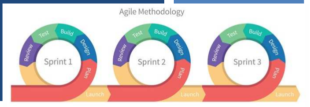
TCSS 360: SOFTWARE DEVELOPMENT AND QUALITY ASSURANCE

Introduction to Software Engineering

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma



FEEDBACK – JAN 4TH

- What point(s) remain least clear to you?
  - The difference between the waterfall and spiral software process models
  - Agile software processes
  - Software-as-a-Service: what does this mean in a practical application
  - Can you elaborate more on SaaS, provide examples
    - Forthcoming
  - Service Oriented Architecture Design
  - The overall project for TCSS 360
    - Details forthcoming –

OBJECTIVES

- Introduction to Software-as-a-Service
- Cloud computing
- Software quality
- Software productivity
- Software-as-a-Service architecture
- TCP/IP, HTTP, REST/JSON
- Examples

SOFTWARE-AS-A-SERVICE (SAAS) & SERVICE ORIENTED ARCHITECTURES (SOA)

WHY SAAS IS > SHRINK-WRAPPED SW…

1. No installation; Not worried about HW capabilities, OS versions, etc.
2. Data stored safely, persistently on (cloud) servers
3. Easy for groups to interact with same data
4. If data is large or changed frequently, simpler to support a master copy at a central site (cloud)
5. Cloud hosting → single HW/OS environment
   → no compatibility hassles for developers
   → incremental deployment: beta test new features with subset of the user base transparently
6. Cloud hosting → simplifies upgrades for developers, *and* no user upgrade requests

SAAS EXAMPLES

- Include server-centric apps, using thin clients
- Search, email, commerce, social nets, video…
- Now also productivity (Google Docs/Office 365), finance (TurboTax Online), IDEs (Codenvy)…

SHRINK WRAPPED SOFTWARE (SWS)

- Client-specific binary, frequent upgrades
- Must work w/many versions of HW, OS, Libraries…
- Hard to maintain
- Extensive compatibility testing per release

BUILDING LARGE SAAS

- Can you design software so that you can recombine independent modules to offer many different apps without a lot of programming?
  - Solves "Agile only good for small teams"

- "[Amazon CEO Jeff Bezos] realized long before the vast majority of Amazonians that Amazon needs to be a platform."

Steve Yegge, Googler, former Amazonian, in a 2011 blog post

2002 JEFF BEZOS EMAIL: AMAZON SERVICES MANDATE

1. "All teams will henceforth expose their data and functionality through service interfaces."
2. "Teams must communicate with each other through these interfaces."
3. "There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network."

AMAZON SW MANDATE – 2

4. "It doesn't matter what [API protocol] technology you use."
5. "Service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions."
6. "Anyone who doesn't do this will be fired."
7. "Thank you; have a nice day!"

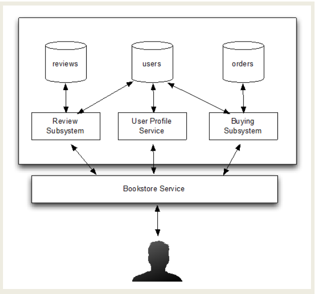
BOOKSTORE – SILO DESIGN

- Internal subsystems share data directly
  - DBs shared internally

- For example: Each subsystem accesses the users DB

- All subsystems composed together in a single API

  ("Bookstore")

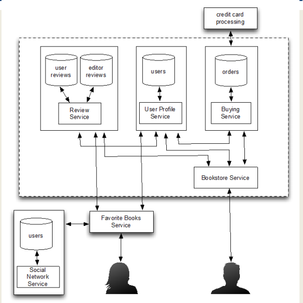


BOOKSTORE – SOA DESIGN

- Subsystems are independent, as if in separate datacenters

- Review Service access User Service API

- Can recombine to make new service ("Favorite Books")



*WHICH OF THE FOLLOWING IS A **DISADVANTAGE** OF SOA COMPARED TO A SILO DESIGN?*

- ❑ SOA may be harder to debug and tune
- ❑ SOA results in lower developer productivity
- ❑ SOA's complexity is a poor match for small teams
- ❑ SOA is more expensive to deploy than SILO as more servers are needed to handle the same workload

- SOA is harder to debug and tune: SOA can experience partial failures as a complete system consists of a number of microservices hosted separately, but composed together

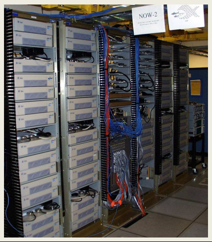
CLOUD COMPUTING

COMPUTER CLUSTERS

- Clusters: Commodity computers connected by Ethernet switches
  - More scalable than conventional servers
  - Much cheaper than conventional servers
  - Dependability through extensive redundancy
  - Few administrators for 1000s servers
- Careful selection of identical HW/SW
  - Interchangeable components
- Virtual Machine Monitors simplify operation

EARLY CLUSTER APPLICATION (1996)

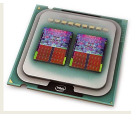
- Inktomi search engine on Network of Workstations (NOW) @ UCB in 1996



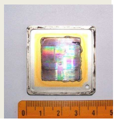
CLOUD COMPUTING NIST GENERAL DEFINITION

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications and services) that can be rapidly provisioned and reused with minimal management effort or service provider interaction"...

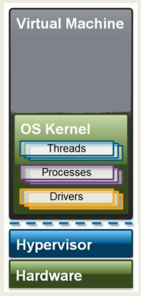
MICROPROCESSORS ADVANCEMENTS



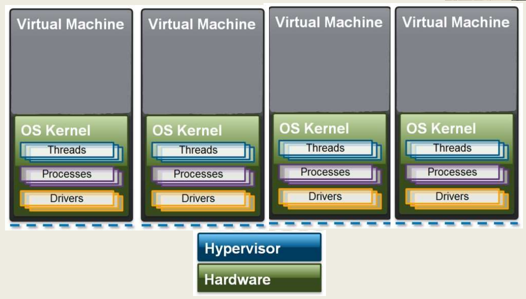
- Smaller die sizes (microns)
  - Lower voltages
  - Improved heat dissipation
  - Energy conservation
  - More transistors, but with similar clock rates
- How do we harness this new transistor density?
  - Multicore CPUs
  - Improve computational throughput
- How do we utilize many-core processors?



