Software at Scale

CSSE6400

Brae Webb

March 21, 2022

Question

How many concurrent users can your software handle?

Question

How many concurrent users can your software handle?

Answer

Maybe *400*? Maximum.

Question

How many concurrent users can your software handle?

Answer

Maybe *400*[1]? Maximum.

[1] HTTP server on a t2.micro EC2 instance

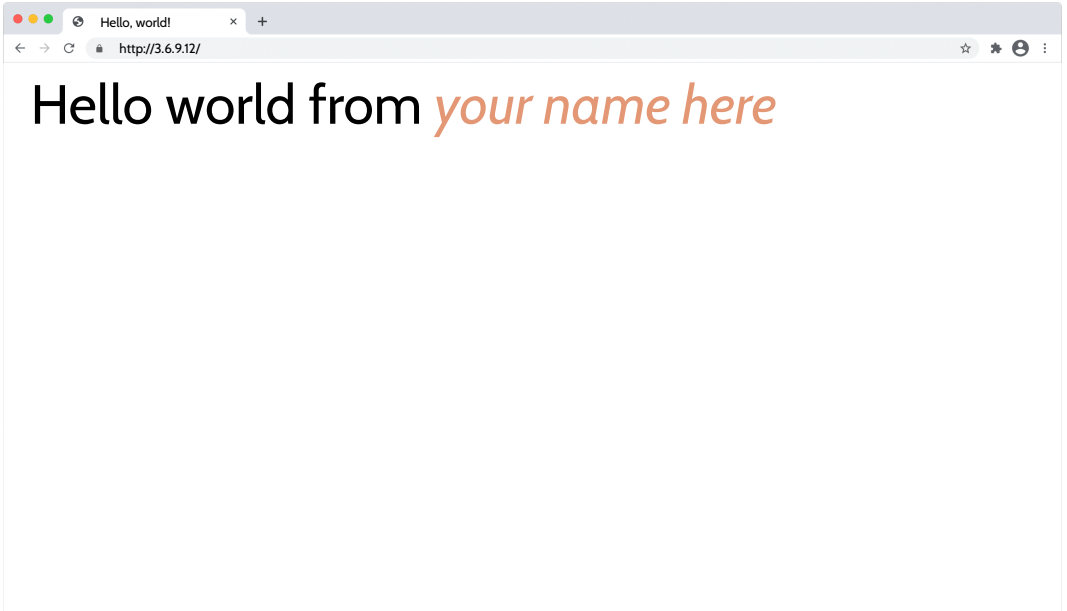
Definition 1. Stress Testing

Measure the robustness of software by pushing usage to an extreme.

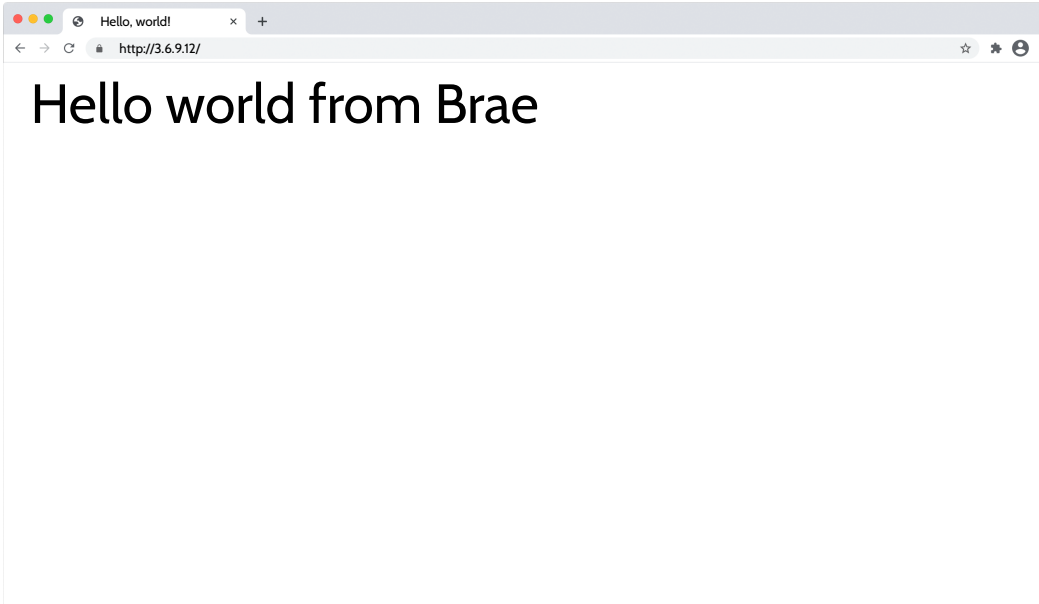
Demonstration

Let’s build ‘hello world’

