Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
7. Productivity: Conciseness, Synthesis, Reuse, and Tools

Ranking Top 200 Jobs (2012)

1. Software Engineer
28. Civil Engineer
38. Nurse
40. Physician
47. Accountant
60. Mechanical Engineer
73. Electrical Engineer
87. Attorney
104. Airline Pilot
133. Fashion Designer
137. High School Teacher
163. Police Officer
173. Flight Attendant
185. Firefighter
196. Newspaper Reporter
200. Lumberjack

InformationWeek 5/15/12. Based on salary, stress levels, hiring outlook, physical demands, and work environment (www.careercast.com)
If SENG so popular, why so many SWE disasters?

(Search Wikipedia for details)
- Therac-25 lethal radiation overdose
  - 1985: Reused SW from machine with HW interlock on machine without it: SW bug => 3 died
- Mars Climate Orbiter disintegration
  - 1999: SW used wrong units (pound-seconds vs. newton-seconds) => wasted $325M
- FBI Virtual Case File project abandonment
  - 2005: give up after 5 years of work => wasted $170M
- Ariane 5 rocket explosion
  - 1996 => wasted $370M

Another Software Production Disaster!

Software Production has a Poor Track Record

Example: Space Shuttle Software
- Cost: $10 Billion, millions of dollars more than planned
- Time: 3 years late
- Quality: First launch of Columbia was cancelled because of a synchronization problem with the Shuttle's 5 onboard computers.
  - Error was traced back to a change made 2 years earlier when a programmer changed a delay factor in an interrupt handler from 50 to 60 milliseconds.
  - The likelihood of the error was small enough, that the error caused no harm during thousands of hours of testing.
- Substantial errors still exist.
  - Astronauts are supplied with a book of known software problems "Program Notes and Waivers".

Typical Project Outcomes

Standish Group Survey, 1999 (from a survey of 8000 business systems projects).

Many Software Projects still Fail!

- The right combination of technologies, SE and OOAD principles are increasingly getting important in software development especially for large projects.
- Although project management techniques, software development methodologies, design patterns, development, testing and architectural modeling techniques and tools have developed in the last decade; many software projects still fail and the percentage of successful projects completed on-time and on budget is still very low.
- The Standish Group's “Chaos Report” in 1994 reported that only 16.2% of software projects were completed on-time and on-budget.
- In 2004, 29% of projects completed on-time and on-budget, with required features and functions.
- Although the improvement is significant, it is dismal when compared with traditional engineering disciplines, such as architecture or electrical engineering.
Software Development Difficult - Why?

- The problem domain (also called application domain) is difficult
- The solution domain is difficult
- The development process is difficult to manage
- Software offers extreme flexibility
- Software is a discrete system
  - Continuous systems have no hidden surprises
  - Discrete systems can have hidden surprises! (Parnas)

David Lorge Parnas is an early pioneer in software engineering who developed the concepts of modularity and information hiding in systems which are the foundation of object oriented methodologies.

Factors Affecting the Quality of a Software System

- **Complexity**
  - The system is so complex that no single programmer can understand it anymore
  - The introduction of one bug fix causes another bug

- **Change**
  - The “Entropy” of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive (“Second Law of Software Dynamics”).
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

Most Common Sources of Project Failures

- Incomplete or changing requirements (%34) (LACK OF PROCESS)
- Poor project planning/management (%33) (LACK OF PROCESS)
- Lack of Resources (11%)
- Didn’t Need Software Any Longer (8%)
- Technology Illiteracy and naive adoption of new technology (4%)
- Others (%10)

Software Engineering

- **Software Engineering (SENG)** means the construction of quality software with a limited budget and a given deadline in the context of constant change.
- **Complex**
- Impossible to understand by a single person
  - Many projects are never finished: “vaporware” (examples)
  - The problem is arbitrary complexity
- Emphasis is on both, on software and on engineering

Engineer: build a product using off-the-shelf components and integrating them under time and budget constraints. ill-defined problems, partial solutions, and has to rely on empirical methods to evaluate solutions.
Why Software Engineering?

- Important to distinguish
  - "easy/small" systems (one/less developer(s), one user/less users, experimental use only) from
  - "hard/large" systems (multiple developers, multiple users, products)

- Experience with "easy/small" systems is misleading
  - One person/small team techniques do not scale up

- Analogy with bridge building:
  - Over a stream = easy, one person job
  - Over River Nile ... ? (the techniques do not scale)

What is Software Engineering?

