

# Software Engineering: A Personal Reflection

Yu Lei (雷羽)

University of Texas, Arlington

June 2014

What is THE best job in the US in 2014?


US News & World Report: Software Developer

# Software Developer

Best Technology Jobs

## Software Developer

Job Profile
Overview
Salary
Reviews & Advice
Job Listings
Civil Engineer
Computer Programmer
Computer Support Specialist
Computer Systems Administrator
Computer Systems Analyst
Database Administrator
Information Security Analyst
IT Manager
Mechanical Engineer
Web Developer



**Overall Score** ★★★★★  
(8.4 out of 10)

**Number of Jobs**  
139,900

**Median Salary**  
\$90,060

**Unemployment Rate**  
2.8 percent

**Show Jobs Near:**  [Find Jobs](#)

Software developers have their fingerprints all over our lives. That alarm clock app that woke you from a dead sleep this morning was designed by at least one software developer. The host of applications you fire up on your computer when you arrive to the office? Yes, software developers had a big hand in shaping those, too. From the mobile app you use to check your bank account balance to the games you play on your tablet to unwind, software developers are along for the ride thanks to their amazing code creations.

This Job is Ranked in

Best Technology Jobs	#1
The 100 Best Jobs	#1

# Consistent Ranking

---

- US News & World Reports
  - 2014: 1<sup>st</sup> Best Job
  - 2013: 7<sup>th</sup> Best Job (1<sup>st</sup> in Technology)
  - 2012: 1<sup>st</sup> Best Job
- CareerCast.com
  - 2014: 7<sup>th</sup> Best Job
  - 2013: 3<sup>rd</sup> Best Job
  - 2012: 1<sup>st</sup> Best Job
- Based on hiring prospects, salary, job satisfaction, and unemployment rates

- What is SwE and Why?
- What is SwE Research?
- Major Challenges in SwE
- How to Conduct SwE Research?

# Software Engineering

---

- Software has become **pervasive** in modern society
  - A fundamental driver to technology and society advancement
- How to build **better** software **faster**, especially at scale?
  - Requirements, analysis, design, coding, testing, maintenance, configuration, documentation, deployment, and etc.

# Software Quality

- 
- **THE** priority concern in software engineering
    - No quality, no engineering!
    - Software malfunctions cost billions of dollars every year, and have severe consequences in a safe-critical environment
  - An important factor that distinguishes a software product from its competition
    - The feature set tends to converge between similar products

# What is Quality?

---

- As perceived by the user
  - Functionally correct, secure, robust, reliable, usable, fast response, high throughput, ...
- As perceived by the developer
  - Code that is well-structured, easy to understand, easy to change, easy to extend, easy to test, ...
- Software = program + documentation
  - Concise, well-written, consistent, up-to-date



# Engineering vs Science

---

- Engineering solutions are not necessarily optimal solutions
- More emphasis on **quality**, **cost**, and **scale** than **optimality**
- The **KISS** Principle: Avoid unnecessary complexity, sometimes even at the cost of optimal performance.

# Engineering vs Art

---

- Unlike art, quality of an engineering product should be not depend on the person(s) who build the product.
- Some people do not like the term of “software engineering”. Why?

- What is SwE and Why?
- What is SwE Research?
- Major Challenges in SwE
- How to Conduct SwE Research?

# What is it?

- 
- Develop **concepts, principles, models, methods, techniques, and tools** that make software engineers more productive
  - In other words, **SwE Research** helps software engineers to do their job better ...
  - ... and in turn software engineers help other people to do their jobs better.

# On-the-large vs On-the-small

---

- **On-the-large**: Manage the overall production process
  - Software process and methodology
  - Project management, i.e., planning & scheduling
- **On-the-small**: Focus on the “**how-to**” of individual tasks
  - Requirements engineering, design patterns, design verification, program analysis, debugging, testing, formal methods, and others

- Similar in terms of both dealing with code (instead of data)
- However, **SwE** reasons about **semantic** properties ...
- ... whereas **Compiler/languages** analyzes code in the **syntactic** domain
- They are moving closer to each other
  - e.g., some compiler plugins are performing semantic analysis, while many SwE techniques need to analyze syntactic structures

# SwE vs System

---

- Similar in terms of both dealing with how to build software systems
- However, **SwE** is about the general software production process, and is typically **application-independent**...
- ... whereas **System** is typically about how to build systems of a specific type/domain, and is mainly concerned at the **design** level

# SwE vs Algorithm

- 
- Similar in terms that both need to do **problem solving**
  - However, **SwE** is about the software production process/activities, whereas **Algorithm** deals with individual “application” problems
  - Also, **SwE** is ultimately about software **implementations**, whereas **Algorithm** is more about “**abstract**” solutions.



# SwE vs Security

---

- **Security** is an important aspect of quality.
- **Software security** is mainly about security of “software” **implementation**...
  - Typically assume there exists a secure design/ protocol/mechanism
- ... whereas (traditional) **Security** is mainly about security in the **design** level

- To a large extent, **SwE** is about analysis of **code**, whereas many other CS areas are about analysis of **data**
  - e.g., database, network, data mining, machine learning, and others
- **Structure of code** is different from **structure of data**...
- Also, code has **dynamic** behavior, whereas data is largely **static**.

# Defining Characteristics

---

- Deals with **semantics** instead of **syntax**
  - Semantic reasoning is often **undecidable!!**
- Analyzes **code** instead of **data**
  - How to analyze the runtime behavior that may be displayed by (static) code?
- Consider the **general** software production process/activities, instead of solving individual problems.
- The ultimate goal is to ensure **quality of implementation.**

- What is SwE and Why?
- What is SwE Research?
- Major Challenges in SwE
- How to Conduct SwE Research?

- 
- How to manage activities in the software production lifecycle?
    - When to perform what activities?
  - Technically, how to balance between **structure/discipline** and **flexibility**?
    - Structure/discipline is important for quality control and for repeatability, but software is complex and dynamic
  - The trend seems to be **incremental**, **iterative**, and **feedback-based** control.

- How to build the right system (vs build the system right)?
- One of the most challenging problems
  - Often customers do not know what they want.
- How to help customers spell out all the requirements?
- How to make sure requirements are valid and consistent with each other?
- How to describe requirements in a way that is amenable to automated analysis?

- 
- **Abstraction** that contains important decisions. But what decisions are or are not important?
  - How to measure quality of a design? Given two designs, which one is better?
    - A good design should not only work for the present, but also facilitate the future.
  - How to create a quality design in a systematic, repeatable manner?
    - Design patterns help to capture best practices.
  - How to verify correctness of a design?

- How to ensure that software is implemented without errors?
- Two basic approaches
  - **Static analysis**: analyze code without execution
  - **Dynamic analysis** or **testing**: execute code and observe the runtime behavior
- Static analysis can be fast, but suffers false positives/negatives
  - Try to automate code inspection/review, but essentially an undecidable problem. How to reduce false positives/negatives?



# Defect Detection (2)

---

- Testing is probably the most widely used approach in practice
  - The key is to be **systematic**, typically through the notion of **coverage**
- Three technical problems:
  - How to select test inputs? How to automatically execute tests? How to evaluate test runs?
- Combine static analysis and testing
  - For example, collect information about program structure to help test generation

- 
- A mathematic treatment of software engineering
    - Precise, rigorous, and provides deep insights
  - How to formally describe and reason about a system and its properties?
    - Deductive reasoning vs model checking
  - How to check correctness of implementations directly, i.e., not only designs?
  - How to make formal methods accessible to average engineers?

# Empirical SwE

- 
- Various SwE methods and techniques have been developed.
    - Are they really effective? How do they compare to each other?
  - Can be time-consuming, and must follow a rigorous procedure
  - How to select a set of subject programs and faults that are representative of true practice?
    - Benchmark programs are preferred but may not be always available

- What is SwE and Why?
- What is SwE Research?
- Some SwE Research Challenges
- How to Conduct SwE Research?

# What is Good Research?

---

- Novel idea and useful
  - The Aha Effect: It is so simple, and why didn't I think of this?
  - Must address a problem of significance
- Solid support
  - Ideas alone do not work
  - Rigorous reasoning and experimental results
- Excellent presentation
  - Make it as clear as possible, instead of “you figure it out”

# Major Tasks

---

- Literature search: Obtain a big picture of “what is out there”
- Problem formulation: Clearly define the input and output
- Propose your idea: What is the trick?
- Prototype & experiments: Provide evidence that it will actually work
- Publication: Get your work known to the community.

- 
- Critical to justify novelty
    - Many papers are rejected due to inadequate discussion on related work
  - Where to search
    - **Online databases:** ACM Digital Library, IEEE Xplore, ScienceDirect, and SpringerLink
    - **Google Scholar, Citeseer, Google**
  - Browse recent proceedings (5 to 10 years) of major conferences
  - Book-keeping: create a web page which collects all the relevant resources

# How to read papers

---

- Major questions to keep in mind
  - What is the problem and why? What is the technical challenge?
  - What is the main idea of the proposed solution? What are the major results?
  - How does the work compare to others?
  - What is your opinion on the strength/weakness of the work?
- Try to finish reading a paper in its entirety before reading its references
- No shortcut!



# Research vs Development

---

- **Development:** We know the problem, and often the solution, but take time to implement/optimize the solution
- **Research:** We do not even know the problem, not to mention the solution
- How to make progress?
  - Clearly define the input and output, systematic exploration of possible ideas, planning & schedule based on deliverables
- Don't give up easily; if you do, try to have a good justification

- Experimental design
  - Research questions, subject programs, metrics
  - What are the major factors that could affect the effectiveness of your approach?
  - Always keep COST in mind!
- Baseline and competing work
  - What is the intuitive approach? How does your work compare to other similar work?
- Make sense out of numbers
  - Not only present the data, but also explain why such data has been presented

# Experiments (2)

---

- Threats to validity
  - Thinking about this helps to improve your design
  - Critical from the 3<sup>rd</sup> perspective
  - How have you tried to reduce the threats
- Make tables self-contained
- Any inconsistencies or surprise results need to be explained

# Be Patient!

- 
- Good research takes time
    - Years of accumulation
  - Software engineering typically has a longer publication cycle
    - Code can be more difficult to analyze than data, due to its internal structure and dynamic behavior
    - Building prototype tools and conducting empirical studies can be time consuming
  - But it can be very rewarding
    - Make impact in real life

# Quotes from Dr. Parnas

---

***Which computer-related area is most in need of investment by government, business or education?***

I think it would have to be software engineering education. First we have to improve the quality of that education. I find that most students receive a poorly structured and random introduction to software issues. They learn a lot of folklore that is too vague to apply and a lot of theory that seems (and often is) irrelevant. Second, we must make sure that people do get a professional education. In a world where you need a license to be a barber, anybody can get a job writing software without any credentials. In the last few years I have devoted much of my time to developing an educational program for engineers who are specialists in software, a program that can be accredited by the professional engineering societies.

***What advice do you have for computer science/software engineering students?***

Most students who are studying computer science really want to study software engineering but they don't have that choice. There are very few programs that are designed as engineering programs but specialize in software.

I would advise students to pay more attention to the fundamental ideas rather than the latest technology. The technology will be out-of-date before they graduate. Fundamental ideas never get out of date. However, what worries

Extracted from his ACM Fellow Profile

<http://www.sigsoft.org/SEN/parnas.html>

Education is what remains after one has forgotten everything he learned in school.

- Albert Einstein