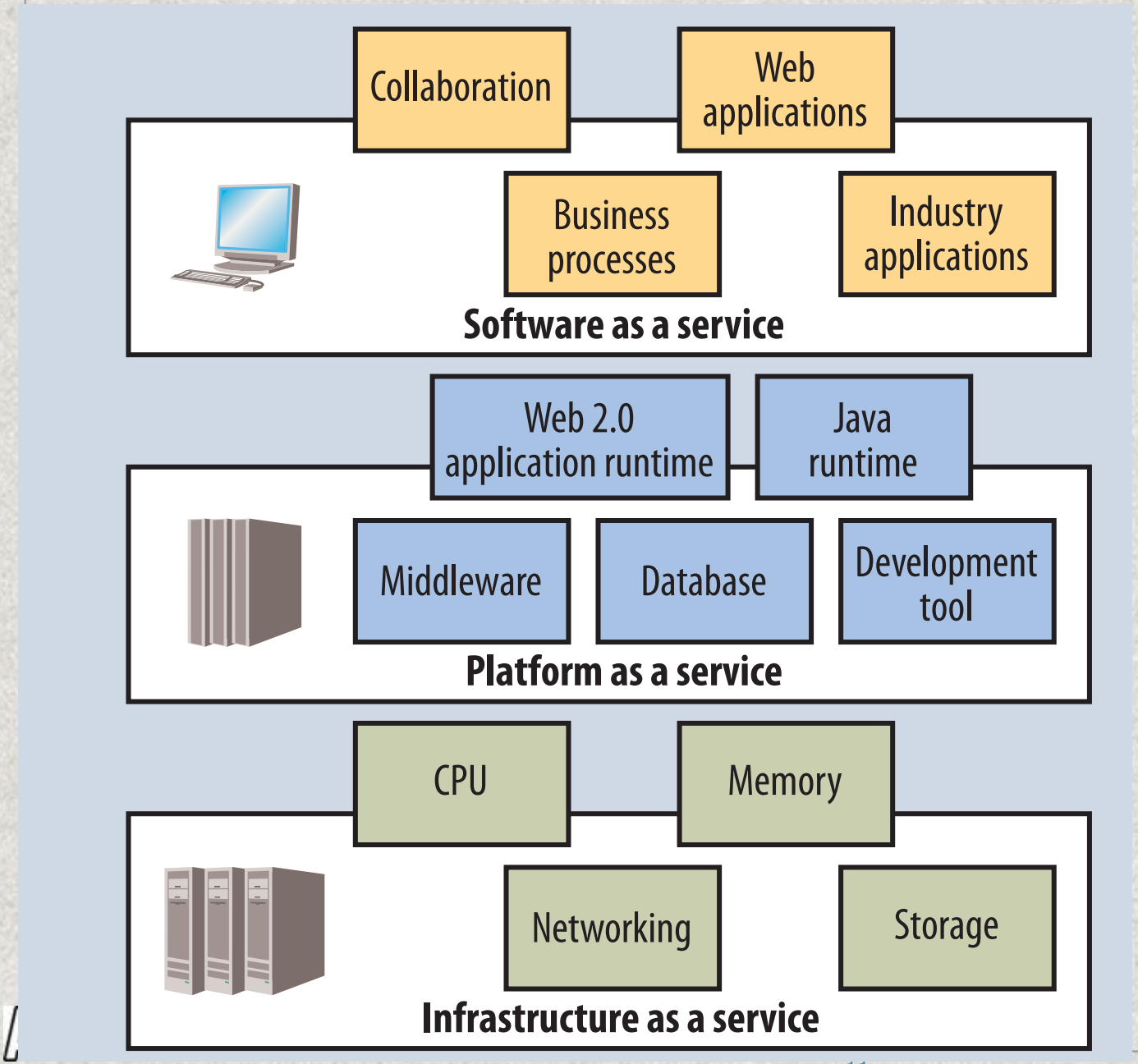
# Service Oriented SE




**Both offer**

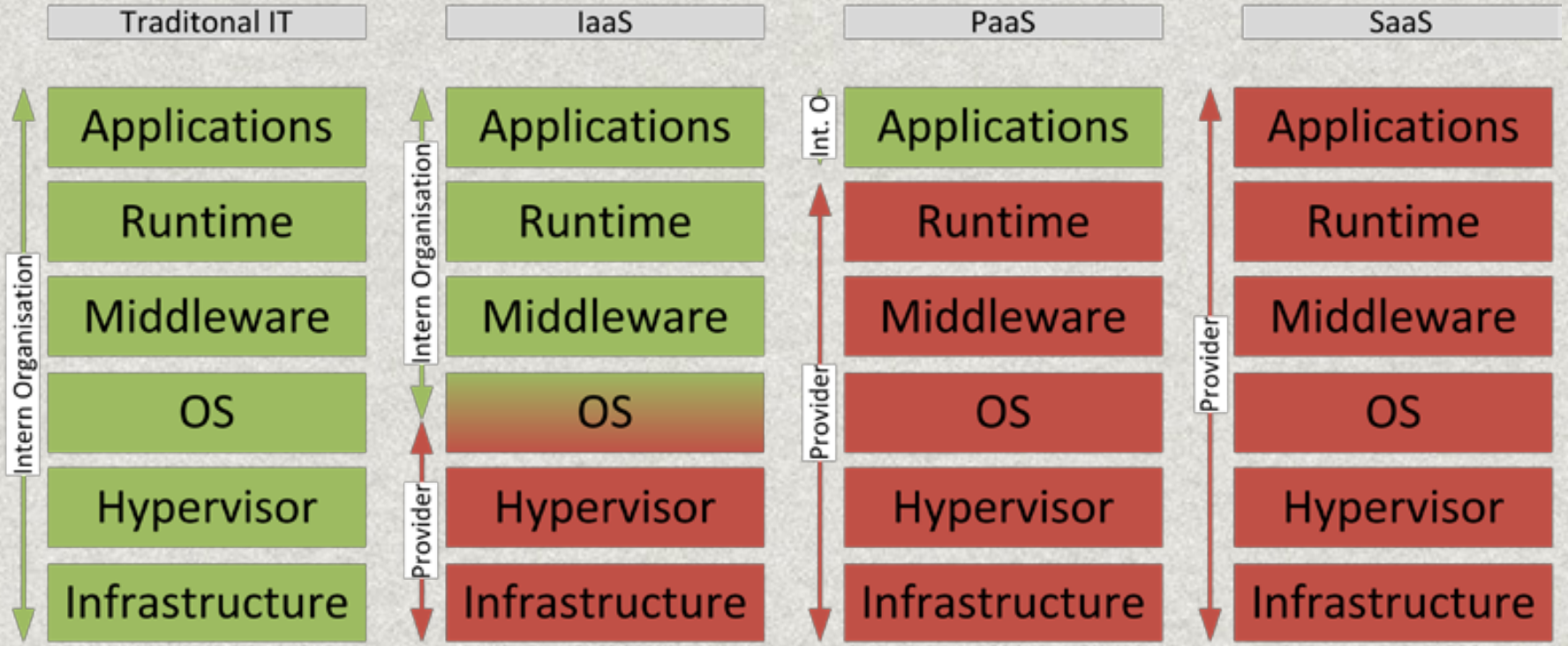
- rapid development
- large scale distributed applications
- resource outsourcing
- IT management with service providers

