Web Technologies

Web programming (II)
from MVC to Web architectures and case studies

Dr. Sabin Corneliu Buraga – profs.info.uaic.ro/~busaco/
“Simplicity is complexity resolved.”

Constantin Brâncuși
Web applications
complex software systems,
in a permanent evolution
Reality

multiple ways of **Web interaction** with the user

mobile laptop  PC  tablet  *(smart)* TV  urban screen
Reality

multiple ways of **Web interaction** with the user

- mobile laptop
- PC
- tablet
- *(smart)* TV
- urban screen

plus, newcomers:

- 🎮 game console
- ⌚ smart watch
- 👟 smart clothing
- 🌐 smart appliances
- 🏡 smart home
- ⚡ smart transportation
Reality

an increased number of users,
having greater expectations from actual software

from (hyper-)text content to social Web applications + natural interaction
Reality

the support for **developing applications** (languages, APIs, libraries, tools,...) provided by the hardware/software platform on server-side and/or client-side
Reality

lack of adjustment regarding the economic (business) requirements

🛠 development vs. 📝 marketing vs. ⌂ management
Reality

with respect to large Web projects

launch delays
over budgeting
lack of functionality
poor application quality
goals psychology behavior
interaction affordances language
features technology algorithms
indexing structure meta-data
tools methodologies incentives

users interface software content creators

adapted from Crumlish & Malone, 2009
evolution of digital (software) products development
Alan Cooper et al., 2014
Web application quality assurance

correctness and reliability
extending + reusing (modularity)
compatibility
efficiency (performance)
portability
Web application quality assurance

usability
functionality
timeliness
maintainability
security
Web application quality assurance

other aspects of interest:
  integrity
  reparability
  verifiability – including monitoring (logging)
  economy
Web application quality assurance

in essence, we must consider:

- request retrieval and routing – dispatch
- providing basic functionalities – core services
- associating software constructs/abstractions (e.g., objects) to the data models – mapping
- data management – data
- system monitoring & auditing – metrics


highscalability.com/blog/2016/10/12/lessons-learned-from-scaling-uber-to-2000-engineers-1000-ser.html
Necessities

a proper specification of aims + requirements
Web application systematic (iterative) development
correct planning of development stages
full monitoring of the entire development process
Necessities

a proper specification of aims + requirements
Web application systematic (iterative) development
correct planning of development stages
full monitoring of the entire development process

› Web engineering
In what way can we develop a Web application?
modeling

Usually, a methodology is adopted

MDA (Model-Driven Architecture) is usually preferred

www.omg.org/mda/
web application development

requirements
software design
build (implementation)
testing
deployment
maintenance
evolution
Web application (software product)

functionality + information (content)

web application development

currently, the agile methodologies are preferred

www.infoq.com/process-practices/

www.webpagefx.com/blog/web-design/agile/
web application development

Modern methodologies – examples:

aim42 – practices & patterns regarding software system evolution, maintenance, migration and improvement

aim42.github.io

12 Factor App – considers building SaaS (Software As A Service) applications

12factor.net
web application development: principles

start with needs
do less
design with data
do the hard work to make it simple
iterate. then iterate again
build for inclusion
understand context
build digital services, not Websites
be consistent, not uniform
make things open; it makes things better

an example for gov.uk – Paul Downey & David Heath (2013)
Establishing quality standards

requirements

users

content

context
requirements

Getting / bidding / negotiating
data (content) and/or source code

copyright

versus

open software (Open Source Licenses)