- Applying engineering principles to software development

- Build software like we build bridges!

- IEEE Definition:
  "Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software"

Software Engineering is

Modeling Activity (Modeling with UML, …)

Problem-Solving activity:

- Engineering is a problem solving activity.
- SE is an engineering activity.
- It requires experimentation, the reuse of pattern solutions, and the incremental evolution of the system toward a solution.
- What makes SE different from problem solving in other sciences is that change occurs while the problem is being solved.
What is Software Engineering?

Software engineering is

- **Knowledge acquisition (KA) activity:**
  - KA is a **nonlinear activity** especially in SE!!
  - The addition of a new piece of information may invalidate all the knowledge we have acquired for the understanding of a system.
  - Even if we had already documented this understanding in documents and code (“The system is 90% coded, we will be done next week”), we must mentally be prepared to **start from scratch**.

Rationale-driven activity:

- **Capture the context** in which decisions were made and the rationale behind these decisions.

Notations, Methods and Methodologies

- **Notation:** A graphical or textual set of rules for representing a model. (UML, …)

- **Method:** Repeatable technique for solving a specific problem. (IEEE SENG standards, …)

- **Methodology (Process):** Collection of methods for solving a class of problems. RUP (Rational Unified Process), XP (Extreme Programming), SCRUM, CMMI Level 2 Project Management Processes, …

Software Engineering: A Problem Solving Activity

- **Analysis:** Understand the nature of the problem and break the problem into pieces
- **Synthesis:** Put the pieces together into a large structure

For problem solving we use

- **Techniques (methods):**
  - Formal procedures for producing results using some well-defined notation
- **Methodologies:**
  - Collection of techniques applied across software development and unified by a philosophical approach
- **Tools:**
  - Instrument or automated systems to accomplish a technique

Software Engineering: Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

- a high quality software system
- with a given budget
- before a given deadline

while change occurs.
Scientist vs Engineer

- **Computer Scientist**
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time...

- **Engineer**
  - Develops a solution for an application-specific problem for a client
  - Uses computers & languages, tools, techniques and methods

- **Software Engineer**
  - Works in multiple application domains
  - Has only limited time ...
  - ...while changes occur in requirements and available technology

Introduction to SaaS and Agile Development

1. What is Software Engineering?

2. **Software as a Service**

3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
7. Productivity: Conciseness, Synthesis, Reuse, and Tools

Software Targets

- **Traditional SW**: binary code installed and runs wholly on client device, which users must upgrade repeatedly
  - Must work with many versions of hardware, many versions of OS
  - New versions must go through extensive release cycle to ensure compatibility
- An alternative where develop SW that only needs to work on one HW & OS platform?
  - Quicker release cycle, no user upgrades?

**Software as a Service: SaaS**

- SaaS delivers SW & data as service over Internet via thin program (e.g., browser) running on client device
  - Search, social networking, video
- Now also SaaS version of traditional SW
  - E.g., Microsoft Office 365, TurboTax Online
- Instructors think SaaS is revolutionary, the future of virtually all software
6 Reasons for SaaS

1. No install worries about HW capability, OS.
2. No worries about data loss (data is remote).
3. Easy for groups to interact with same data.
4. If data is large or changed frequently, simpler to keep 1 copy at central site.
5. 1 copy of SW, single HW/OS environment => no compatibility hassles for developers => beta test new features on 1% of users.
6. 1 copy => simplifies upgrades for developers and no user upgrade requests.

SaaS Loves Rails

- Many frameworks/languages for SaaS
  - Django/Python, Zend/PHP, Spring/Java
- We use Ruby on Rails ("Rails")
- Rails popular programming framework that uses Ruby – e.g., Twitter
- Ruby, a modern scripting language: object oriented, functional, automatic memory management, dynamic types, reuse via mix-ins & closures, synthesis via metaprogramming

Why take time for Ruby/Rails?

- 14 weeks to learn:
  - Part time (taking 3 other classes or full time job)
- Only hope is highly productive language, tools, framework: We (A. Fox, D. Patterson) believe Rails is best

Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
7. Productivity: Conciseness, Synthesis, Reuse, and Tools
Software Architecture

- Can you design software so that you can recombine independent modules to offer many different apps without a lot of programming?