# 3 Cloud Services



# 3 Cloud Services

**Fig. 1 – Service models. Green indicates as level owned and operated by the organization, red that is run and operated by the service provider.**



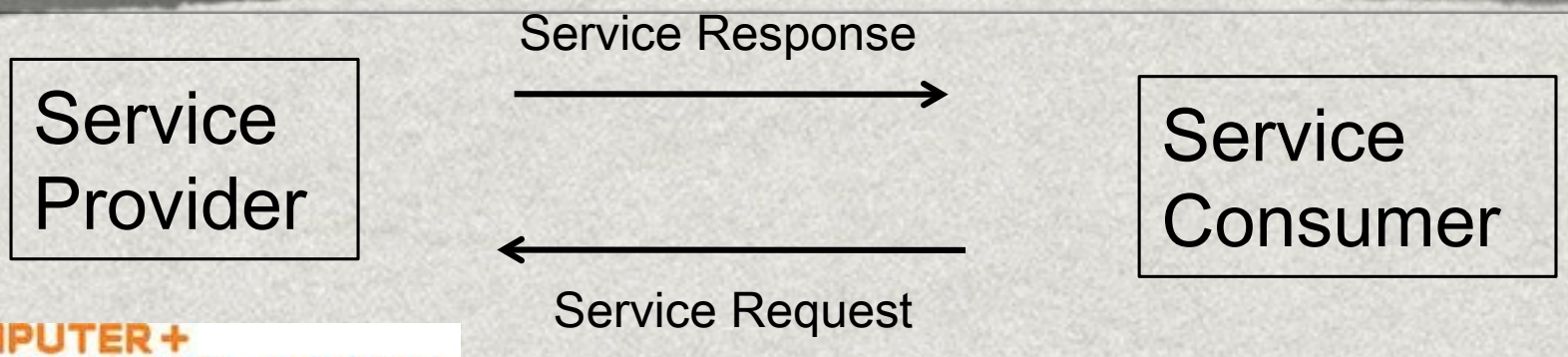

Source: Based on the model developed by NIST (2011)

# Service Oriented Computing

- Service developers follow SOA, an architectural model for creating and sharing computing processes, packaged as services
- Each service is an independent software entity with a well-defined standard interface that provides certain functions over networks
- A service is different from a traditional software artifact in that it's autonomous, self- described, reusable, and highly portable.
- software itself can be a service—a self-contained, stateless, and platform-independent entity with a URL, an interface, and functions that can be described and discovered as XML data.

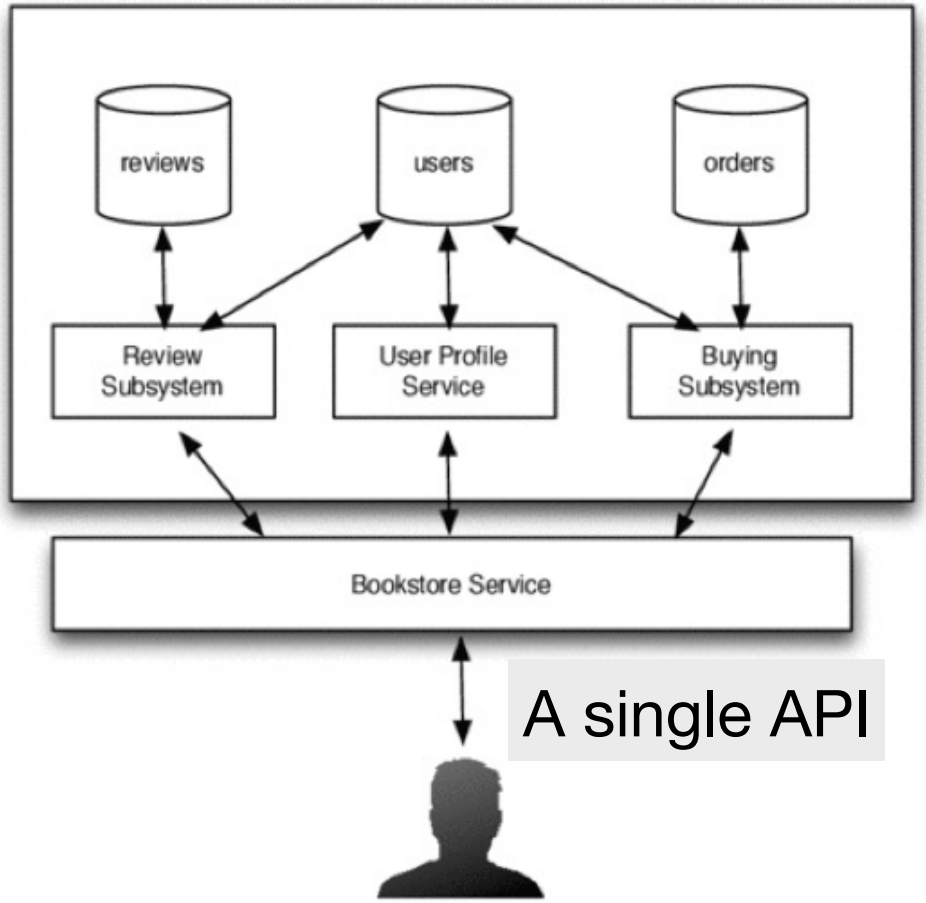
# Service Oriented Computing

- A service provider (or developer) is the party who develops and hosts the service.
- A service consumer is a person or program that uses a service to build an application.
- No service can name or access another services data - it can only make requests for data through an external API
- REUSABILITY is the big payoff