www.opensource.org/licenses/category

+ 

open data

Creative Commons – www.creativecommons.org/licenses/
<table>
<thead>
<tr>
<th>License</th>
<th>Code License</th>
<th>Rules</th>
<th>Managed By</th>
<th>Views</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Commons Clause&quot; License Condition v1.0</td>
<td>kevin</td>
<td>2</td>
<td>kevin</td>
<td>8171</td>
<td>0</td>
</tr>
<tr>
<td>YouTube Terms of Service</td>
<td>seldon</td>
<td></td>
<td>seldon</td>
<td>33001</td>
<td>6</td>
</tr>
<tr>
<td>Sleepycat License</td>
<td>valeriedouglas</td>
<td>3</td>
<td>kevin</td>
<td>16625</td>
<td>4</td>
</tr>
<tr>
<td>Fair Source License 0.9 (Fair-Source-0.9)</td>
<td>kevin</td>
<td>6</td>
<td>bfitzg</td>
<td>10807</td>
<td>5</td>
</tr>
<tr>
<td>Mozilla Public License 1.0 (MPL-1.0)</td>
<td>bfitzg</td>
<td>5</td>
<td>bfitzg</td>
<td>16575</td>
<td>5</td>
</tr>
<tr>
<td>GNU Lesser General Public License v3 (LGPL-3.0)</td>
<td>kevin</td>
<td>5</td>
<td>kevin</td>
<td>158692</td>
<td>14</td>
</tr>
<tr>
<td>GNU General Public License v2.0 (GPL-2.0)</td>
<td>kevin</td>
<td>4</td>
<td>kevin</td>
<td>151672</td>
<td>1</td>
</tr>
</tbody>
</table>
requirements

Documenting on the Web application domain by attracting experts – subject matter expert (SME) or domain expert – in the domain of the problem to be solved by the Web application
requirements: examples

Vision (big idea)

**Basecamp**: “solves the critical problems that every growing business deals with”

**Vimeo**: “watch, upload and share HD and 4k videos with no ads”

**Wikidata**: “a free and open knowledge base that can be read and edited by both humans and machines”
requirements: examples

Starting points in the Flickr development

initial assumptions:

people like to share memories

using the success of blogging

sharing of (personal) photos, not just notes

providing support for commenting + tagging
new types of requirements

Regarding the content

- audience – *e.g.*, internationalization
- navigation context
- user preferences
- permanent availability (7 days, 24 hours/day)
- using heterogeneous data sources
- support for searching, filtering, recommending
  etc.
new types of requirements

User interaction in the Web context

as well as the social Web

canent mash-up

“it’s yours to take, re-arrange and re-use”
new types of requirements

Regarding the execution (runtime) environment

Web browser (in)dependence

wired vs. wireless
on-line vs. off-line

support for various HTML standards
multi-device interactivity (responsive Web design)
new types of requirements

Concerning the evolution

end users are able to use the Web application without (re)installing it on a computer/device
new types of requirements: aspects

initially:
providing essential capabilities – less is more
new types of requirements: aspects

initially:
providing essential capabilities – less is more

next versions:
extending the Web application – usually, via a public API (Application Programming Interface), encouraging the development of user-provided solutions
architectures

Web applications quality is influenced by the base architecture

[martinfowler.com/architecture/](http://martinfowler.com/architecture/)
architectures

Web application architecture

body of code
that’s seen by developers as a single unit

group of functionality
that business customers see as a single unit

initiative
that those with the money see as a single budget
architectures

The development of a software architecture considers:

**functional requirements**

dictated by customers, visitors, competition, decision factors (management), social/technological growth,

...
architectures

The development of a software architecture considers:

**qualitative factors**

- usability
- performance
- security
- data/code reusing
- etc.
architectures

The development of a software architecture considers:

**techn(olog)ical aspects**

- hardware/software platform – *i.e.* operating system
- middleware infrastructure
- available services – *e.g.*, via public APIs
- programming language(s)
- legacy systems

...
The development of a software architecture considers:

- experience

adopting existing architectures and platforms

- design patterns

using specific solutions: libraries, frameworks, tools,...

- project management

- etc.
client(s)
firewall
proxy
middleware
Web server(s)
Web application server(s)
frameworks, libraries, other components
persistent storage server(s) – *e.g.* databases
multimedia content server(s)
content management server(s) – *e.g.*, CMS, wiki
legacy applications/systems

typical “ingredients”
client(s)
firewall
proxy
middleware
**Web server(s)**
**Web application server(s)**
frameworks, libraries, other components
**persistent storage server(s) – e.g. databases**
multimedia content server(s)
content management server(s) – *e.g.*, CMS, wiki
legacy applications/systems

eventually, using cloud services – **cloud computing**
computing resources and data sharing on demand, with other computers/devices, based on Internet technologies (hosting, scalable infrastructure, parallel processing, monitoring, ...)
Are there certain “recipes” regarding Web application development?
design