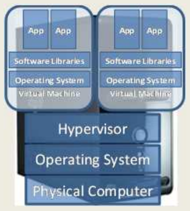
VIRTUALIZATION



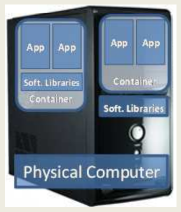
VIRTUALIZATION



CONTAINERIZATION

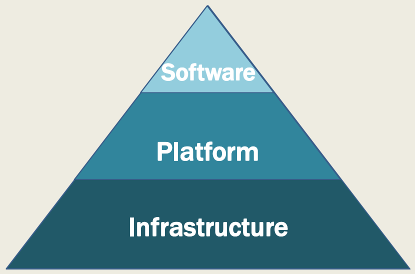


Virtualization

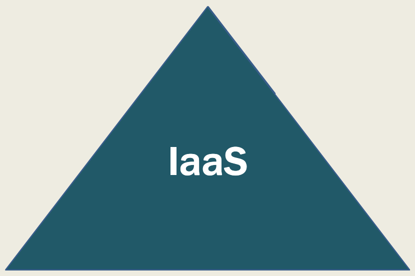


Containerization

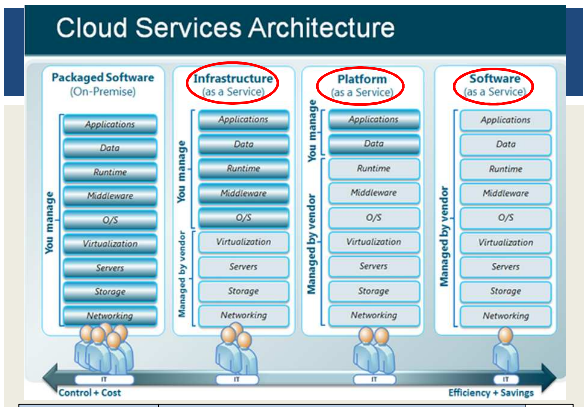
CLOUD COMPUTING STACK



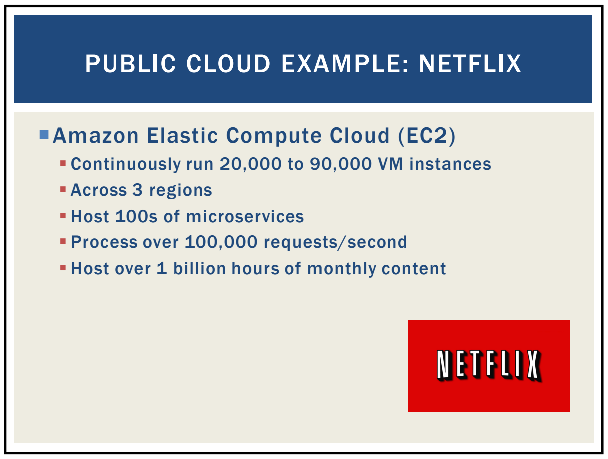
CLOUD COMPUTING STACK



Cloud Services Architecture



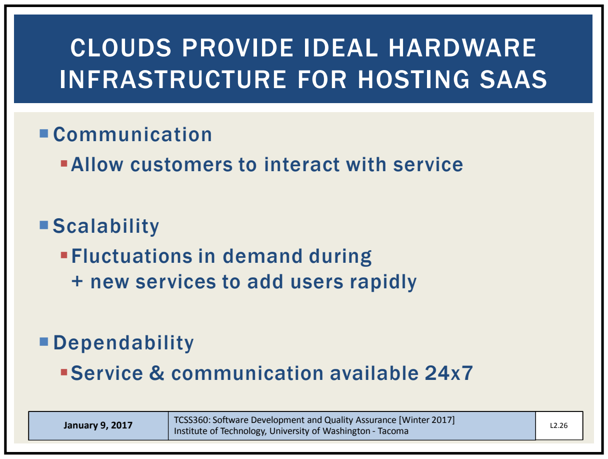
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.24



PUBLIC CLOUD EXAMPLE: NETFLIX

- Amazon Elastic Compute Cloud (EC2)
  - Continuously run 20,000 to 90,000 VM instances
  - Across 3 regions
  - Host 100s of microservices
  - Process over 100,000 requests/second
  - Host over 1 billion hours of monthly content

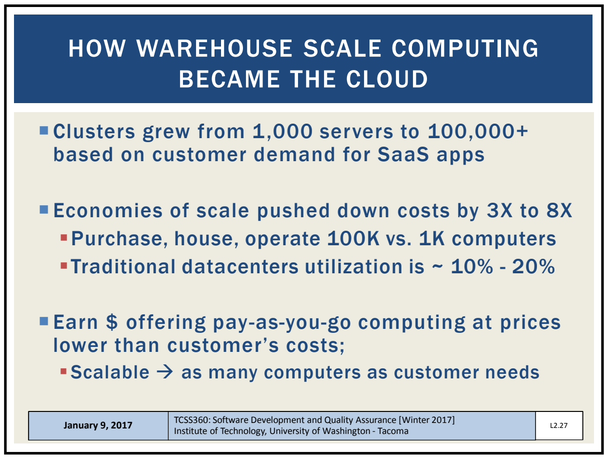
NETFLIX



CLOUDS PROVIDE IDEAL HARDWARE INFRASTRUCTURE FOR HOSTING SAAS

- Communication
  - Allow customers to interact with service
- Scalability
  - Fluctuations in demand during + new services to add users rapidly
- Dependability
  - Service & communication available 24x7

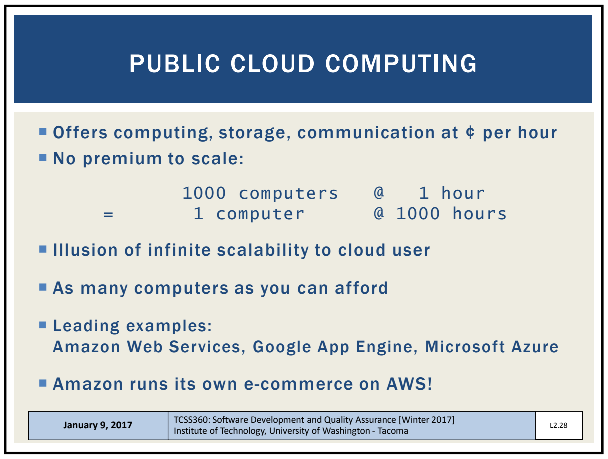
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.26