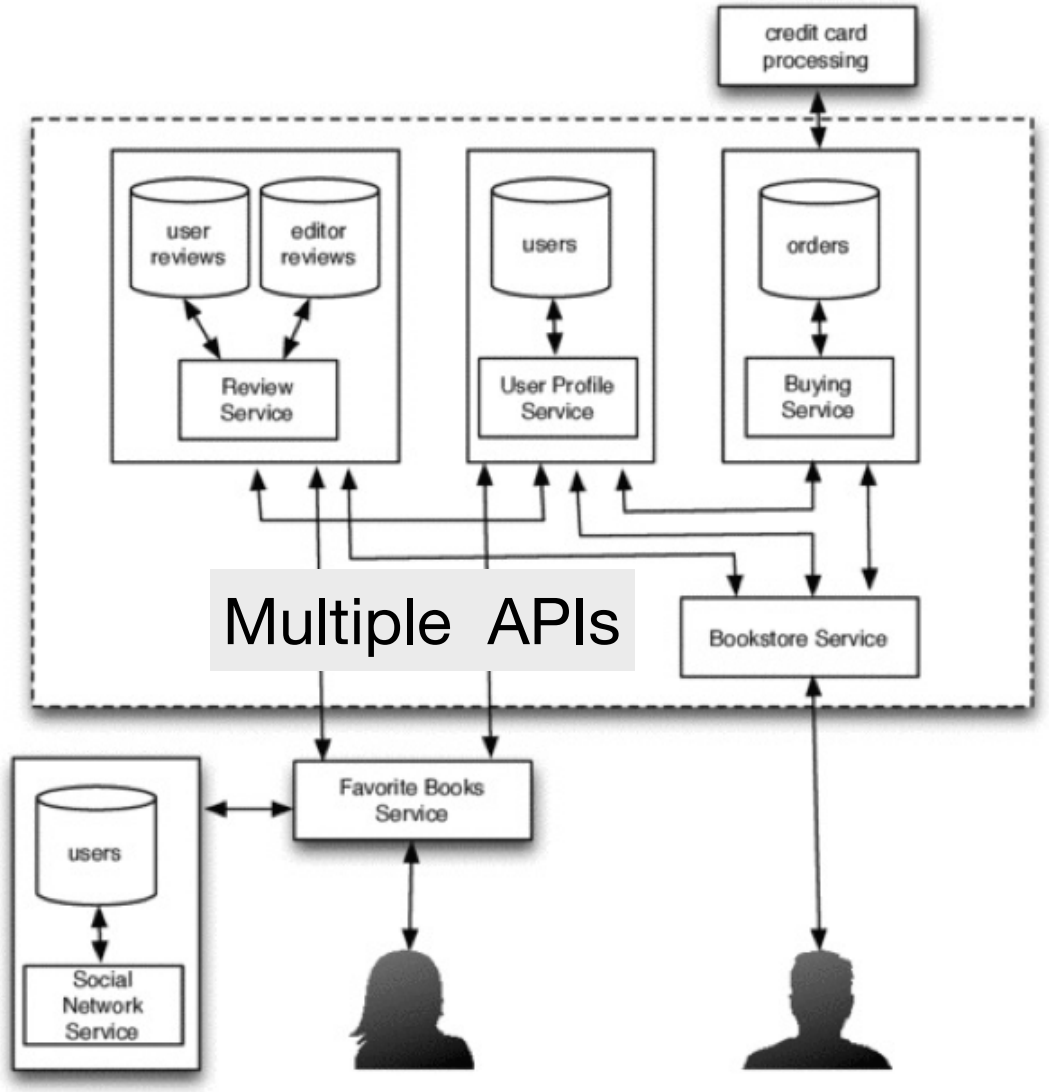


# Some notable Apps using SOA

- Amazon
- Facebook
- Twitter
- Google




A single API




Multiple APIs

- No service can name or access another services data - it can only make requests for data through an external API
- REUSABILITY is the big payoff - can combine in different ways

# The payoff

PLUS

- Reusability is very high
- testing is easier because APIs are explicit

MINUS

- Performance hit (many layers in software stack)
- Possibility of partial failure makes dependability planning more challenging.

# SaaS - A special case of SOA

Delivers software and data over the internet.

Usually a browser is all that is required at the client end

Examples

- Facebook
- Twitter
- Netflix
- Xero
- Google Docs
- Google
- Microsoft Online

# Benefits of SaaS

1. Since customers do not need to install the application, they don't have to worry whether their hardware is the right brand or fast enough, nor whether they have the correct version of the operating system.
2. The data associated with the service is generally kept with the service, so customers need not worry about backing it up, losing it due to a local hardware malfunction, or even losing the whole device, such as a phone or tablet.
3. When a group of users wants to collectively interact with the same data, SaaS is a natural vehicle.
4. When data is large and/or updated frequently, it may make more sense to centralize data and offer remote access via SaaS.
5. Only a single copy of the server software runs in a uniform, tightly-controlled hardware and operating system environment selected by the developer, which avoids the compatibility hassles of distributing binaries that must run on wide-ranging computers and operating systems. In addition, developers can test new versions of the application on a small fraction of the real customers temporarily without disturbing most customers.
6. SaaS companies compete regularly on bringing out new features to help ensure that their customers do not abandon them for a competitor who offers a better service.
7. Since only developers have a copy of the software, they can upgrade the software and underlying hardware frequently as long as they don't violate the external application program interfaces (API). Moreover, developers don't need to annoy users with the seemingly endless requests for permission to upgrade their applications.

# Is developing for the Cloud  Different?

- Developed Code such as enterprise Java/.NET doesn't care where the code is deployed (cloud or not). A web app developed with Spring-MVC can be deployed to an application server like JBoss in the cloud or locally.
- But there are specific architectures, languages, tools related to developing for the cloud and in the cloud!
- Cloud technologies have changed the way software development can be done

Cassandra
MongoDB
Closure
F#
Scala
Ceylon
MongoDB
Hadoop
Node.js
SOAP
REST
Confluence
Green hopper

# Examples of SaaS programming frameworks and the programming languages they are written in

| *SaaS Programming Framework* | *Programming Language* |
|---|---|
| Active Server Pages (ASP.NET) | C#, VB.NET |
| Django | Python |
| Enterprise Java Beans (EJB) | Java |
| JavaServer Pages (JSP) | Java |
| Rails | Ruby |
| Sinatra | Ruby |
| Spring | Java |
| Zend | PHP |

# Virtualization

- Unlimited number of servers available for development - (e.g. dev, test, pre-production, production…

- Fast setup of servers compared to setting up physical server

- Lowers friction for innovation and experimentation…create a new development environment quickly to test out ideas.