Pattern

rule denoting a relation between a context, a problem and a solution
A pattern specification and/or “identification” could be found on various tiers:

data presentation – UI, user interaction, visualization,...
processing – business logic, scripting etc.
component integration – code library development
data storage – database queries, database design,...
Examples of pattern collections (pattern repositories) concerning software design

wiki.c2.com/?DesignPatterns

patterns of enterprise application architecture

martinfowler.com/eaaCatalog/

user interaction (Adele – a repository of publicly available design systems and pattern libraries)

adele.uxpin.com
Web Patterns

Model View Controller
Page Controller
Front Controller
Template View
Transform View
Application Controller

M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley (2003)
Session State Patterns

Client Session State
Server Session State
Database Session State

Data Source Architectural Patterns

Table Data Gateway
Row Data Gateway
Active Record
Data Mapper

Web application = **interface** + **program** + **content** (data)

3-tier application
web architectures

Data structure model is detached by the processing model (business logic, application control) and the data presentation model (Web interface)

principle: separation of concerns
Most Web applications are developed by adopting **MVC (Model-View-Controller)**

Trygve Reenskaug, 1979

[heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf](heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf)
web architectures: mvc

- **View**: presentation + interaction
- **Model**: data structure
- **Controller**: processing
web architectures: **mvc**

- **View**: on client(s) side – *e.g.*, Web
- **Model**: persistent storage
- **Controller**: (server and/or client) application
web architectures: mvc

 символьно ↔ 符号
HTML, CSS, SVG, MathML, WebGL, WebXR, etc.

html, css, svg, mathml, webgl, webxr, etc.

(No)SQL, JSON, XML (XQuery), RDF (SPARQL), …

💡 application servers, frameworks

View
on client(s) side – e.g., Web

Controller
(server and/or client) application

Model
persistent storage
web architectures: mvc

Could be implemented in a non-object-oriented language encouraged/imposed by specific Web frameworks

various examples:
- ASP.NET MVC (C# et al.)
- Catalyst (Perl)
- ColdBox (ColdFusion)
- Django (Python)
- FuelPHP
- Grails (Groovy)
- Laravel (PHP)
- Lift (Scala)
- Rails (Ruby)
- Sails (Node.js)
- TurboGears (Python)
- Yesod (Haskell)
- Wicket (Java)
- Wt (C++)
- Zikula (PHP)
- ZK (Java)
web architectures: mvc

Controller

retrieves requests from a client (GET/POST requests representing various user actions)

manages the resources required for request fulfilment

usually, will call a model according to the required action and, after that, will select the corresponding view
web architectures: mvc

Model

software-managed resources – users, messages, products, etc. – have specific models

signifies data + rules (*i.e.* constraints) concerning data

› **concepts** managed by the Web application

offers the controller a representation of desired data and is responsible with the validation of stored data
web architectures: mvc

View

provides various ways of presenting the data offered by the model via controller

multiple views can exist, a specific one being chosen by the controller
typical steps:
(1) request sent by client – *e.g.*, Web browser,
(2) routing request to the controller,
(3) choosing a *model*, (4) providing the desired data,
(5) selecting a *view*, (6) content directed to the client
A Web application’s generic architecture will consist of a set of resources regarding the controller, model, and view.
A Web application’s generic architecture will consist of a set of resources regarding the controller, model, and view.

commonly, the used Web framework will enforce a specific file structure of the application to be implemented.
a Web application “skeleton” created with Ruby on Rails rubyonrails.org
the directory structure of a Web application built with CakePHP framework

cakephp.org

others: CodeIgniter, FuelPHP, Laravel, Symfony, Yii
the directory structure generated for a Web application using the Play framework for Java and Scala

www.playframework.org
the software project structure for an ASP.NET MVC application www.asp.net/mvc
Derived variants:

HMVC (Hierarchical Model-View-Controller)
MVVM (Model View ViewModel)
MVP (Model View Presenter)

for details, see
Herberto Graca, MVC and its alternatives (2017)
herbertograca.com/2017/08/17/mvc-and-its-variants/
Richard’s guide to software development

HOW THE SOFTWARE IS DESIGNED.

HOW MUCH TIME HAS TO BE SPENT ON EACH PART.