Our Goal



My Goal



```
» cat hello-server.tf
```

```
resource "aws_instance" "hello-server" {
    ami = "ami-04902260ca3d33422"
    instance_type = "t2.micro"
}
```

```
» cat hello-server.tf
```

```
resource "aws_instance" "hello-server" {
    ami = "ami-04902260ca3d33422"
    instance_type = "t2.micro"

    user_data = file("${path.module}/setup.sh")
}
```

```
» cat setup.sh
```

```
#!/bin/bash
yum -y install httpd
systemctl enable httpd
systemctl start httpd
echo '<html><title>Hello, world!</title><h1>Hello world from Brae</h1></html>' > /
    var/www/html/index.html
```

```
» cat hello-server.tf
```

```
resource "aws_instance" "hello-server" {
    ami = "ami-04902260ca3d33422"
    instance_type = "t2.micro"

    user_data = file("${path.module}/setup.sh")

    associate_public_ip_address = true
    subnet_id = module.network.subnets[0].public.id
    vpc_security_group_ids = [
        module.network.http-port.id
    ]
}
```

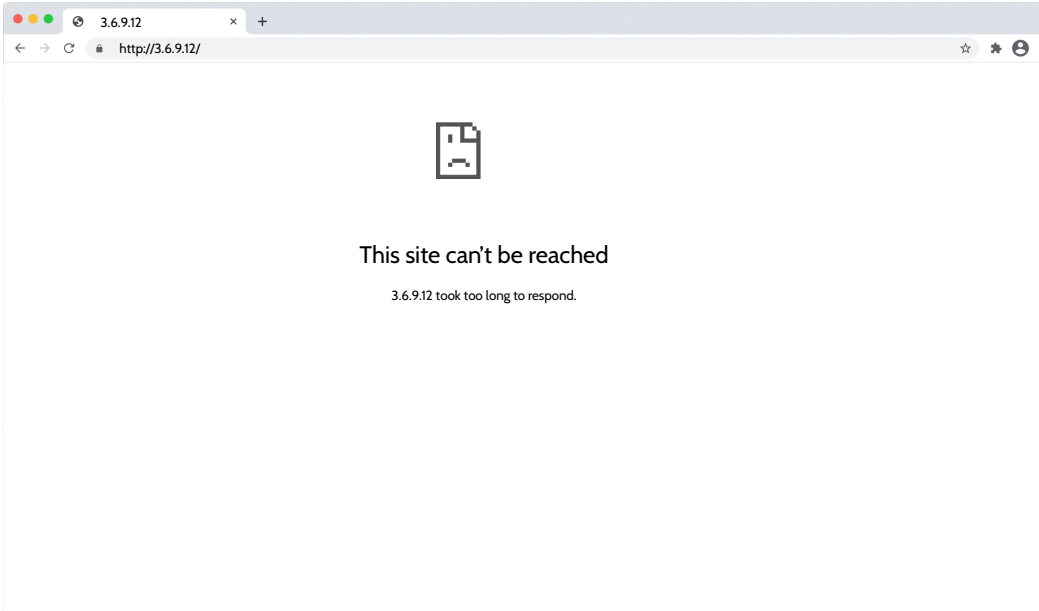
```
» cat hello-server.tf
```

```
resource "aws_instance" "hello-server" {
    ami = "ami-04902260ca3d33422"
    instance_type = "t2.micro"

    user_data = file("${path.module}/setup.sh")

    associate_public_ip_address = true
    subnet_id = module.network.subnets[0].public.id
    vpc_security_group_ids = [
        module.network.http-port.id
    ]

    tags = {
        Name = "hello-server"
    }
}
```

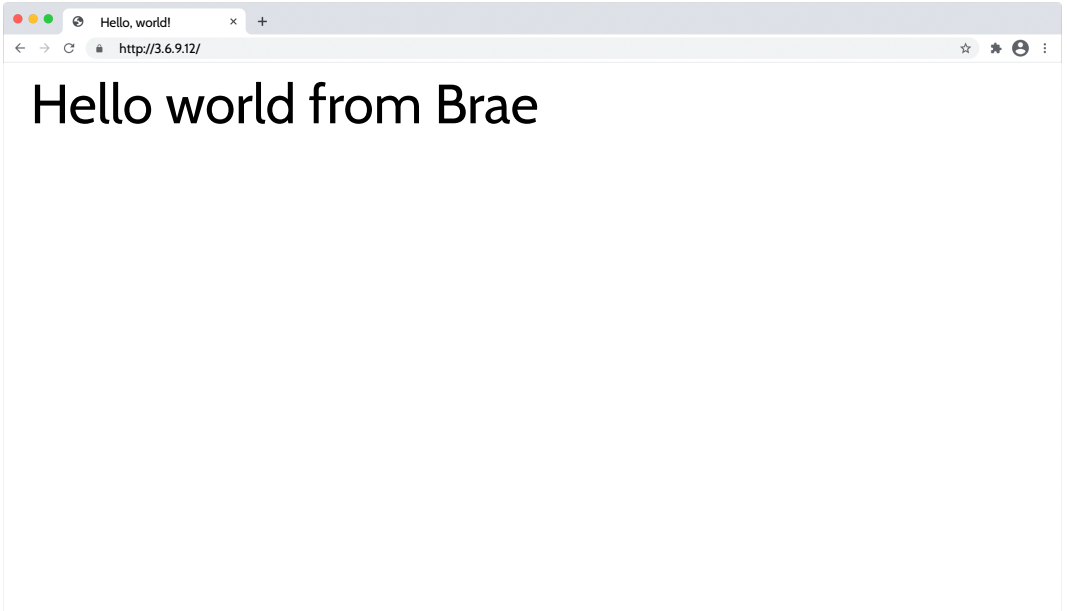
Starting the server

```
>> terraform init
>> terraform plan
>> terraform apply
```

Before


3.6.9.12
http://3.6.9.12/
This site can't be reached
3.6.9.12 took too long to respond.

After



Question

How much traffic can this website handle?

```
» cat stress-test.js
```

```
import http from 'k6/http';
import { check, sleep } from 'k6';

const IP = "http://3.6.9.12/";
export default function() {
    const res = http.get(IP);
    check(res, { 'status was 200': (r) => r.status == 200 });
    sleep(1);
}
```

```
» cat stress-test.js
```

```
import http from 'k6/http';
import { check, sleep } from 'k6';

const IP = "http://3.6.9.12/";
export const options = {
    stages: [
        { duration: '2m', target: 100 },
    ],
};
export default function() {
    const res = http.get(IP);
    check(res, { 'status was 200': (r) => r.status == 200 });
    sleep(1);
}
```

Run the tests

```
>> k6 run stress-test.js
```

Looks good so far

```
 status was 200
  100% - 347867 / 0

checks.....................: 100%
data_received..............: 100 MB 44 kB/s
data_sent..................: 27 MB 12 kB/s
iterations.................: 347997 152.552084/s
vus........................: 1 min=1 max=400
```

Let's upgrade the traffic

```
» cat stress-test.js
```

```
export const options = {
    stages: [
        { duration: '2m', target: 100 },
        { duration: '5m', target: 100 },
        { duration: '2m', target: 200 },
        { duration: '5m', target: 200 },
        { duration: '2m', target: 300 }, // around the breaking point
        { duration: '5m', target: 300 },
        { duration: '2m', target: 400 }, // beyond the breaking point
        { duration: '5m', target: 400 },
        { duration: '2m', target: 0 }, // scale down
    ],
};
```

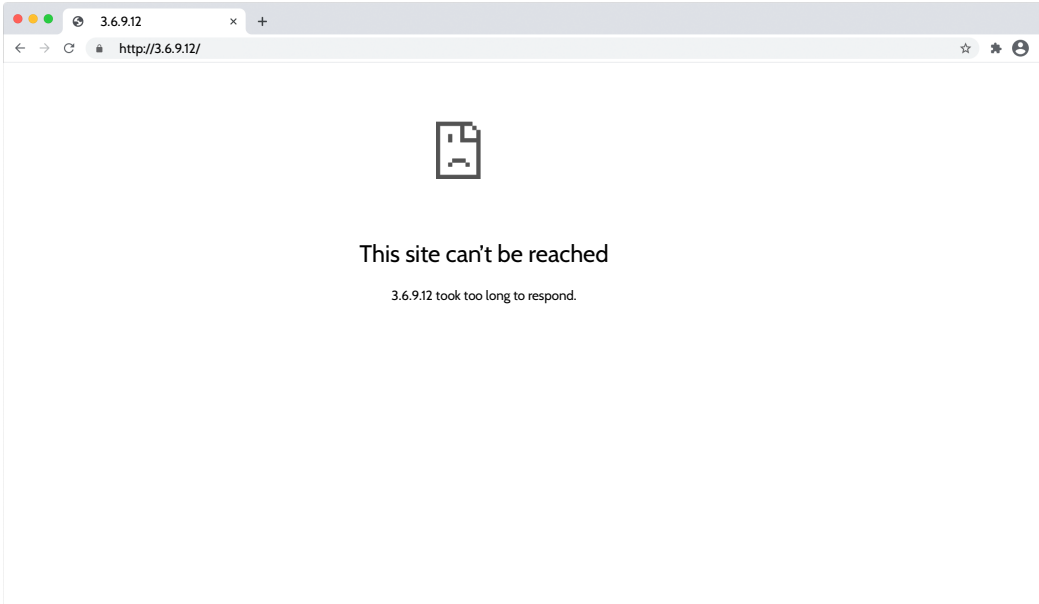
And run the tests again

```
>> k6 run stress-test.js
```

Oh no...

```
status was 200
99% - 347867 / 130

checks.....................: 99.96%
data_received..............: 100 MB 44 kB/s
data_sent..................: 27 MB 12 kB/s
iterations.................: 347997 152.552084/s
vus........................: 1 min=1 max=400
```

Back to square one



Question

How can we fix this?

Question

How can we fix this?

Answer

More servers?

```
» cat hello-server.tf
```

```
resource "aws_instance" "hello-server" {
    ami = "ami-04902260ca3d33422"
    instance_type = "t2.micro"

    user_data = file("${path.module}/setup.sh")

    associate_public_ip_address = true
    subnet_id = module.network.subnets[0].public.id
    vpc_security_group_ids = [
        module.network.http-port.id
    ]

    tags = {
        Name = "hello-server"
    }
}
```

```
» cat hello-scale.tf
```

```
resource "aws_instance" "hello-server" {
  count = 4

  ami = "ami-04902260ca3d33422"
  instance_type = "t2.micro"
  user_data = file("${path.module}/setup.sh")

  associate_public_ip_address = true
  subnet_id = module.network.subnets[count.index].public.id
  vpc_security_group_ids = [
    module.network.http-port.id
  ]

  tags = {
    Name = "hello-server-${count.index}"
  }
}
```

Definition 2. Target Group

A collection of EC2 instances.
More specifically, a collection of network connection points to EC2 instances.

An empty HTTP target group

```
» cat hello-scale.tf
```

```
resource "aws_lb_target_group" "hello-target" {
  name = "hello-target-group"
  port = 80
  protocol = "HTTP"
  vpc_id = module.network.vpc.id
}
```

Definition 3. Health Check

Monitors attributes of hardware or software to detect deficiencies.

Add a health check

```
» cat hello-scale.tf
```

```
resource "aws_lb_target_group" "hello-target" {
  name = "hello-target-group"
  port = 80
  protocol = "HTTP"
  vpc_id = module.network.vpc.id

  health_check {
    port = 80
    protocol = "HTTP"
    timeout = 5
    interval = 10
  }
}
```

Add our instances to the target group

```
» cat hello-scale.tf
```

```
resource "aws_lb_target_group_attachment" "hello-target-link" {
  count = length(aws_instance.hello-server)
  target_group_arn = aws_lb_target_group.hello-target.arn
  target_id = aws_instance.hello-server[count.index].id
  port = 80
}
```

Definition 4. Load Balancer

A networking tool to route and distribute traffic to targets.

Create a load balancer

```
» cat hello-scale.tf
```

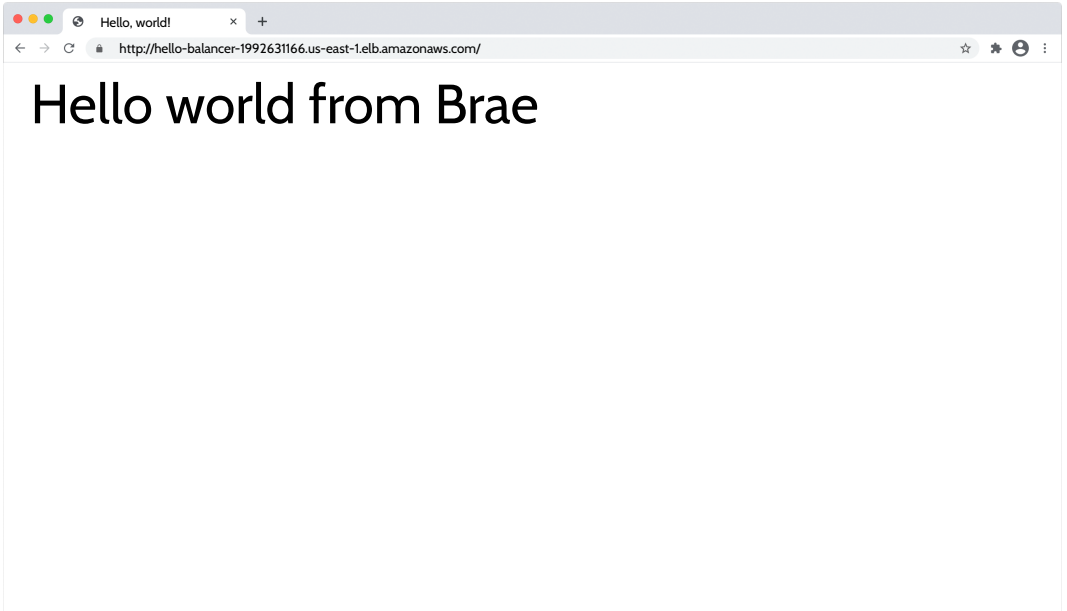
```
resource "aws_lb" "hello-balancer" {
  name = "hello-balancer"
  internal = false
  load_balancer_type = "application"
  subnets = [
    module.network.subnets[0].public.id,
    module.network.subnets[1].public.id,
    module.network.subnets[2].public.id,
    module.network.subnets[3].public.id
  ]
  security_groups = [
    module.network.http-port.id
  ]
}
```

Route load balancer traffic to the target group

```
» cat hello-scale.tf
```

```
resource "aws_lb_listener" "app" {
  load_balancer_arn = aws_lb.hello-balancer.arn
  port = "80"
  protocol = "HTTP"

  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.hello-target.arn
  }
}
```

We're live!


Hello, world!
http://hello-balancer-1992631166.us-east-1.elb.amazonaws.com/
Hello world from Brae

Exercise

Use *k6* to determine the new *load limits*

References

[1] Martin Fowler.
Software architecture guide.
https://martinfowler.com/architecture/, August 2019.

[2] Matthias Galster and Samuil Angelov.
What makes teaching software architecture difficult?
In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 356–359. Association for Computing Machinery, 2016.

[3] Robert C. Martin.
Design principles and design patterns.
https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf, 2000.
Accessed: 2022-01-10.

[4] Philippe Kruchten, Rafael Capilla, and Juan Carlos Duenas.
The decision view's role in software architecture practice.
IEEE software, 26(2):36–42, 2009.

[5] Michael Nygard.
Documenting architecture decisions.
`https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions`, November 2011.
Accessed: 2022-01-27.

[6] Olaf Zimmermann.
Architectural decisions – the making of.
`https://ozimmer.ch/practices/2020/04/27/ArchitectureDecisionMaking.html`, March 2021.
Accessed: 2022-02-02.

[7] Eli Perkins.
Why write adrs.
https://github.blog/2020-08-13-why-write-adrs/, August 2020.
Accessed: 2022-02-02.

[8] Eric Boersma.
7 application security principles you need to know.
https://www.cprime.com/resources/blog/
security-by-design-7-principles-you-need-to-know/, October 2020.

[9] Jerome H Saltzer and Michael D Schroeder.
The protection of information in computer systems.
Proceedings of the IEEE, 63(9):1278–1308, September 1975.

[10] Morrie Gasser.
Building a Secure Computer System, pages 35–44.
Van Nostrand Reinhold Company, January 1988.

[11] John Viega and Gary R McGraw.
Building Secure Software: How to Avoid Security Problems the Right Way, pages 91–113.
Addison-Wesley Professional, September 2001.

[12] Michael Howard and David LeBlanc.
Security Principles To Live By, page 64.
Microsoft Press Redmond, Wash., December 2002.

[13] Michael Gegick and Sean Barnum.
Failing securely.
`https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/failing-securely`, December 2005.

[14] Santosh Janardhan.
More details about the October 4 outage.
`https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/`, October 2021.

[15] C. E. Shannon.
Communication theory of secrecy systems.
The Bell System Technical Journal, 28(4):656–715, 1949.

[16] Sam Manjarres.
2021 world password day: How many will be stolen this year?
`https://www.secplicity.org/2021/05/04/2021-world-password-day-how-many-will-be-stolen-this-year/`, May 2021.

[17] Jerome H. Saltzer.
Protection and the control of information sharing in multics.
Communications of the ACM, 17(7):388–402, July 1974.

[18] Emma Roth.
Open source developer corrupts widely-used libraries, affecting tons of projects.
`https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected`, January 2022.

[19] Brian Foote and Joseph Yoder.
Big ball of mud.
Pattern languages of program design, 4:654–692, 1997.

[20] Gulsah.
How to avoid spaghetti code.
`https://tech.zensurance.com/posts/spaghetti-code`, November 2020.
note = "Accessed: 2022-02-18".

[21] Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.
In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[22] David J. DeWitt and Michael Stonebraker.
Mapreduce: A major step backwards.
`https://dsf.berkeley.edu/cs286/papers/backwards-vertica2008.pdf`, January 2008.

[23] henszey.
Smallest x86 ELF hello world.
http://timelessname.com/elfbin/.

[24] Changhui Xu.
Docker: From scratch.
https://codeburst.io/docker-from-scratch-2a84552470c8, July 2020.

[25] Philippe Kruchten.
Architectural blueprints — the '4+1' view model of software architecture.
IEEE software, 12(6):42–50, 1995.
https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

[26] Richard Thomas and Brae Webb.
Architectural views.
pages 42–50, February 2022.

[27] Alexander Shvets.
Observer.
https://refactoring.guru/design-patterns/observer.
note = "Accessed: 2022-02-18".

[28] Tom Hilburn, Alice Squires, Heidi Davidz, and Richard Turner.
Federal aviation administration (faa) advanced automation system (aas).
https://www.sebokwiki.org/wiki/Federal_Aviation_Administration_(FAA)_Advanced_Automation_System_(AAS), October 2021.
Example from the *Guide to the Systems Engineering Body of Knowledge*
https://www.sebokwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)&oldid=63222.

[29] Raymond Lister, Colin Fidge, and Donna Teague.
Further evidence of a relationship between explaining, tracing and writing skills in introductory programming.
In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, page 161–165, New York, NY, USA, 2009. Association for Computing Machinery.

[30] Raymond Lister.
Concrete and other neo-piagetian forms of reasoning in the novice programmer.
In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*, ACE '11, page 9–18, AUS, 2011. Australian Computer Society, Inc.

[31] Len Bass, Paul Clements, and Rick Kazman.
Software Architecture in Practice.
Addison-Wesley Professional, 3rd edition, September 2012.

[32] Len Bass, Paul Clements, and Rick Kazman.
Software Architecture in Practice.
Addison-Wesley, 4th edition, August 2021.

[33] David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson.
Documenting Software Architectures: Views and Beyond.
Addison-Wesley Professional, 2nd edition, 2010.

[34] Mark Richards and Neal Ford.
Fundamentals of Software Architecture: An Engineering Approach.
O'Reilly Media, Inc., January 2020.

[35] Sam Newman.
Building Microservices.
O'Reilly Media, Inc., February 2015.

[36] Simon Brown.
Software Architecture for Developers - Volume 2.
Leanpub, January 2022.
https://leanpub.com/visualising-software-architecture.

[37] Philippe Kruchten.
The Rational Unified Process: An Introduction.
Addison-Wesley Professional, 2004.

[38] Gerald M. Weinberg.
The Psychology of Computer Programming.
John Wiley & Sons, Inc., USA, 1985.

[39] Andrew Hoffman.
Web Application Security: Exploration and Countermeasures for Modern Web Applications.
O'Reilly Media, Inc., 2020.

[40] Leszek A. Maciaszek.
Requirements Analysis and System Design.
Addison-Wesley Harlow, 3rd edition, 2007.

[41] Architecture Capability Team.
NATO Architecture Framework.
NATO, 4th edition, September 2020.

[42] Nick Rozanski and Eóin Woods.
Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.
Addison-Wesley, 2nd edition, 2012.

[43] The Open Group Architecture Forum.
The Open Group Architecture Framework Standard.
The Open Group, 9.2 edition, 2018.
https://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html.

[44] *ISO/IEC/IEEE 42010:2011*.
ISO, 2011.

[45] *Unified Modeling Language*.
OMG, 2.5.1 edition, December 2017.
`https://www.uml.org/`.