# Cloud Computing APIs

- APIs makes IT resources available to cloud users
- Provides a mechanism for Could applications to communicate (pass messages)
- Purpose is to hide complex computation behind simple interface
- Facebook, Twitter, Google Maps, Netflix, Xero etc
- Cloud APIs are service oriented, usually based on SOAP or REST

# View rottenpotatoes

1. A Web client (Firefox) requests the Rotten Potatoes home page from a Web server (WEBrick).

2. WEBrick obtains content from the Rotten Potatoes app and sends this content back to Firefox

3. Firefox displays the content and closes the HTTP connection.

**Figure 2.1:** 100,000-foot view of a SaaS client-server system.

# Start the server for rottenpotatoes and view it on a browser

1. A Web client (Firefox) requests the Rotten Potatoes home page from a Web server (WEBrick).

2. WEBrick obtains content from the Rotten Potatoes app and sends this content back to Firefox

3. Firefox displays the content and closes the HTTP connection.

**Figure 2.1:** 100,000-foot view of a SaaS client-server system.

1. A Web client (Firefox) requests the Rotten Potatoes home page from a Web server (WEBrick).
   a) Firefox constructs an HTTP request using the URI ***http://localhost:3000*** to contact an HTTP server (WEBrick) listening on port 3000 on the same computer as Firefox itself (***localhost***).
   b) WEBrick, listening on port 3000, receives the HTTP request for the resource '/movies' (the list of all movies in Rotten Potatoes).

**Self-Check 2.2.1.** What happens if we visit the URI `http://google.com:3000` and why?

- The connection will eventually "time out" unable to contact a server, because Google (like almost all Web sites)

**Self-Check 2.2.2.** What happens if we try to access RottenPotatoes at (say) `http://localhost:3300` (instead of `:3000`) and why?

- You get a "connection refused" since nothing is listening on port 3300.

**Summary**

- Web browsers and servers communicate using the ***HyperText Transfer Protocol***. HTTP relies on ***TCP/IP*** (Transmission Control Protocol/Internet Protocol) to reliably exchange ordered sequences of bytes.
- Each computer connected to a TCP/IP network has an ***IP address*** such as 128.32.244.172, although the ***Domain Name System*** (DNS) allows the use of human-friendly names instead. The special name `localhost` refers to the local computer and resolves to the special IP address 127.0.0.1.
- Each application running on a particular computer must "listen" on a distinct ***TCP port***, numbered from 1 to 65535 ($2^{16}$-1). Port 80 is used by HTTP (Web) servers.
- To run a SaaS app locally, you activate an HTTP server listening on a port on `localhost`. WEBrick, Rails' lightweight server, uses port 3000.
- A ***Uniform Resource Identifier*** (URI) names a resource available on the Internet. The interpretation of the resource name varies from application to application.
- HTTP is a stateless protocol in that every request is independent of every other request, even from the same user. ***HTTP cookies*** allow the association of HTTP requests from the same user. It's the browser's responsibility to accept a cookie from an HTTP server and ensure that the cookie is included with future requests sent to that server.

1. A Web client (Firefox) requests the Rotten Potatoes home page from a Web server (WEBrick).
   a) Firefox constructs an HTTP request using the URI ***http://localhost:3000*** to contact an HTTP server (WEBrick) listening on port 3000 on the same computer as Firefox itself (***localhost***).
   b) WEBrick, listening on port 3000, receives the HTTP request for the resource '/movies' (the list of all movies in Rotten Potatoes).

2. WEBrick obtains content from the Rotten Potatoes app and sends this content back to Firefox
   a) WEBrick returns content encoded in HTML, again using HTTP. The HTML may contain references to other kinds of media such as images to embed in the displayed page. The HTML may also contain a reference to a CSS stylesheet containing formatting information describing the desired visual attributes of the page (font sizes, colors, layout, and so on).

3. Firefox displays the content and closes the HTTP connection.
   a) Firefox fetches any referenced assets (CSS, images, and so on) by repeating the previous four steps as needed but providing the URIs of the desired assets as referenced in the HTML page.
   b) Firefox displays the page according to the CSS formatting directives and including any referenced assets such as embedded images.

**Figure 2.6:** SaaS from 10,000 feet. Compared to Figure 2.4, step 2 has been expanded to describe the content returned by the Web server, and step 3 has been expanded to describe the role of CSS in how the Web browser renders the content.