HOW THE SOFTWARE LOOKS BEFORE THE BETA TEST.

HOW THE SOFTWARE LOOKS AFTER THE BETA TEST.

HOW THE SOFTWARE IS ADVERTISED.

WHAT THE CUSTOMER REALLY WANTED.

HOW THE SOFTWARE LOOKS TWO VERSIONS LATER.

I STILL LIKE YOU ANYWAY.

?:

TOOT
By what means a Web application could be implemented?
implementation

Web application server

goal:
optimizing the development processes of complex Web applications
could encourage or impose an architectural approach for Web application development – e.g., MVC or variants
Web application architecture: classic MV* approach

dumb client  

fat server

browser  

<Web/> pages

HTML, CSS,…

presentation  processing  data access

content generation at Web server level
usually, monolithic application

www.leaseweblabs.com/2013/10/api-first-architecture-fat-vs-thin-server-debate/
Web application architecture: classic MV* approach

dumb client

fat server

browser

frontend

<Web/> pages

HTML, CSS,...

presentation

processing

data access

backend

design principle: layers of isolation

changes made in a particular layer have no impact on or do not affect the components in another layer
implementation

Framework

facilitates complex Web application development, by simplifying usual operations (e.g., access to databases, caching, code generation, session management, access control) and/or boosting the code-source reuse
implementation

Various frameworks facilitating Web application development on the backend:

**ASP.NET**: ASP.NET Core MVC

**Java**: Play, Spring, Sling, Struts, Tapestry, Wicket

**JavaScript** (*Node.js*): Express, Geddy, LoopBack, Meteor

**Perl**: Catalyst, Dancer, Mojolicious

**Python**: CherryPy, Django, Flask, TurboGears

**Ruby**: Padrino, Rails, Sinatra
implementation

Web library

a collection of reusable computational resources – *i.e.*, data structures + code – providing specific functionalities (behaviors) implemented in a given programming language
implementation

Web library

a collection of reusable computational resources – *i.e.*, data structures + code – providing specific functionalities (behaviors) implemented in a given programming language could be referred by other source code (software): application server, framework, library, service, API, or Web component
Open source libraries – examples:

- Apache PDFBox – pdfbox.apache.org
- Beautiful Soup – www.crummy.com/software/BeautifulSoup
- D3.js – d3js.org
- Expat – libexpat.github.io
- ImageMagick – www.imagemagick.org
- libcurl – curl.haxx.se
- Libxml2 – www.xmlsoft.org
- OpenCV – opencv.org
- Requests-HTML – github.com/kennethreitz/requests-html
- TensorFlow.js – www.tensorflow.org/js/
- zlib – www.zlib.net
implementation

Web service

software – remotely used by other applications/services – providing a specific functionality,
    frequently through an API (Application Programming Interface)

its implementation must not be known by the programmer that invokes the service

details in future lectures
implementation

SDK (Software Development Kit)

encapsulates the API functionalities into a library
(implemented in a programming language,
for a specific software/hardware platform)

API façade pattern

example: Octokit (for .NET, Objective-C, Ruby)
provided by Github – developer.github.com/v3/libraries/
implementation

Web component

part of a Web application that includes a set of related features
developed by using a JavaScript library/framework

e.g., calendar, news feed reader, URL sharing button
implementation

Widget

application – stand-alone or included into a container (e.g., a HTML document) – offering a specific functionality

runs on a client (a platform provided by an operating system and/or a Web browser)
implementation

(Web) app

an installable (Web) application that uses the APIs provided by a platform: browser, application server, operating system,...

*a distributed computer software application designed for optimal use on specific screen sizes and with particular interface technologies*

Robert Shilston, 2013
Web browser

platform (OS + device)

native app

single page app

app store

Google Drive app for Android/iOS

drive.google.com

HTTP WebSocket

advanced

Web applications and services (APIs)

Google Drive API
Google Drive SDK for Java, Node.js, Python

adapted from Adrian Colyer (2012)
implementation

Add-on

a generic name for browser-associated applications (extensions, visual themes, dictionaries, Web search engine interfaces, plug-ins etc.)

examples: addons.mozilla.org
development

Using development environments

examples – native applications (for desktop):
Anjuta, Aptana Studio, Eclipse, Emacs, IntelliJ IDEA, KomodoIDE, Padre, PHPStorm, PyCharm, RubyMine, Visual Studio, Zend Studio

cloud computing-based solutions:
AWS Cloud9, Codenvy, Koding, RELP.it etc.