HOW WAREHOUSE SCALE COMPUTING BECAME THE CLOUD

- Clusters grew from 1,000 servers to 100,000+ based on customer demand for SaaS apps
- Economies of scale pushed down costs by 3X to 8X
  - Purchase, house, operate 100K vs. 1K computers
  - Traditional datacenters utilization is ~ 10% - 20%
- Earn $ offering pay-as-you-go computing at prices lower than customer's costs;
  - Scalable → as many computers as customer needs

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.27



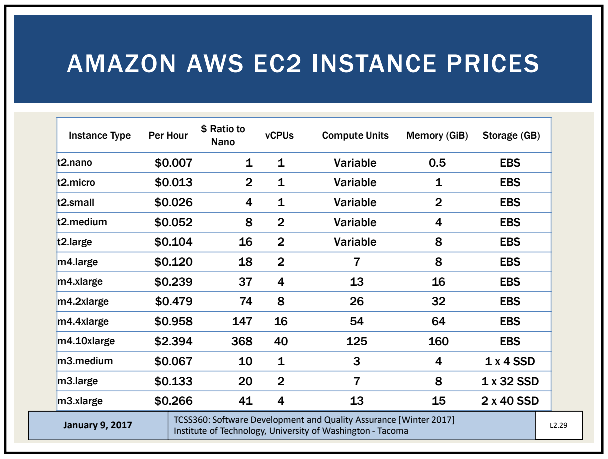
PUBLIC CLOUD COMPUTING

- Offers computing, storage, communication at ¢ per hour
- No premium to scale:

```
        1000 computers   @   1 hour
=         1 computer     @ 1000 hours
```

- Illusion of infinite scalability to cloud user
- As many computers as you can afford
- Leading examples:
  Amazon Web Services, Google App Engine, Microsoft Azure
- Amazon runs its own e-commerce on AWS!

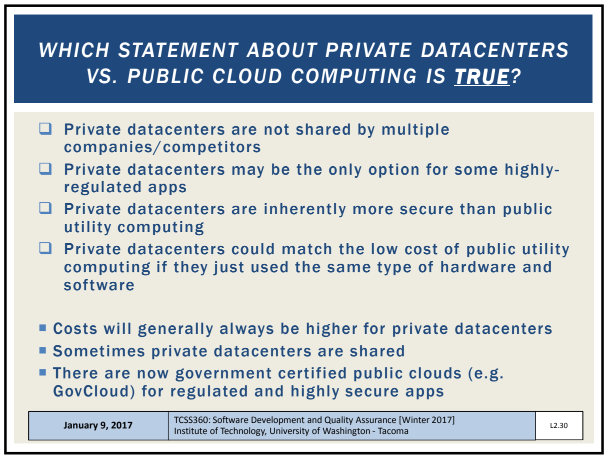
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.28



AMAZON AWS EC2 INSTANCE PRICES

Instance Type	Per Hour	$ Ratio to Nano	vCPUs	Compute Units	Memory (GiB)	Storage (GB)
t2.nano	$0.007	1	1	Variable	0.5	EBS
t2.micro	$0.013	2	1	Variable	1	EBS
t2.small	$0.026	4	1	Variable	2	EBS
t2.medium	$0.052	8	2	Variable	4	EBS
t2.large	$0.104	16	2	Variable	8	EBS
m4.large	$0.120	18	2	7	8	EBS
m4.xlarge	$0.239	37	4	13	16	EBS
m4.2xlarge	$0.479	74	8	26	32	EBS
m4.4xlarge	$0.958	147	16	54	64	EBS
m4.10xlarge	$2.394	368	40	125	160	EBS
m3.medium	$0.067	10	1	3	4	1 x 4 SSD
m3.large	$0.133	20	2	7	8	1 x 32 SSD
m3.xlarge	$0.266	41	4	13	15	2 x 40 SSD

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.29

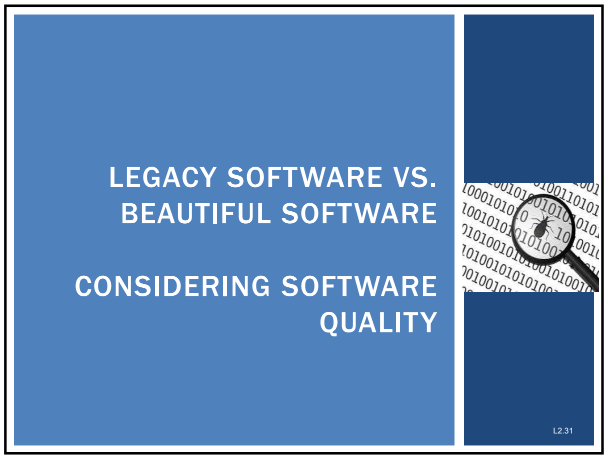


*WHICH STATEMENT ABOUT PRIVATE DATACENTERS VS. PUBLIC CLOUD COMPUTING IS **TRUE**?*

- ❑ Private datacenters are not shared by multiple companies/competitors
- ❑ Private datacenters may be the only option for some highly-regulated apps
- ❑ Private datacenters are inherently more secure than public utility computing
- ❑ Private datacenters could match the low cost of public utility computing if they just used the same type of hardware and software

- Costs will generally always be higher for private datacenters
- Sometimes private datacenters are shared
- There are now government certified public clouds (e.g. GovCloud) for regulated and highly secure apps

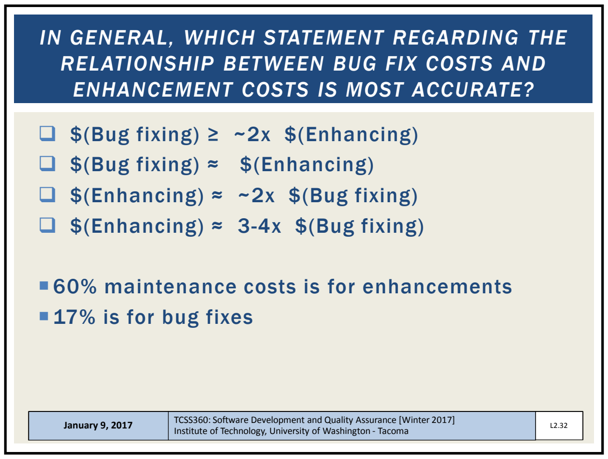
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.30