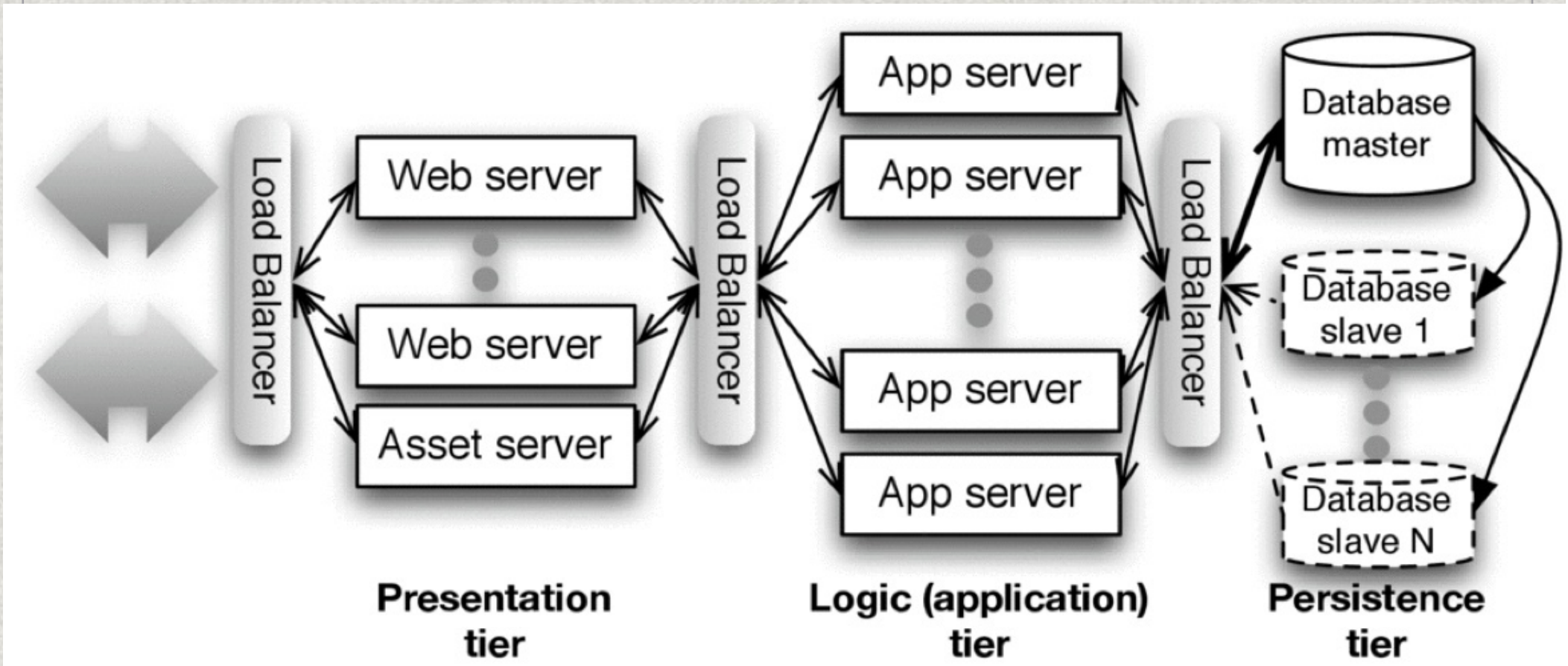



**Figure 2.7:** The 3-tier ***shared-nothing*** architecture, so called because entities within a tier generally do not communicate with each other, allows adding computers to each tier independently to match demand. ***Load balancers***, which distribute workload evenly, can be either hardware appliances or specially-configured Web servers. The statelessness of HTTP makes shared-nothing possible: since all requests are independent, any server in the presentation or logic tier can be assigned to any request. However, scaling the persistence tier is much more challenging, as the text explains.

1. A Web client (Firefox) requests the Rotten Potatoes home page from a Web server (WEBrick).
   a) Firefox constructs an HTTP request using the URI ***http://localhost:3000*** to contact an HTTP server (WEBrick) listening on port 3000 on the same computer as Firefox itself (***localhost***).
   b) WEBrick, listening on port 3000, receives the HTTP request for the resource '/movies' (the list of all movies in Rotten Potatoes).

2. WEBrick obtains content from the Rotten Potatoes app and sends this content back to Firefox
   a) Via the Rack middleware (written in Ruby), WEBrick calls Rotten Potatoes code in the application tier. This code generates the page content using movie information stored in the persistence tier implemented by a SQLite database using local files.
   b) WEBrick returns content encoded in HTML, again using HTTP. The HTML may contain references to other kinds of media such as images to embed in the displayed page. The HTML may also contain a reference to a CSS stylesheet containing formatting information describing the desired visual attributes of the page (font sizes, colors, layout, and so on).

3. Firefox displays the content and closes the HTTP connection.
   a) Firefox fetches any referenced assets (CSS, images, and so on) by repeating the previous four steps as needed but providing the URIs of the desired assets as referenced in the HTML page.
   b) Firefox displays the page according to the CSS formatting directives and including any referenced assets such as embedded images.

**Figure 2.8:** SaaS from 5,000 feet. Compared to Figure 2.6, step 2a has been inserted, describing the actions of the SaaS server in terms of the three-tier architecture.

# For next week - be prepared to explain or ask questions about

Read and Understand the……

Model View Controller perspective

The REST service perspective

# Lab for this week

Setup GitHub

Setup public/private key pair for use of ssh to communicate securely with the cloud services like GitHub

provide Github with your public key

Set up Heroku and provide your public key

# Assessments

Marks for Lab completion

Assessment 1 (40%) - Consult a software development company about the practices and tools. Write a FICTIONAL case study and give presentation of insights to share learning.

Assessment 2 (60%) - Investigate Software Development Methods

Conduct an experiment related to SDM

OR

Develop an application applying SDM to the development of this application. The application will be specified by your lecturer (product owner) and the development environment

# Collaboration

The Cloud makes it easy for developers to share their work and ideas with other team members.

Geographically dispersed teams have fast easy and consistent real-time access to the same information.

- Confluence
- GreenHopper
- Crucible
- VMWAre ZImbra
- Trello


# Cloud Tools Galore!

**IDEs**

- Koding, Cloud9, Codeanywhere, Codenvy, ShiftEdit

**Source Control**

- **GitHub**, BitBucket, Cloudforge, Mercurial, Fossil

**Source code Security Scanning**

- Fortify on Demand (HP), IBM Managed Security Services, Veracode

**Bug Tracking**

- Jira, Trackstudio, YouTrack, Issue tracking Anywhere

**Continuous Integration**

- TeamCity, **Travis CI,** Shining Panda, CircleCI, FaZend, Hosted CI

**Databases**

- MySQL, MongoDB (JSON-like documents, NoSQL), CouchDB, Cassandra, Neo4j

# Cloud Testing/Monitoring Tools

- **Functional testing**
  - SauceLabs, uTest, Mob4Hire (managed crowd for mobile apps)
- **Performance/Stress testing**
  - LoadStorm, BlazeMeter HP, Load Runner in the cloud
- **Monitoring**
  - Pingdom, monitor.us