S. Buraga, Code in “clouds”: www.slideshare.net/busaco/cu-codul-n-nori
Development as a Service
DigitalOcean, Google Cloud Platform, Heroku, Jelastic, OpenStack, Windows Azure,...

runtime environment

development

code repositories

BitBucket, GitHub

Web: AWS Cloud9, Koding, Ideone, etc.
desktop: Eclipse, Visual Studio (Code),...

integrated devel. environment

useful tools at github.com/ripienaar/free-for-dev
Automatic generation of documentation, in various formats

specific instruments: documentation generators

examples:

Doc, Document! X, Doxygen, JavaDoc, JSDoc, phpDocumentor
development

VCS – Version Control Systems

code review, revision control, versioning

monitoring changes in the source code made by a team of programmers on the same group of programs (codebase)
Client/server tools:
Apache Subversion – SVN
Microsoft Team Foundation Server – TFS

Distributed solutions:
Git (implemented in bash, C, and Perl)
  git-scm.com
Mercurial (developed in Python)
  mercurial.selenic.com
Rational Team Concert (offered by IBM)
  jazz.net/products/rational-team-concert/

Web systems to host software
(SCM – source code management):
BitBucket – developer.atlassian.com/cloud/bitbucket
GitHub – developer.github.com
GitLab – about.gitlab.com/handbook/
Encouraging/enforcing coding conventions and styles

client level:
HTML + CSS
JavaScript – profs.info.uaic.ro/~busaco/teach/courses/staw/web-film.html#week7

server level:
C# – github.com/dennisdooomen/csharpguidelines
Perl – perldoc.perl.org/perlstyle.html
PHP – www.php-fig.org/psr/psr-2/
Python – www.python.org/dev/peps/
Ruby – github.com/styleguide/ruby
Scala – docs.scala-lang.org/style/

for others, see google.github.io/styleguide/
development

Software package management

searching, installing, compiling, dependence checking

examples:
Bower, Composer, npm, NuGet, RubyGems, Yarn

also, study github.com/showcases/package-managers
development

Support for workflows
optionally, automatically performed

producing a Web application from source code +
additional components (build tool)

examples:
Ant, Grunt, Gulp, make, Mimoza, Rake, tup, Yeoman
Tests concerning the source-code

automatic test units – general framework: xUnit, HttpUnit, JUnit (Java), PHPUnit, xUnit.net (C#, F#), Test::Class (Perl), unittest (Python), Unit.js + JSUnit, FireUnit, Mocha, Selenium etc. for client-side

for details, study xunitpatterns.com
testing

Web application specific tests regarding **content** – structure, HTML + CSS validation,…

**hypertext** issues (*e.g.*, broken links)

**usability** – including accessibility, multi-lingual data

**Web interface esthetics** – hard to evaluate/test
testing

Web application specific tests

component integration

permanent availability and flexibility (continuous evolution)

device independence – multi-screen (a large number of possible devices + characteristics)
testing

Other kinds of testing:
concerning performance
load, stressing, continuous testing, scalability

real-life case studies:
High Scalability – highscalability.com
Performance Planet – calendar.perfplanet.com
Web Performance Stats – wpostats.com
Other kinds of testing:

regarding security

in a future lecture
testing: example

HTML documents – validator.w3.org service
CSS stylesheets – CSS Lint: csslint.net
JSON data – validation using JSONLint
XML documents – well-formed / valid
Client-side scripting (JavaScript) via JS/ES Hint
Server-side (backend) programs – e.g., xUnit
File system integrity and access policies
Database integrity and access policies
Web navigator support – caniuse.com
Security problems – www.owasp.org
Aspects regarding Web application performance
running

Site publishing

dedicated server
vs.
Web hosting provider
free vs. commercial solution

response time, scalability, security, technical support,...
running

Content maintenance (administration)

obtaining, creating, preparing, managing, presenting, processing, publishing, and reusing the content in a systematic and structured manner
running: management