LEGACY SOFTWARE VS. BEAUTIFUL SOFTWARE

CONSIDERING SOFTWARE QUALITY

L2.31

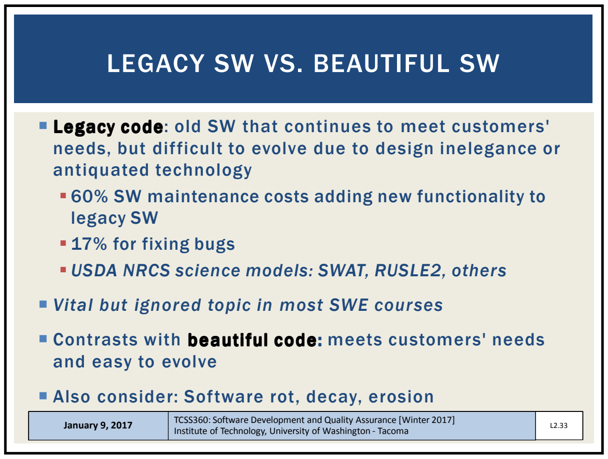


IN GENERAL, WHICH STATEMENT REGARDING THE RELATIONSHIP BETWEEN BUG FIX COSTS AND ENHANCEMENT COSTS IS MOST ACCURATE?

- ❑ \$(Bug fixing) ≥ ~2x \$(Enhancing)
- ❑ \$(Bug fixing) ≈ \$(Enhancing)
- ❑ \$(Enhancing) ≈ ~2x \$(Bug fixing)
- ❑ \$(Enhancing) ≈ 3-4x \$(Bug fixing)

- 60% maintenance costs is for enhancements
- 17% is for bug fixes

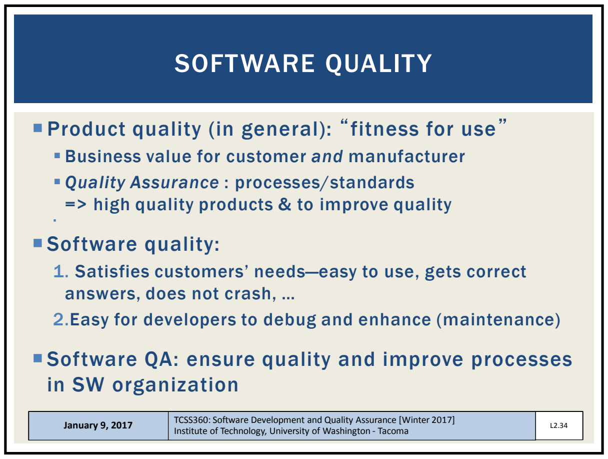
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.32



LEGACY SW VS. BEAUTIFUL SW

- **Legacy code**: old SW that continues to meet customers' needs, but difficult to evolve due to design inelegance or antiquated technology
  - 60% SW maintenance costs adding new functionality to legacy SW
  - 17% for fixing bugs
  - *USDA NRCS science models: SWAT, RUSLE2, others*
- *Vital but ignored topic in most SWE courses*
- Contrasts with **beautiful code:** meets customers' needs and easy to evolve
- Also consider: Software rot, decay, erosion

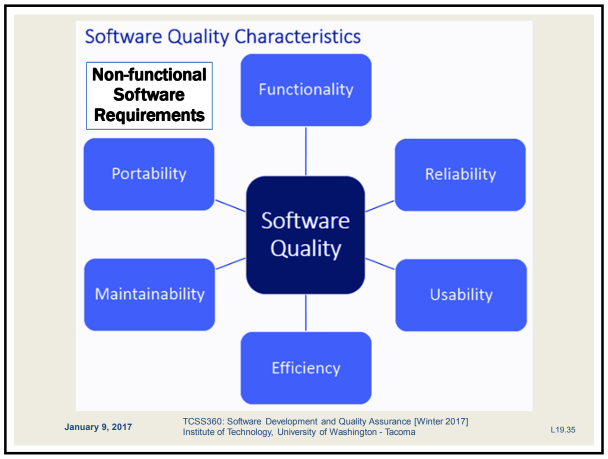
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.33



SOFTWARE QUALITY

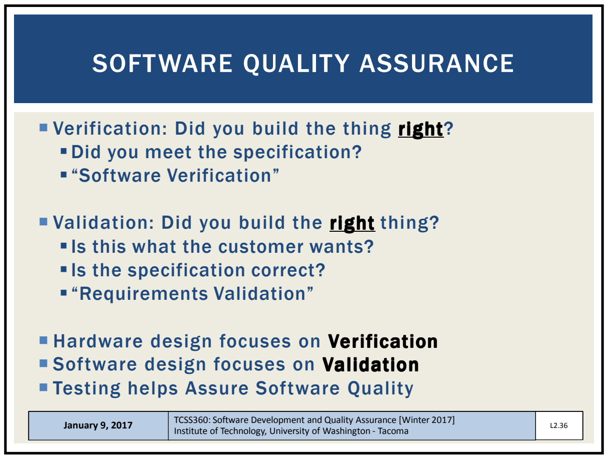
- Product quality (in general): "fitness for use"
  - Business value for customer *and* manufacturer
  - *Quality Assurance* : processes/standards => high quality products & to improve quality
- Software quality:
  1. Satisfies customers' needs—easy to use, gets correct answers, does not crash, …
  2. Easy for developers to debug and enhance (maintenance)
- Software QA: ensure quality and improve processes in SW organization

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.34



Software Quality Characteristics

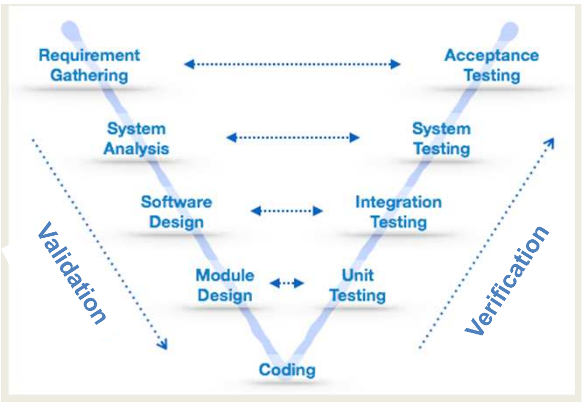
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L19.35



SOFTWARE QUALITY ASSURANCE

- Verification: Did you build the thing **right**?
  - Did you meet the specification?
  - "Software Verification"
- Validation: Did you build the **right** thing?
  - Is this what the customer wants?
  - Is the specification correct?
  - "Requirements Validation"
- Hardware design focuses on **Verification**
- Software design focuses on **Validation**
- Testing helps Assure Software Quality

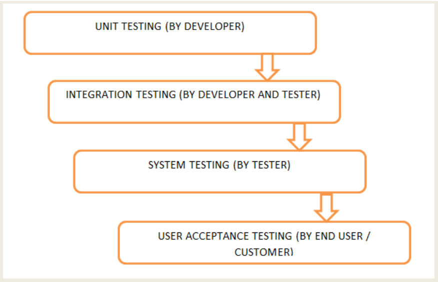
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.36



EXHAUSTIVE TESTING IS INFEASIBLE

- Divide and conquer: perform different tests at different phases of SW development
  - Higher level tests don't repeat lower level tests
- Coverage: various measurements of what % of the system is "exercised" by a test suite. (% of Lines of Code LOC)
- **System or acceptance test**: Check if the integrated program meets requirements, expectations (***scope creep***)
- **Integration test**: Test if interfaces between units communicate correctly. ***Do the parts work together?***
- **Module or functional test**: test individual units
- **Unit test**: test a single method, does it do what is expected?

LEVELS OF TESTING


UNIT TESTING (BY DEVELOPER)
INTEGRATION TESTING (BY DEVELOPER AND TESTER)
SYSTEM TESTING (BY TESTER)
USER ACCEPTANCE TESTING (BY END USER / CUSTOMER)

WHAT STATEMENT IS __NOT TRUE__ ABOUT TESTING?

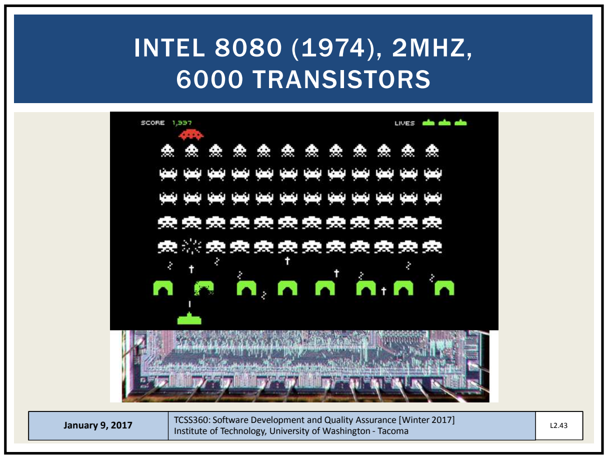
- ❑ Tests that have outlived their usefulness should be discarded
- ❑ While difficult to achieve, 100% test coverage ensures customer requirements have been fulfilled.
- ❑ Higher level tests typically delegate more detailed testing to lower levels
- ❑ Unit testing works within a single class and module testing works across classes

- 100% test coverage ensures design reliability (verification) but not that we've built the right product (validation)

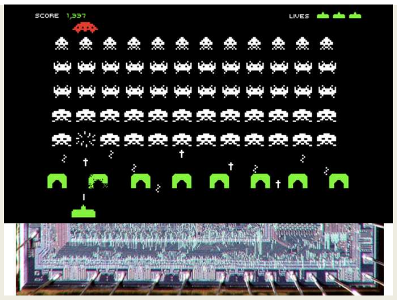
IN SUMMARY...

- **Agile vs. Plan & Document**: Small teams, quick iterations w/customer feedback, to reduce risk of building the wrong thing (validation)
- **Service Oriented Architecture**: build large systems by combining smaller standalone services → reuse!
- **Cloud Computing**: Economies of scale of large computer warehouses led to cost savings & higher server (CPU) utilization, enabling reduced costs for everyone
  - ***Utility Computing***
- **Testing**: to assure software quality, including verification (code) and validation (requirements) is good for all stakeholders (customers and developers)

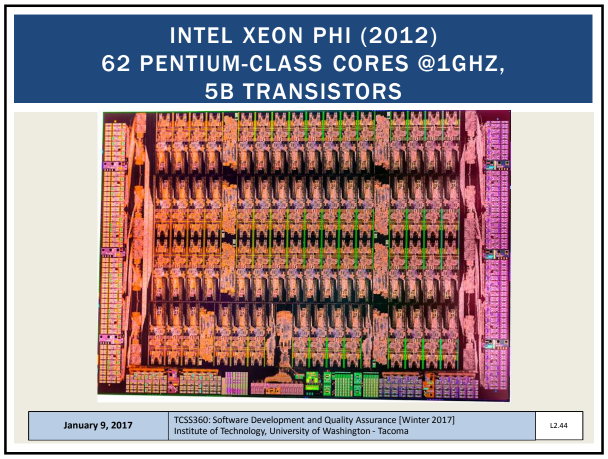
SOFTWARE PRODUCTIVITY



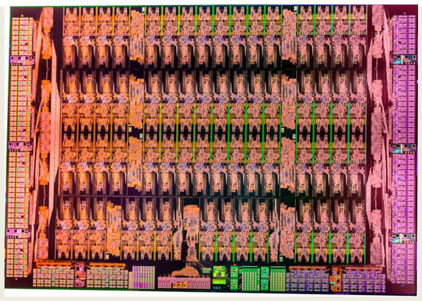
INTEL 8080 (1974), 2MHZ, 6000 TRANSISTORS



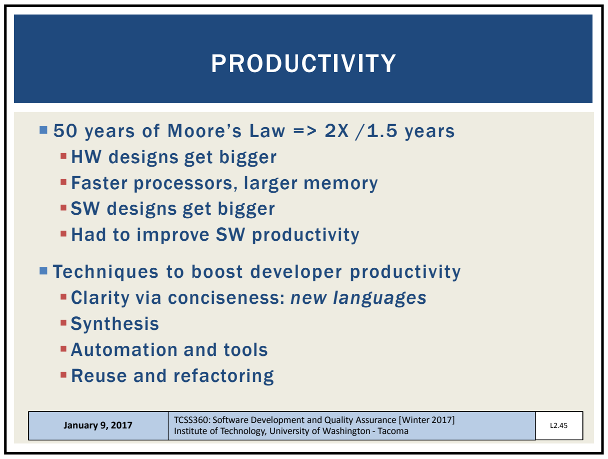
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.43