Service Oriented Architecture

- SOA: SW architecture where all components are designed to be services.
- Apps composed of interoperable services.
  - Easy to tailor new version for subset of users.
  - Also easier to recover from mistake in design.
- Contrast to “SW silo” without internal APIs.

Bookstore: Silo

- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API (“Bookstore”)

Bookstore: SOA

- Subsystems independent, as if in separate datacenters
  - Review Service access User Service API
- Can recombine to make new service (“Favorite Books”)

(Figure 1.2, Engineering Software as a Service by Armando Fox and David Patterson, 2nd Beta edition, 2013.)
Characteristics of SOA

• SOA does not match the traditional layered model of software, in which each higher layer is built directly from the primitives of the immediately lower layer as in siloed software.
• SOA implies **vertical slices** through many layers, and these slices are connected together to form a service.
• While SOA usually means a bit more work compared to building a siloed service, the payback is tremendous **reusability**.
• Another upside of SOA is that the **explicit APIs** make testing easier.

Two widely accepted downsides to SOA

• First, *each invocation of a service involves the higher cost* of wading through the deeper software stack of a network interface, so there is a **performance hit to SOA**.

• Second, while a siloed system is very likely to be completely down on a failure, software engineers using SOA must deal with the sticky case of **partial failures**, so SOA makes dependability planning a bit more challenging.

CEO: Amazon shall use SOA!

Amazon started in 1995 with siloed software for its online retailing site. According to former Amazonian Steve Yegge, in 2002 the CEO and founder of Amazon mandated a change to what we would today call SOA. He broadcast an email to all employees along these lines:

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 interprocess 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.”

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!”
Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
7. Productivity: Conciseness, Synthesis, Reuse, and Tools

The Next Revolution in IT: Cloud Computing

- **Classical Computing**
  - Buy & Own
    - Hardware, System Software, Applications often to meet peak needs.
  - Install, Configure, Test, Verify, Evaluate
  - Manage
  - ...
  - Finally, use it
  - $$...$$ $(High CapEx)

- **Cloud Computing**
  - Subscribe
  - Use
  - $ - pay for what you use, based on QoS

(Courtesy of Raj Buyya, 2012)

The Cloud Stack: Separation of Responsibilities

<table>
<thead>
<tr>
<th>Standard</th>
<th>Infrastructure (as a Service)</th>
<th>Platform (as a Service)</th>
<th>Software (as a Service)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Customer Manages</td>
<td>Customer Manages</td>
<td>Service Provider Manages</td>
</tr>
<tr>
<td></td>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Middleware</td>
<td>Middleware</td>
<td>Middleware</td>
</tr>
<tr>
<td></td>
<td>O/S</td>
<td>O/S</td>
<td>O/S</td>
</tr>
<tr>
<td></td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
</tr>
<tr>
<td></td>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
</tr>
<tr>
<td></td>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
</tr>
<tr>
<td></td>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
</tr>
<tr>
<td></td>
<td><strong>Optimized Data Center</strong></td>
<td><strong>Cloud Data Center</strong></td>
<td></td>
</tr>
</tbody>
</table>

What is ideal HW for SaaS?

- Amazon, Google, Microsoft … developed hardware systems to run SaaS
- What did they use: Mainframes? Supercomputers?
- How can independent SW developers build SaaS apps and compete without similar HW investments to Amazon, Google, Microsoft?
**SaaS Infrastructure?**

SaaS’ s 3 demands on infrastructure

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

---

**Services on Clusters**

Clusters: Commodity computers connected by commodity Ethernet switches

1. More scalable than conventional servers
2. Much cheaper than conventional servers
   – 20X for equivalent vs. largest servers
3. Dependability via extensive redundancy
4. Few operators for 1000s servers
   – Careful selection of identical HW/SW
   – Virtual Machine Monitors simplify operation

---

**Warehouse Scale Computers**

- Clusters grew from 1000 servers to 100,000 based on customer demand for SaaS apps
- Economies of scale pushed down cost of largest datacenter by factors 3X to 8X
  - Purchase, house, operate 100K v. 1K computers
- Traditional datacenters utilized 10% - 20%
- Earn $ offering pay-as-you-go use at less than customer’s costs for as many computers as customer needs

---

**Utility Computing / Public Cloud Computing**

- Offers computing, storage, communication at pennies 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