At the organizational level:
knowledge management

CRM – Client Relationship Management

ERP – Enterprise Resource Planning

workflow + business rule management

EAI – Enterprise Application Integration
running: management

At the technical level:

managing the content by non-technical personnel on the basis of separation of concerns principle

Content Management Systems (CMS)

collaborative tools
(e.g., an enterprise wiki)
running: management

Regarding the user:

Web interaction – *e.g.*, usability
profs.info.uaic.ro/~busaco/teach/courses/hci/

social Web application design patterns
profs.info.uaic.ro/~busaco/teach/courses/hci/hci-film.html#week7

frontend performance
profs.info.uaic.ro/~busaco/teach/courses/staw/web-film.html#week13
running: usage analysis

Explicit methods
based on data provided by users
*e.g.*, surveys and monitoring (user testing),
e-mail message analysis, reactions on social networks,...

Implicit methods
automatic data collecting (user analytics)
usually, via cookies
running: usage analysis

User profile generation: Web usage mining

log file analysis
(e.g., access.log – Apache, AWStats, ...)

Website “popularity” measurement:
load speed, number of accesses, visit time + duration etc.

monitoring/reporting services
examples: Google Analytics, WordPress Statistics
parameters of a web project

main objective
duration
cost
methodology
technologies
processes
outcome
human resources
team profile
Web project team

- management
  - Web Project Manager
- functionality
  - Software Engineer(s)*
  - Multimedia Designer(s)
- content (data)
  - Domain Expert
  - Business Expert

*frontend or backend or full-stack (frontend & backend)

www.slideshare.net/busaco/sabin-buraga-dezvoltator-web-n-2019
Several examples regarding Web application architectures?
case study: flickr

Aim:

online sharing of graphical content (photos)

representative social Web application

community aggregation – image as social object

support for annotations via tagging

+ comments
case study: **flickr – technologies**

**PHP** (processing – application logic, access to API, content presentation via **Smarty**, e-mail module)
**Perl** (data validation)
**Java** (storage node management)
**MySQL** (data stored in InnoDB format)
**ImageMagick** (C library for image processing)
**Ajax** (asynchronous interaction)
**Linux** (runtime platform)

other details at [highscalability.com/flickr-architecture](http://highscalability.com/flickr-architecture)
initial architecture – according to Cal Henderson, 2007
application programming interfaces (APIs) provided by Flickr facilitate the access to Web services for applications running on various platforms requests: REST, XML-RPC, SOAP responses: REST, XML-RPC, SOAP, JSON

www.flickr.com/services/api/
generic aspects concerning the system design:

resource categories: *user + picture*

relations between *user* instances (e.g., *follow*)

relations between *user* and *picture* instances (*make, depicts, comment, like, ...*)

performance features:
response time, scalable software architecture, scalable persistent storage, image optimization

recommending resources (*user/picture*) of interest

details in *Create a Photo Sharing App (2016)*

blog.gainlo.co/index.php/2016/03/01/system-design-interview-question-create-a-photo-sharing-app/
case study: netflix

Aim: providing video content on demand (streaming) + Web TV

services available on multiple devices/platforms

cloud-based deployment

adopting, among others, various open technologies

netflixtechblog.com
aspects of interest:
content (movies, TV shows,…),
storage & encoding/decoding (transcoding),
adaptive streaming,
playback (support for heterogeneous devices)
<table>
<thead>
<tr>
<th>Category</th>
<th>Technology Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>backend processing</strong></td>
<td>Java, Python, Node.js (JavaScript)</td>
</tr>
<tr>
<td><strong>frontend processing</strong></td>
<td>React, WinJS (JavaScript)</td>
</tr>
<tr>
<td><strong>storage systems</strong></td>
<td>MySQL, Apache Cassandra, Apache Hadoop, Apache Hive, Oracle DB</td>
</tr>
<tr>
<td><strong>cloud services</strong></td>
<td>Amazon EC2 (video processing) &lt;br&gt;Amazon S3 (storage)</td>
</tr>
<tr>
<td><strong>SQL services</strong></td>
<td>Amazon RDS (Relational DB Service)</td>
</tr>
<tr>
<td><strong>NoSQL services</strong></td>
<td>Amazon DynamoDB</td>
</tr>
<tr>
<td><strong>code management</strong></td>
<td>GitHub (implemented in Ruby + C)</td>
</tr>
<tr>
<td><strong>continuous integration</strong></td>
<td>Jenkins (Java implementation)</td>
</tr>
<tr>
<td><strong>server management</strong></td>
<td>Apache Mesos (written in C++)</td>
</tr>
<tr>
<td><strong>content distribution networks</strong></td>
<td>Open Connect CDN (FreeBSD, Nginx), Akamai, Level 3, Limelight</td>
</tr>
<tr>
<td><strong>monitoring</strong></td>
<td>Boundary, LogicMonitor, Vector,...</td>
</tr>
</tbody>
</table>
case study: netflix

overview of the internal architecture
Di Lin, Girish Lingappa & Jitender Aswani (2019)
case study: gitlab

Aim: Web application for managing the software development cycle via Git from project planning and source code management to continuous integration and monitoring

distributed freely (community edition) – installable on Linux – or through subscriptions (enterprise edition)

docs.gitlab.com/ee/development/architecture.html
about.gitlab.com/handbook/engineering/infrastructure/production-architecture/
reverse proxy: **NGINX** • Web server: **Unicorn** • persistent storage: **PostgreSQL** (users, meta-data), **Redis** (Web sessions, cache, message queues) • access to Git: **Gitaly** • message queues processing: **Sidekiq** • monitoring (metrics): **Prometeus** • infrastructure: **Terraform**
<table>
<thead>
<tr>
<th>Web Application</th>
<th>Backend processing</th>
<th>Persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>Perl, Java</td>
<td>MySQL (MariaDB),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amazon DynamoDB,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amazon SimpleDB,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amazon ElastiCache</td>
</tr>
<tr>
<td>Coursera</td>
<td>Django (Python),</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Node.js (JavaScript), Play (Scala)</td>
<td>Apache Cassandra</td>
</tr>
<tr>
<td>DuckDuckGo</td>
<td>Node.js</td>
<td>PostgreSQL</td>
</tr>
<tr>
<td>Facebook</td>
<td>Hack, PHP (HHVM),</td>
<td>RocksDB, Presto,</td>
</tr>
<tr>
<td></td>
<td>Tornado (Python),</td>
<td>Cassandra, Beringei</td>
</tr>
<tr>
<td></td>
<td>Java, JavaScript</td>
<td></td>
</tr>
<tr>
<td>Forbes</td>
<td>ASP.NET, Node.js,</td>
<td>MySQL (MariaDB)</td>
</tr>
<tr>
<td></td>
<td>PHP 7, Ruby</td>
<td></td>
</tr>
<tr>
<td>Google</td>
<td>C++, Dart, Go,</td>
<td>BigTable, MariaDB</td>
</tr>
<tr>
<td></td>
<td>Java, Python</td>
<td></td>
</tr>
</tbody>
</table>

[stackshare.io/stacks]
<table>
<thead>
<tr>
<th>Web Application</th>
<th>Backend processing</th>
<th>Persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linkedin</td>
<td>Grails (Java), JavaScript, Scala</td>
<td>MySQL, Oracle DB, RocksDB, Hadoop</td>
</tr>
<tr>
<td>Lyft</td>
<td>PHP, Flask (Python), Java, Go, C++</td>
<td>MongoDB, DynamoDB, Redis</td>
</tr>
<tr>
<td>Medium</td>
<td>Node.js, Go</td>
<td>Neo4j, DynamoDB, Redis</td>
</tr>
<tr>
<td>Pinterest</td>
<td>Django (Python), Java, Go</td>
<td>MySQL, Hadoop, Apache HBase, Redis, Memcached</td>
</tr>
<tr>
<td>Stack Overflow</td>
<td>.NET Framework (C#)</td>
<td>MS SQL Server, Redis</td>
</tr>
<tr>
<td>Sound Cloud</td>
<td>Clojure, Scala, JRuby</td>
<td>MySQL</td>
</tr>
</tbody>
</table>

[link to stackshare.io/stacks]
inspecting the technologies used by a Web application using the **WhatRuns** tool
summary

Web programming ▶ Web engineering
Web application development – essential aspects
next episode:
Web application development in PHP