INTEL XEON PHI (2012) 62 PENTIUM-CLASS CORES @1GHZ, 5B TRANSISTORS



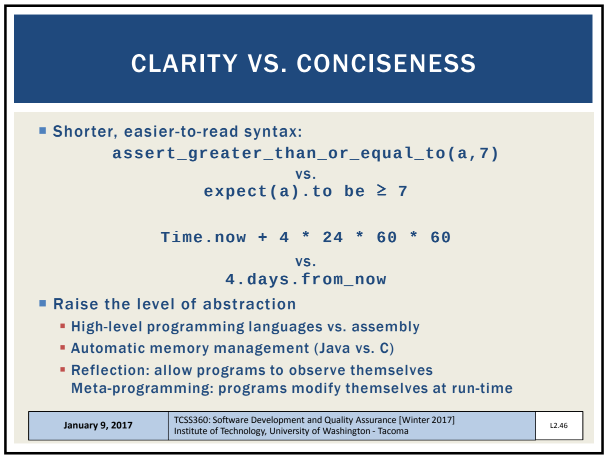
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.44



PRODUCTIVITY

- 50 years of Moore's Law => 2X /1.5 years
  - HW designs get bigger
  - Faster processors, larger memory
  - SW designs get bigger
  - Had to improve SW productivity
- Techniques to boost developer productivity
  - Clarity via conciseness: *new languages*
  - Synthesis
  - Automation and tools
  - Reuse and refactoring

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.45



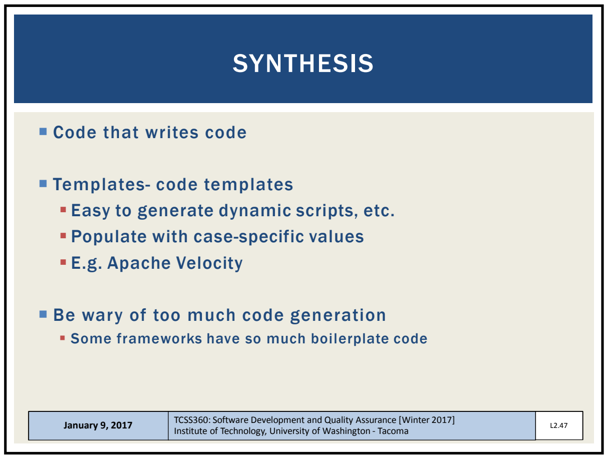
CLARITY VS. CONCISENESS

- Shorter, easier-to-read syntax:

```
assert_greater_than_or_equal_to(a,7)
                 vs.
        expect(a).to be ≥ 7

  Time.now + 4 * 24 * 60 * 60
                 vs.
          4.days.from_now
```

- Raise the level of abstraction
  - High-level programming languages vs. assembly
  - Automatic memory management (Java vs. C)
  - Reflection: allow programs to observe themselves
    Meta-programming: programs modify themselves at run-time

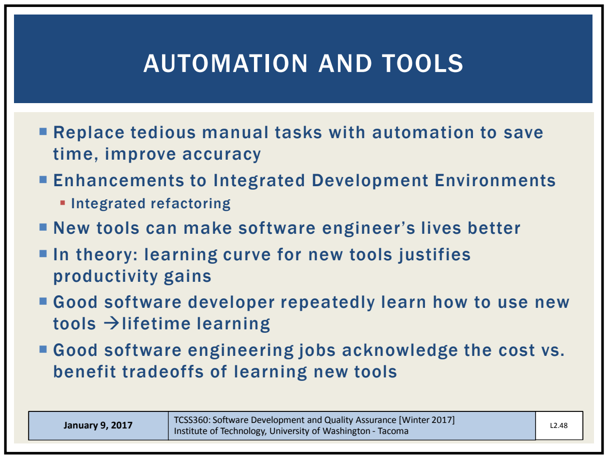
January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.46



SYNTHESIS

- Code that writes code
- Templates- code templates
  - Easy to generate dynamic scripts, etc.
  - Populate with case-specific values
  - E.g. Apache Velocity
- Be wary of too much code generation
  - Some frameworks have so much boilerplate code

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.47



AUTOMATION AND TOOLS

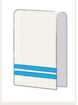
- Replace tedious manual tasks with automation to save time, improve accuracy
- Enhancements to Integrated Development Environments
  - Integrated refactoring
- New tools can make software engineer's lives better
- In theory: learning curve for new tools justifies productivity gains
- Good software developer repeatedly learn how to use new tools →lifetime learning
- Good software engineering jobs acknowledge the cost vs. benefit tradeoffs of learning new tools

January 9, 2017 | TCSS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma | L2.48

REUSE

- How do these help with reuse?
  - Procedures and functions
  - Object oriented programming
  - Software libraries
  - Programming frameworks
  - Design patterns

REFACTORING

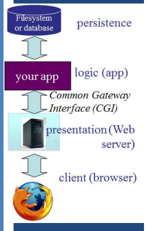


- Don't Repeat Yourself → D.R.Y. principle
  - *"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."*
- Consider maintenance on a system with copied code vs. applying the same repair with DRY coding
  - Over time, can we still recognize the copies?
    - Software decay, software grime
  - How do we know we all the instances requiring change?
- Best practice:
  Refactor code to extract commonality

IN SUMMARY…

- **Computer Engineering**: Has delivered more than a **6x order of magnitude** expansion of the number of transistors on a CPU since 1974.
  6,000 → 7,200,000,000
- **Software Engineering**: Harness new languages, code-generation, more automation and improved development tools, reuse and refactoring to develop software to use modern computers and cloud systems.