Engineering SaaS for Cloud Computing is radically different from engineering shrink-wrap software for PCs and servers
## 2013 AWS Instances & Prices

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Per Hour</th>
<th>$ to small</th>
<th>Virtual Cores</th>
<th>Compute Units</th>
<th>Memory (GiB)</th>
<th>Storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>$0.06</td>
<td>1.0</td>
<td>1</td>
<td>1.0</td>
<td>1.7</td>
<td>1 x 160</td>
</tr>
<tr>
<td>m1.medium</td>
<td>$0.12</td>
<td>2.0</td>
<td>1</td>
<td>2.0</td>
<td>3.8</td>
<td>1 x 410</td>
</tr>
<tr>
<td>m1.large</td>
<td>$0.24</td>
<td>4.0</td>
<td>2</td>
<td>4.0</td>
<td>7.5</td>
<td>2 x 420</td>
</tr>
<tr>
<td>m3.xlarge</td>
<td>$0.48</td>
<td>8.0</td>
<td>4</td>
<td>8.0</td>
<td>15.0</td>
<td>4 x 420</td>
</tr>
<tr>
<td>m3.2xlarge</td>
<td>$1.00</td>
<td>16.7</td>
<td>8</td>
<td>26.0</td>
<td>30.0</td>
<td>EBS</td>
</tr>
<tr>
<td>c1.medium</td>
<td>$0.15</td>
<td>2.5</td>
<td>2</td>
<td>5.0</td>
<td>1.7</td>
<td>1 x 350</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>$0.58</td>
<td>9.7</td>
<td>8</td>
<td>20.0</td>
<td>7.0</td>
<td>4 x 420</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>$2.40</td>
<td>40.0</td>
<td>32</td>
<td>88.0</td>
<td>60.5</td>
<td>4 x 840</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>$0.41</td>
<td>6.8</td>
<td>2</td>
<td>6.5</td>
<td>17.1</td>
<td>1 x 420</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>$0.82</td>
<td>13.7</td>
<td>4</td>
<td>13.0</td>
<td>34.2</td>
<td>1 x 850</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>$1.64</td>
<td>27.3</td>
<td>8</td>
<td>26.0</td>
<td>68.4</td>
<td>2 x 840</td>
</tr>
<tr>
<td>cr1.8xlarge</td>
<td>$3.50</td>
<td>58.3</td>
<td>32</td>
<td>88.0</td>
<td>244.0</td>
<td>2 x 120 SSD</td>
</tr>
<tr>
<td>hi1.4xlarge</td>
<td>$3.10</td>
<td>51.7</td>
<td>16</td>
<td>35.0</td>
<td>60.5</td>
<td>2 x 1024 SSD</td>
</tr>
<tr>
<td>hs1.8xlarge</td>
<td>$4.60</td>
<td>76.7</td>
<td>16</td>
<td>35.0</td>
<td>117.0</td>
<td>24 x 2048</td>
</tr>
<tr>
<td>t1.micro</td>
<td>$0.02</td>
<td>0.3</td>
<td>1</td>
<td>varies</td>
<td>0.6</td>
<td>EBS</td>
</tr>
<tr>
<td>cg1.4xlarge</td>
<td>$2.10</td>
<td>35.0</td>
<td>16</td>
<td>33.5</td>
<td>22.5</td>
<td>2 x 840</td>
</tr>
</tbody>
</table>

## Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing

### 5. Beautiful vs. Legacy Code

• **Legacy code**: old SW that continues to meet customers' needs, but difficult to evolve due to design inelegance or antiquated technology
  - ____% SW maintenance costs adding new functionality to legacy SW
  - ____% for fixing bugs

• **Contrasts with beautiful code**: meets customers' needs and easy to evolve

## Legacy SW vs. Beautiful SW

### Programming Aesthetics

• Do I care what others think of my code?
  - If it works, does it matter what code looks like?
### 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

- Contrasts with **beautiful code**: meets customers' needs and easy to evolve

### Legacy Code: Vital but Ignored

- Missing from traditional SWE courses and textbooks
- #1 request from industry experts we asked: What should be in new SWE course?
  - Save work by reusing existing code (e.g., open source)
- Will have legacy lectures and programming assignments later in course
  - Helps you learn how to make beautiful code

---

### Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
6. **Software Quality Assurance: Testing**