# Web Analytics

- Google Analytics
- Clicky
- Woopra
- Clicktale

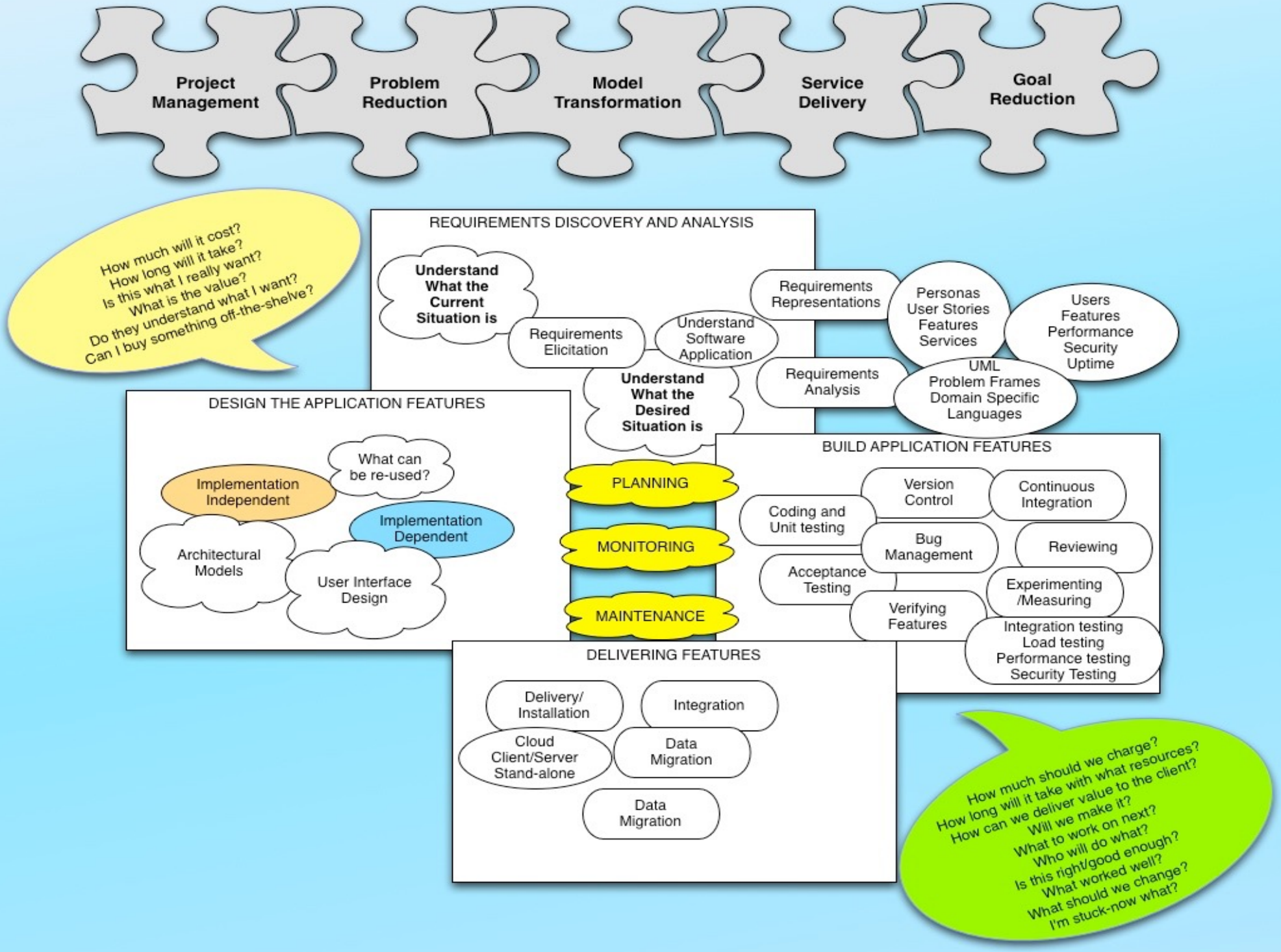
# New Programming Languages

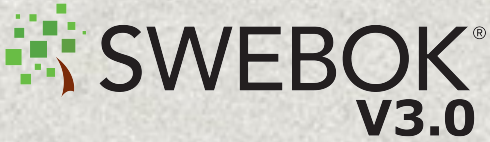
Multi-Paradigm: Object Oriented + Functional Programming

- Multi-threading
- Parallelism (two or more functions can be run in parallel on multi-CPU systems)
- Hot deployment (replace component)
- Ceylon, Chapel, Clojure, Dart, F#, Fantom, GO, HAXE, OPA, Scala, X10, Zimbu

# Libraries and Frameworks

- Hadoop (Apache)
  - Coordinates a number of machines working on a problem and compile work into a single answer
- Akka
- Node.js (built on Chrome's Javascript runtime)
- SignalR (ASP.NET developers)

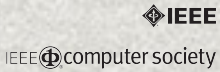

Project Management
Problem Reduction
Model Transformation
Service Delivery
Goal Reduction
How much will it cost?
How long will it take?
Is this what I really want?
What is the value?
Do they understand what I want?
Can I buy something off-the-shelve?
REQUIREMENTS DISCOVERY AND ANALYSIS
Understand What the Current Situation is
Requirements Elicitation
Understand Software Application
Understand What the Desired Situation is
Requirements Representations
Personas
User Stories
Features
Services
Users
Features
Performance
Security
Uptime
Requirements Analysis
UML
Problem Frames
Domain Specific Languages
DESIGN THE APPLICATION FEATURES
What can be re-used?
Implementation Independent
Implementation Dependent
Architectural Models
User Interface Design
PLANNING
MONITORING
MAINTENANCE
BUILD APPLICATION FEATURES
Version Control
Continuous Integration
Coding and Unit testing
Bug Management
Reviewing
Acceptance Testing
Experimenting /Measuring
Verifying Features
Integration testing
Load testing
Performance testing
Security Testing
DELIVERING FEATURES
Delivery/ Installation
Integration
Cloud
Client/Server
Stand-alone
Data Migration
Data Migration
How much should we charge?
How long will it take with what resources?
How can we deliver value to the client?
Will we make it?
What to work on next?
Who will do what?
Is this right/good enough?
What worked well?
What should we change?
I'm stuck-now what?




*Guide to the Software Engineering Body of Knowledge*

**Editors**

Pierre Bourque
Richard E. (Dick) Fairley




# SWEBOK

# International Conference on Software Engineering

# Modern Software Development Methods

Modern Agile ITPM is a mash-up from a selection of a variety of good ideas and practices from a variety of sources.

Scrum

Kanban

PMBoK

Lean

Extreme Programming

PMI

# Are Agile Methods Really Used?

7th ANNUAL STATE of AGILE

VERSIONONE® Agile Made Easier

Plan on using Agile Project Management for future projects

2011 - 59%
2012 - **83%**

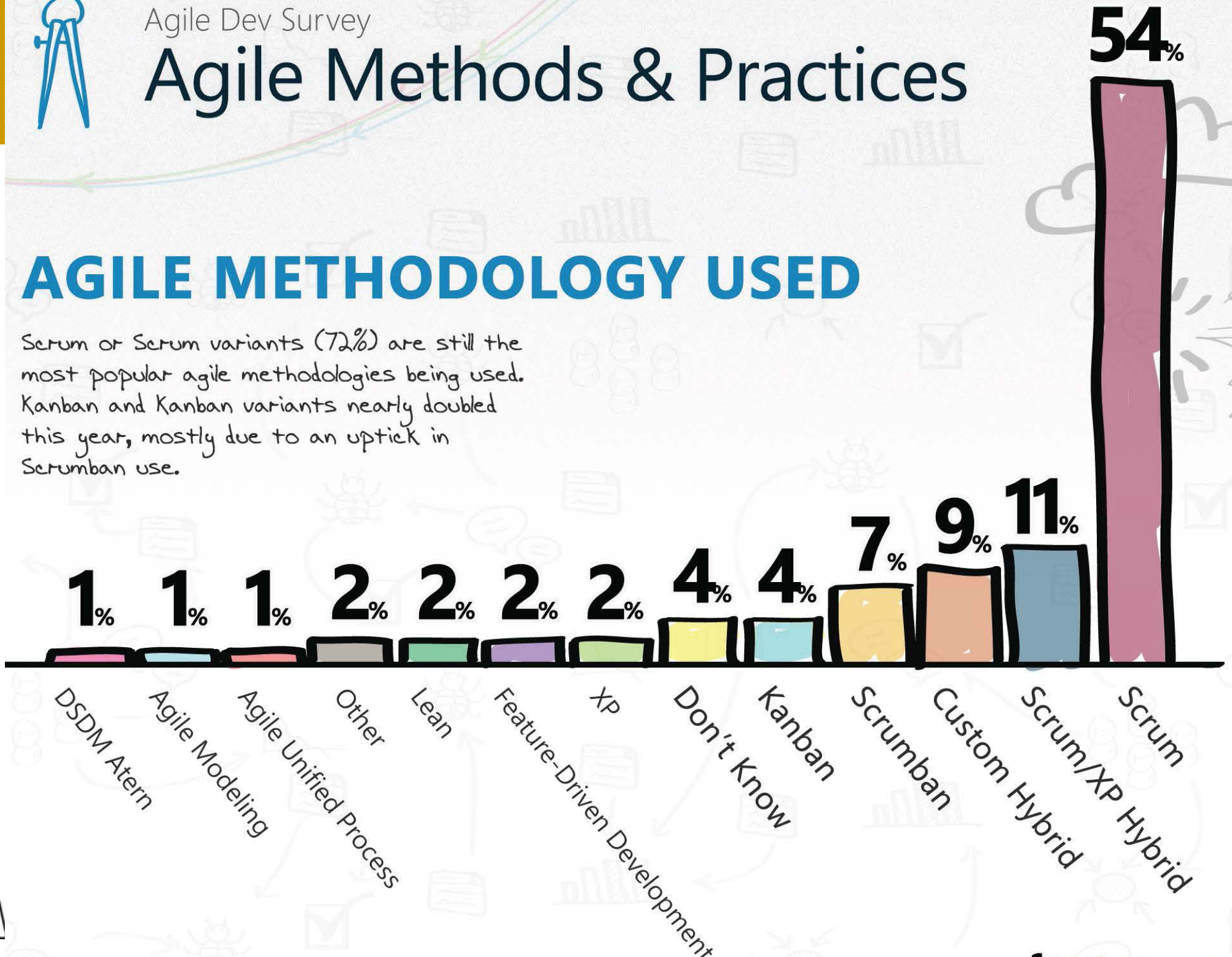

AUT UNIVERSITY | COMPUTER + MATHEMATICAL SCIENCES

Agile Dev Survey

# Agile Methods & Practices

## AGILE METHODOLOGY USED

Scrum or Scrum variants (72%) are still the most popular agile methodologies being used. Kanban and Kanban variants nearly doubled this year, mostly due to an uptick in Scrumban use.

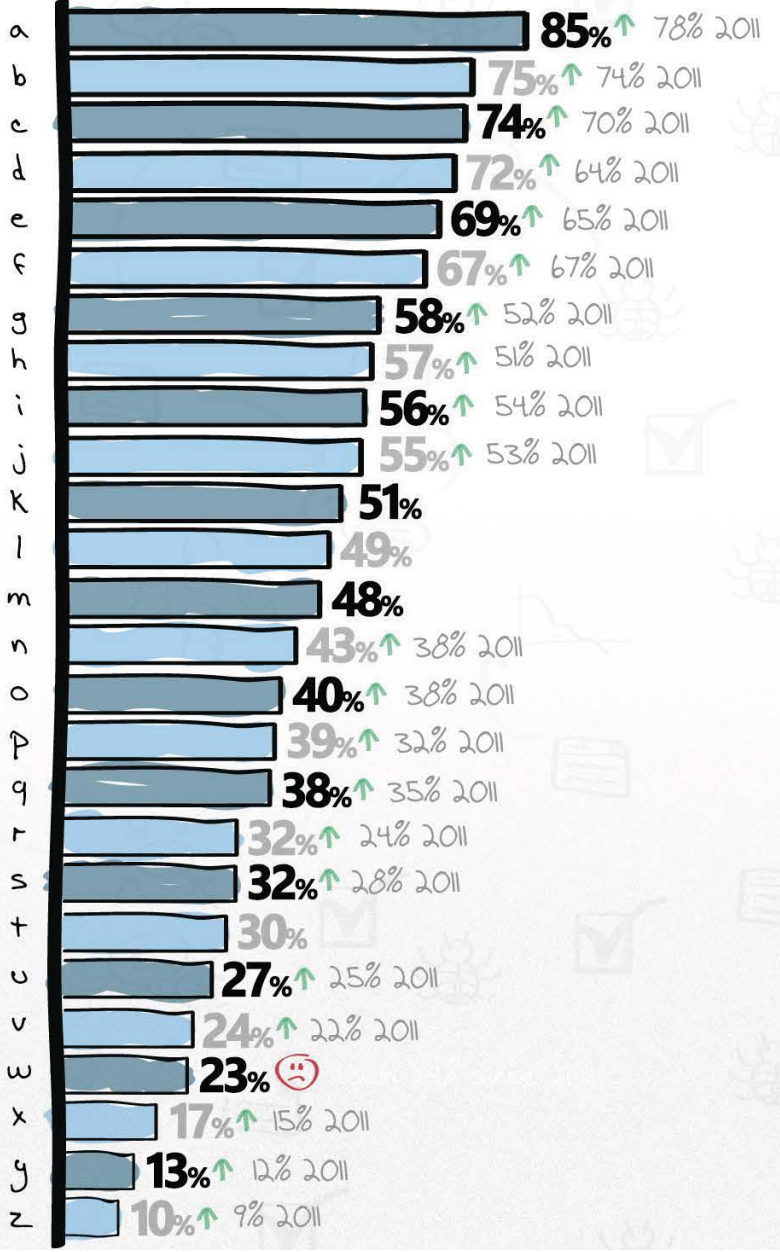


# AGILE TECHNIQUES EMPLOYED

Again this year, core agile tenets currently in use are* Daily Standup, Iteration Planning and Unit Testing. The two techniques that grew the most in usage from this year to last year were Kanban and Retrospectives; yet, agile techniques increased in every area but one (Continuous Deployment).

*Respondents were able to select multiple options.

- a Daily Standup
- b Iteration Planning
- c Unit Testing
- d Retrospectives
- e Release Planning
- f Burndown/ Team-Based Estimation
- g Velocity
- h Coding Standards
- i Continuous Integration
- j Automated Builds
- k Dedicated Product Owner
- l Integrated Dev/QA
- m Refactoring
- n Open Workarea
- o TDD
- p Digital Taskboard
- q Story Mapping
- r Kanban
- s Collective Code Ownership
- t Pair Programming
- u Automated Acceptance Testing
- v Analog Taskboard
- w Continuous Deployment
- x Agile Games
- y Cycle Time
- z BDD




AUT UNIVERSITY COMPUTER + MATHEMATICAL SCIENCES

SERL Software Engineering Research Laboratory

Agile Dev Survey

# Reasons for Adopting Agile

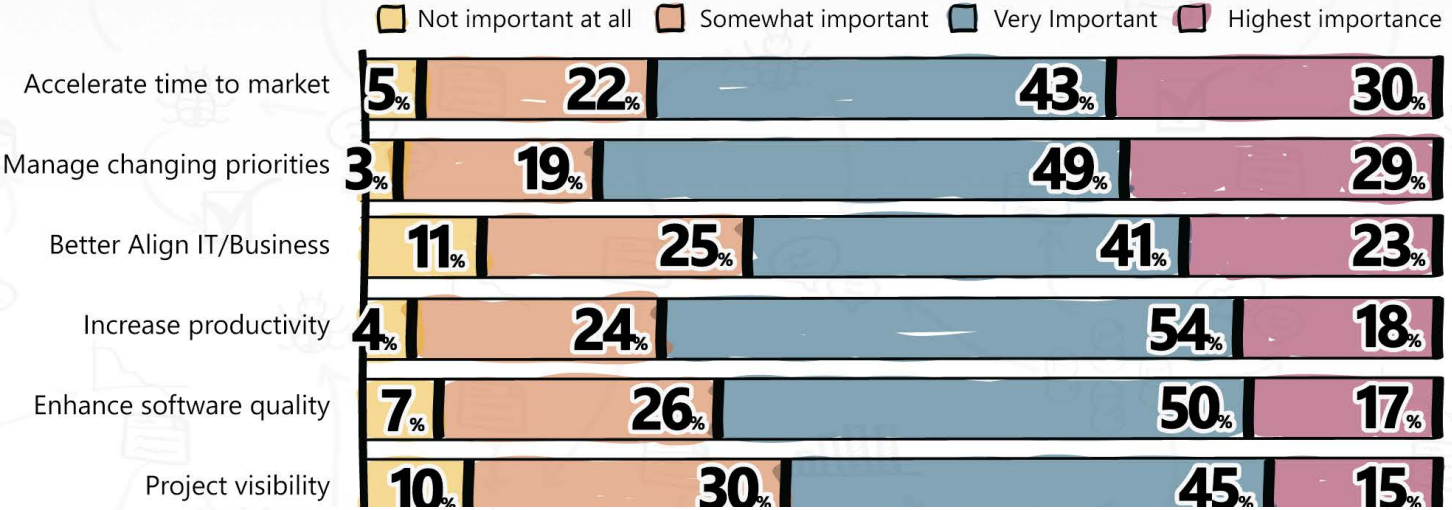
7th ANNUAL STATE of AGILE

## WHY AGILE?

Most responses centered on better customer focus and increased predictability.

Once again, top 3 reasons* respondents cited for adopting agile were to accelerate time to market, more easily manage changing priorities, and to better align IT and business objectives:

*Respondents were able to select multiple options.




Simplify developm

Enhance software maintainability/

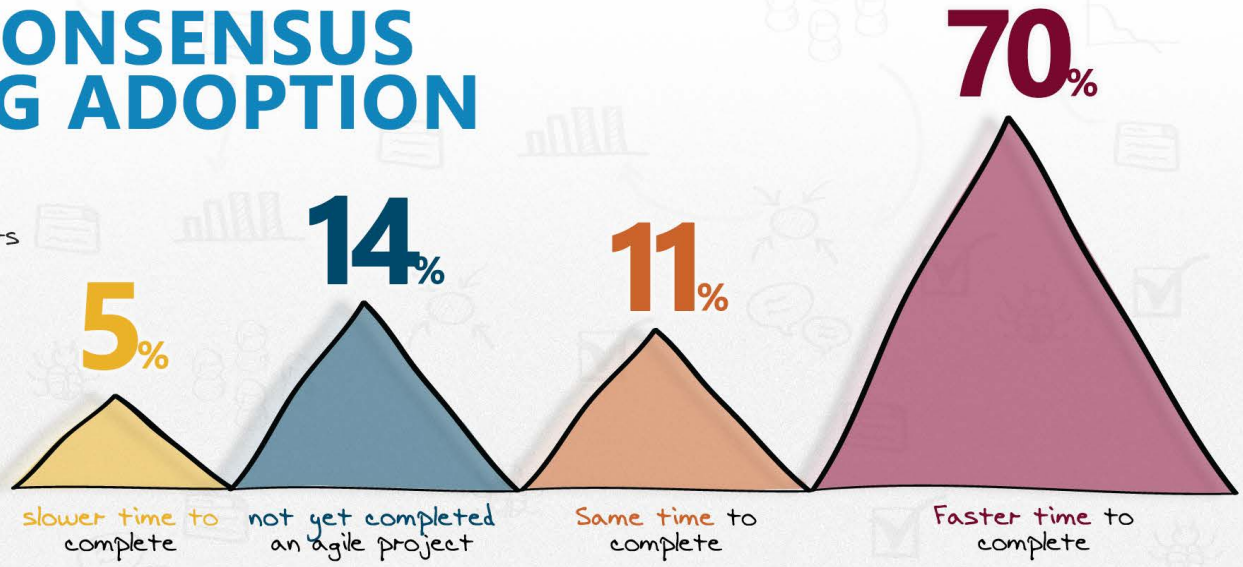
Improve t

Improve/increase engineeri

Manage distril

## OVERALL CONSENSUS REGARDING ADOPTION OF AGILE

The vast majority of respondents felt that agile projects have a faster time to completion.



# My Empirically Based Educational Philosophy

Constructivist Learning

Collaborative Learning

Student centred learning

Lecturer as facilitator and guide

Learning to Learn

Deep thinking/learning rather than shallow, non-critical

Questioning is a master skill

Communicating ideas is as important as having them