CHAPTER 2. THE ARCHITECTURE OF SAAS APPLICATIONS



SAAS ARCHITECTURE

- Key terms
  - **Software architecture**
    - How software subsystems interconnect to meet application functional and non-functional requirements
  - **Design patterns**
    - A general architectural solution for a family of similar problems
    - Design patters emerge when studying common solutions to architectural design

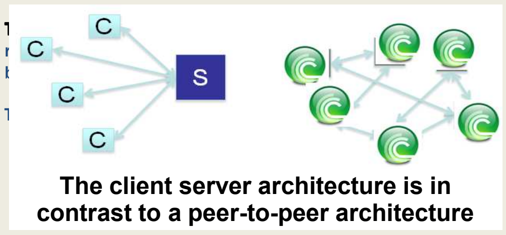
DESIGN PATTERNS

- The "Gang of Four" book
- Object-oriented design patterns
- Seminal book on OO patterns
- Now somewhat dated, examples is C++ 98
- Patterns can be applied in any OO language
- Many things are called "patterns", but not all are OO design patterns

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

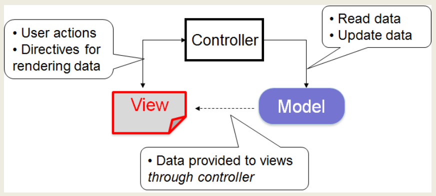
KEY TERMS - 2

- **Client-server**: A client makes requests, a server responds to the request of many clients
- T
  r
  t
- T



The client server architecture is in contrast to a peer-to-peer architecture

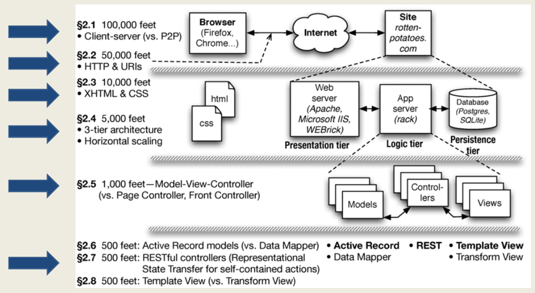
MODEL VIEW CONTROLLER


• User actions
• Directives for rendering data
Controller
• Read data
• Update data
View
Model
• Data provided to views through controller

MODEL VIEW CONTROLLER - 2

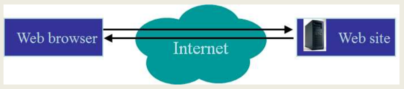
- The key to MVC is the **separation** of tiers
  - Allows many "presentations" to be created from one model
  - Controller (services) are reused
  - GUI does not depend on controller, and vice-a-versa
  - Tiers are hosted individually using different computer infrastructure
    - Model: database server(s)
    - Controller: web application server(s): Tomcat, Glassfish
    - View: client's web browser

SAAS ARCHITECTURE


§2.1 100,000 feet
• Client-server (vs. P2P)
Browser (Firefox, Chrome...)
Internet
Site rotten-potatoes.com
§2.2 50,000 feet
• HTTP & URIs
§2.3 10,000 feet
• XHTML & CSS
html
css
Web server (Apache, Microsoft IIS, WEBrick)
App server (rack)
Database (Postgres, SQLite)
§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling
Presentation tier
Logic tier
Persistence tier
§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller)
Models
Control-lers
Views
§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)
• Active Record
• Data Mapper
• REST
• Template View
• Transform View

100,000 FEET

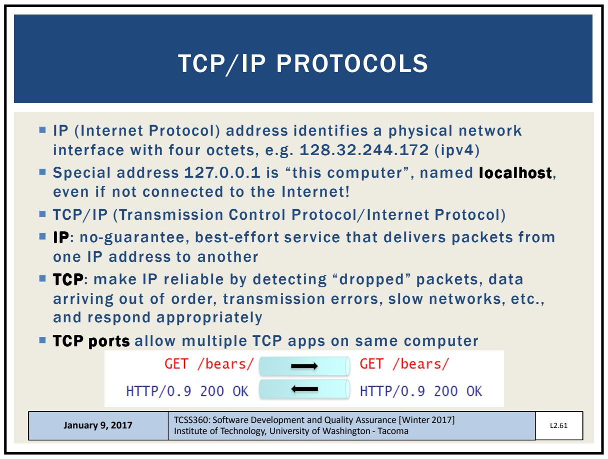
- The web consists primarily of a client/server architecture
- Web clients make requests, servers reply



- This architecture has proliferated
  - Smart phones
  - Tablets

PROLIFERATION OF CLIENTS AND DEVICES

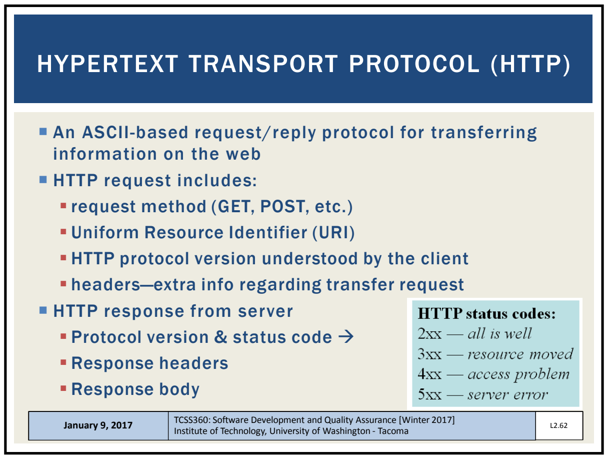
A Momentous Transition
Today's Dominant Endpoints
Dominant Endpoints in 2025
Transportation and Connected Vehicles
Precision Agriculture
Industrial Automation
Healthcare
Smart Grid
Intelligent Buildings
2 to 3 billion
A person behind every device
1 trillion
Devices organized as systems



TCP/IP PROTOCOLS

- IP (Internet Protocol) address identifies a physical network interface with four octets, e.g. 128.32.244.172 (ipv4)
- Special address 127.0.0.1 is "this computer", named **localhost**, even if not connected to the Internet!
- TCP/IP (Transmission Control Protocol/Internet Protocol)
- **IP**: no-guarantee, best-effort service that delivers packets from one IP address to another
- **TCP**: make IP reliable by detecting "dropped" packets, data arriving out of order, transmission errors, slow networks, etc., and respond appropriately
- **TCP ports** allow multiple TCP apps on same computer

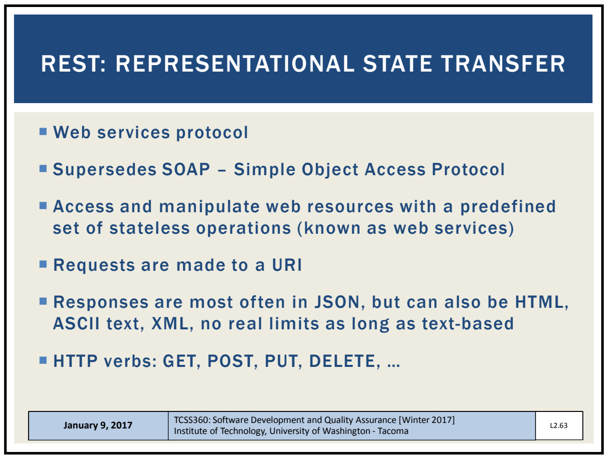
```
GET /bears/          ⟶   GET /bears/
HTTP/0.9 200 OK      ⟵   HTTP/0.9 200 OK
```



HYPERTEXT TRANSPORT PROTOCOL (HTTP)

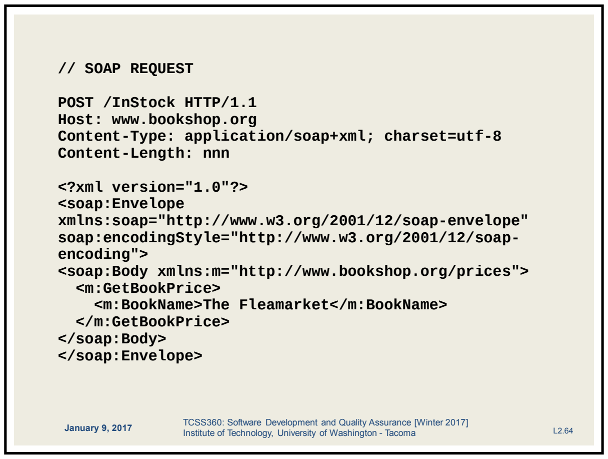
- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

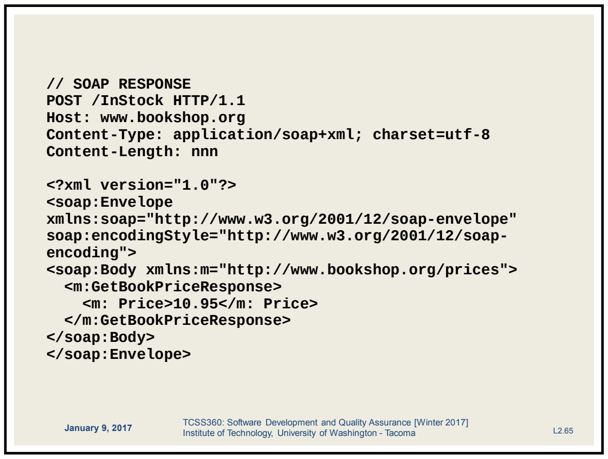


REST: REPRESENTATIONAL STATE TRANSFER

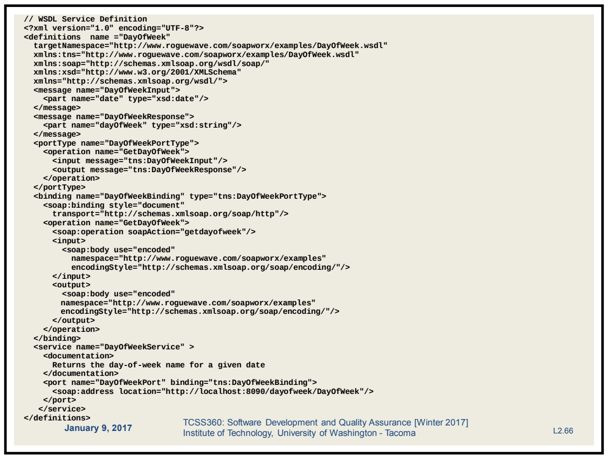
- Web services protocol
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Requests are made to a URI
- Responses are most often in JSON, but can also be HTML, ASCII text, XML, no real limits as long as text-based
- HTTP verbs: GET, POST, PUT, DELETE, …



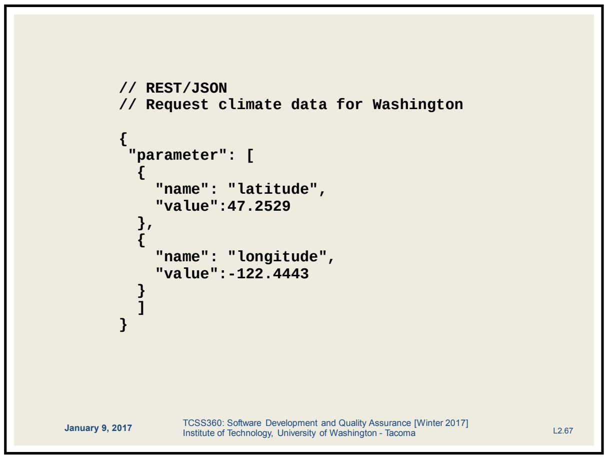
```
// SOAP REQUEST

POST /InStock HTTP/1.1
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:Body xmlns:m="http://www.bookshop.org/prices">
  <m:GetBookPrice>
    <m:BookName>The Fleamarket</m:BookName>
  </m:GetBookPrice>
</soap:Body>
</soap:Envelope>
```



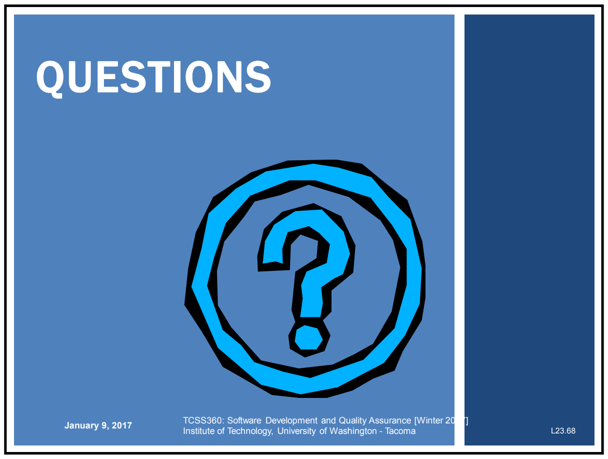
```
// SOAP RESPONSE
POST /InStock HTTP/1.1
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:Body xmlns:m="http://www.bookshop.org/prices">
  <m:GetBookPriceResponse>
    <m: Price>10.95</m: Price>
  </m:GetBookPriceResponse>
</soap:Body>
</soap:Envelope>
```



```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
   </service>
</definitions>
```



```
// REST/JSON
// Request climate data for Washington

{
 "parameter": [
  {
    "name": "latitude",
    "value":47.2529
  },
  {
    "name": "longitude",
    "value":-122.4443
  }
  ]
}
```



QUESTIONS