### Software Quality

**Software Quality**

- What is software quality, and how do we assure it? (QA)
- **V&V**: What is the difference (if any) between Verification and Validation?
Meaning of “Verification” and “Validation”

**Verification:**
are we building *the product right*?

**Validation:**
are we building *the right product*?

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. Be easy for developer to debug and enhance

- **Software QA:** ensure quality and improve processes in SW organization

Assurance

- **Verification:** Did you build the thing *right*?
  - Did you meet the specification?
- **Validation:** Did you build the right *thing*?
  - Is this what the customer wants?
  - Is the specification correct?
- Hardware focus generally __________
- Software focus generally __________
- 2 options: Testing and Formal Methods
Testing

- Exhaustive testing infeasible
- Divide and conquer: perform different tests at different phases of SW development
  - Upper level doesn’t redo tests of lower level

<table>
<thead>
<tr>
<th>System or acceptance test: integrated program meets its specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration test: interfaces between units have consistent assumptions, communicate correctly</td>
</tr>
<tr>
<td>Module or functional test: across individual units</td>
</tr>
<tr>
<td>Unit test: single method does what was expected</td>
</tr>
</tbody>
</table>

More Testing

- Black box vs. White Box testing
  - Testing based on specs vs. on implementation
- Test Coverage: % of code paths tested
- Regression testing: automatically rerun old tests so changes don’t break what used to work
- Continuous Integration (CI) testing: continuous regression testing on each code check-in vs. later testing phase

Introduction to SaaS and Agile Development

1. What is Software Engineering?
2. Software as a Service
3. Service Oriented Architecture (SOA)
4. Cloud Computing
5. Beautiful vs. Legacy Code
7. Productivity: Conciseness, Synthesis, Reuse, and Tools

Moore’s Law is Alive and Well

2X transistors/Chip Every 1.5 years Called Moore’s Law

Microprocessors have become smaller, denser, and more powerful.

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra
Productivity: Conciseness, Synthesis, Reuse, and Tools

Productivity

- 50 years of Moore’s Law => 2X /1.5 years
  - HW designs get bigger
  - Faster processors and bigger memories
  - SW designs get bigger
  - Had to improve SW productivity

- 4 techniques
  1. Clarity via Conciseness
  2. Synthesis
  3. Reuse
  4. Automation and Tools

Clarity via Conciseness

1. Syntax: shorter and easier to read
   assert_greater_than_or_equal_to(a,7)
   vs. ________________

2. Raise the level of abstraction:
   - HLL programming languages vs. assembly lang
   - Automatic memory management (Java vs. C)
   - Scripting languages: reflection, metaprogramming
   - ....

Synthesis

- The implementation is generated rather than created manually.

- **Logic synthesis for hardware engineers** meant that they could describe hardware as Boolean functions and receive *highly optimized transistors* that implemented those functions.

- **Software synthesis**
  - BitBlt: generate code to fit situation & remove conditional test

- Research Stage: Programming by example
Reuse

- Reuse old code vs. write new code

Techniques in historical order:
1. Procedures and functions
2. Standardized libraries (reuse single task)
3. Object oriented programming: reuse and manage collections of tasks
4. Design patterns: reuse a general strategy even if implementation varies

Automation and Tools

- Replace tedious manual tasks with automation to save time, improve accuracy
  - New tool can make lives better (e.g., make)
- Concerns with new tools: Dependability, UI quality, picking which one from several
- Good software developer must repeatedly learn how to use new tools: lifetime learning
  - Lots of chances in this course: Cucumber, RSpec, Pivotal Tracker, ...

DRY (Don't Repeat Yourself)

- “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.”
  - Andy Hunt and Dave Thomas, 1999
- Don't Repeat Yourself (DRY)
  - Don’t want to find many places have to apply same repair
- Refactor code so that can have a single place to do things

And in Conclusion: § § 1.1-1.7

- Class: SW eng. Principles via Cloud app by team for customer + enhancing legacy app
- SaaS less hassle for developers and users
- Service Oriented Architecture makes it easy to reuse current code to create new apps
- Scale led to savings/CPU => reduced cost of Cloud Computing => Utility Computing
- Testing to assure software quality, which means good for customer and developer
- Developer Productivity: Conciseness, Synthesis, Reuse, and Tools
Acknowledgement

- Slides have been based in-part upon original slides/notes of a number of books and people including:
