

# SENG371

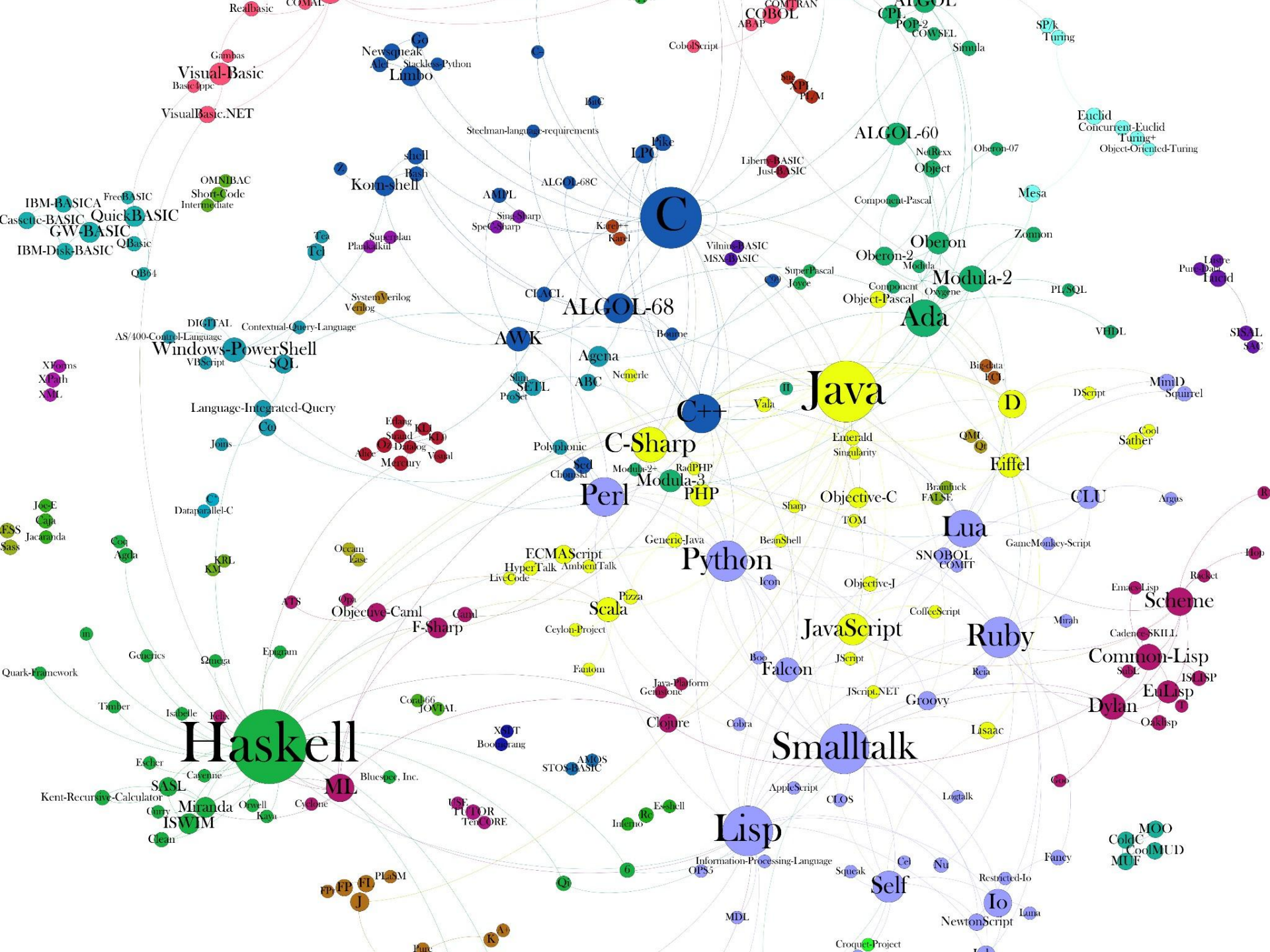
1968 NATO Software  
Engineering Conference

# 1.0 Software Evolution

- Software is going to evolve rapidly
- Every person will interact with a computer in some way in their everyday activities
- Legacy systems are going to become very complex and very unmaintainable
- So many programming languages?

In 1968, less than 30

Now, hundreds.



## 2.0 Education

- Object-Oriented over Procedural
- Waterfall engineering technique is not suitable with Software Development.
  - Use Agile programming
  - Test Driven Development
- Education early.
  - SW ubiquitous, but many learn informally.
- Communication is key.
  - global collaboration (internet)

## 3.0 Scalability

"Don't build for today, build for forever"  
- Romil Khanna

- Software systems can grow to accommodate huge numbers of people.
- Dealing with big data.

# Advice to 1968 Software Engineers

- Software Evolution
- Program Understanding
- Security

- Daniel, Brad, Dave, George

# Software Evolution

- Software will have to go through continual change in order to adapt with its environment and previously unknown requirements, no system is ever finished (David)
- Failure to evolve will lead to system depreciation over time, and eventually to obsolescence (George)
- Machines must be able to communicate with each other in order to dynamically adjust at runtime (failure of one machine must not cause whole system to crash) (Daniel)
- Organizations will have huge investments in software systems. To maintain the value of these assets, they must be changed and updated over time (Brad)



# Program Understanding

- Multiple approaches:
  - top-down - use own experience and try to confirm expectations
  - Bottom-up - iteratively abstract high-level understanding by reading code,
  - Opportunistic - mix of both strategies (Brad)
- Much mechanical and electrical engineering, diagrams must be used to show software systems. Code in background, much like equations (Daniel)
- There will be programs available for you to help analyse, document, and improve previously confusing code. (George)
- There will be a heavy emphasis on documentation to aid with program understanding, ie: **DON'T FORGET TO COMMENT YOUR CODE.** (David)




# Security

- Software engineers must spend 10% or more of development time solely focused on security to always stay one step ahead of an attacks. (Daniel)
- Dependence on information technology makes software assurance a key element of business continuity, national security, and homeland security (Brad)
- People will continually find and exploit vulnerabilities in software systems as a lot of money can be made in doing so. (David)
- One of the commonly used methods for improving software security is ethical hacking, where software engineers will purposely try and find vulnerabilities so they can be fixed before they are found by malicious hackers. (George)



# Advice to Pioneers

Nicholas Guillemot, Vishwendra Gahlot,  
Richard McKenzie, Marcelo Gomes



# Software Evolution

- Code Written now likely still used in 20 years
- Create a documentation standard and promote across industry
- Document for future people working on project, not just using it
- Keep Source + Documentation + Update

# Software Engineering Education

- Practical Experience
- Case Studies of past failures & successes
- Encourage Industry Professionals teaching / guest lecturing more often

# Open Source Software

- Make Software public & allow collaboration
  - Working together for betterment of industry
  - Lowers entry barrier to industry
- Use open source license, along with paid support, to still make money but have more accessible product

# **SENG 371**

## **Dear History**

**Justin, Adam, Geoff, Scott**

# Software Evolution

- Software maintenance will grow to consume 40-60% of the software development cycle
  - Strive to evolve software rather than create legacy systems with an finite life span.
- Software development will move towards a decentralized system
  - The days of mainframes are over. Evolving systems must be componentized.



# Software Engineering Education

- Maintainability is extremely important
  - No product can be considered completely finished
- Antipatterns can be recognized
  - Look for common solutions to problems that don't actually work
- Promote continued learning
  - Software changes quickly, current skills will be less applicable in 10 years.
- Axe the waterfall model
  - No project can perfectly follow the waterfall model
  - You will almost never define all requirements

# Software Engineering Education (Continued)

- Reverse Engineering should be taught in schools
  - Software engineers will often have to work with undocumented legacy systems.
  - Reverse engineering is a key component of program understanding.
  - Reverse engineering concepts should be taught to encourage the development of widely adopted and automated RE tools.

# Agile Development

- **Static development methodologies don't work**
  - Requirements cannot all be known before development begins
  - Often requirements will change during development
  - Clients like to see prototype software over documents
  - Initial system design may not work.
- **Instead develop in cycles**
  - Focus on creating working software over any documentation.
  - Work with the client instead of creating a contract.
  - Be open to change instead of following a plan.

# Software Evolution

- Software efforts and costs will eventually be spent mostly in maintenance and evolution phase
  - Systems should be designed to be adaptable, maintainable and scalable.
  - Systems will eventually require constant development and adjustment
- Start work early on making software evolution more cost and time effective

# Software Architecture

- Design for expanding scale
  - Scale of systems will grow exponentially
  - Design systems with that in mind
  - Importance of strong/stable architecture grows with the systems size
- Try to make designs functionally independent and able to work together
  - Must find a way to encapsulate similar functionality or else maintainability of large systems becomes impossible

# Interfaces

- Develop standards early
  - Investigate clear standards for both human computer interaction and computer-computer interaction
- Connectors are important
  - Spend lots of time making computers work together
  - Systems should abstract away functionality not needed by the 'user'

# Software Evolution

- Aim to create systems that are easy to maintain and evolve
  - Up to 90% of the software development is now being used for maintenance now-a-days (in the future!).
- Make use of existing libraries when possible
  - Less in-house source code to manage/maintain and saves you from reinventing the already existing wheel.
- Try to follow the DRY and SOLID Principles in OO Design/Implementation
  - This will greatly increase the maintainability of your application/system



# Software Architecture

- A coherent architecture is key to any software system (design, design design)
  - Without it you may incur additional maintenance or refactoring costs
- Make use of existing design patterns and watch for the development of anti-patterns
  - Design patterns provide a framework to launch from and are often more maintainable
- Be consistent in your design at each level/layer of an application and prefer composition to inheritance.

# Continuous Learning

- It is important to remain up-to-date on emerging technologies and concepts
  - Research is always being done to solve existing problems in software and development systems/methods of conquering problems
- You are only worth as much as you know
  - The more you know the better! So learn ALL OF IT!
- I know you guys are at the beginning of the 'internet' (arpanet), but when it goes public, adopt it and share your knowledge with the world.
  - It'll be like 1982, if history repeats itself

### Slide 3

---

1 arpanet the precursor of the internet was built in 1968

Curtis St. Pierre,

1 Why you no fix it then

Paul Hunter,

2 I no know what you want

Curtis St. Pierre,

1 idunno eitherr  
maybe no cursing? haha

Mikko Sanchez,

# **What Advice Would We Give?**

**To the 1968 NATO conference in  
Garmisch Germany**

# Software Evolution

- Standards
- Agile development methodologies
- Reverse Engineering

# Software Architecture

- Modularity of software components
- Levels of abstraction
- Separation of concerns

# Copyright

- How do you copyright software?
- Can you copyright an API?
- How do you patent a software user interface?
- How can software companies protect their assets?



# Software Education

---

- K - 12 Education needs to be implemented
- Undergraduate Education effectiveness needs improvement
- Undergraduate Education needs to focus on industry needs

Jeremy, Wes, Anita

---

# Software Architecture

---

- Allows early analysis of a system
- Crucial design decisions are made early on
- Gives stakeholders an idea of what they will see
- Reduces overall costs

Jeremy, Wes, Anita

---

# Software Tools

---

- Invest in a good interface. Software tools will have more users in the future and the key to adoption is an interface that is easy to use.
- Focus on simplicity, flexibility and extensibility. A successful tool does its job well and can easily be integrated into commonly used platforms and environments.
- Extensive evaluation

Jeremy, Wes, Anita

---

# **SENG 371**

## **Dear History**

**Justin, Adam, Geoff, Scott**

# Software Evolution

- Software maintenance will grow to consume 40-60% of the software development cycle
  - Strive to evolve software rather than create legacy systems with an finite life span.
- Software development will move towards a decentralized system
  - The days of mainframes are over. Evolving systems must be componentized.

# Software Engineering Education

- Maintainability is extremely important
  - No product can be considered completely finished
- Antipatterns can be recognized
  - Look for common solutions to problems that don't actually work
- Promote continued learning
  - Software changes quickly, current skills will be less applicable in 10 years.
- Axe the waterfall model
  - No project can perfectly follow the waterfall model
  - You will almost never define all requirements

# Software Engineering Education (Continued)

- Reverse Engineering should be taught in schools
  - Software engineers will often have to work with undocumented legacy systems.
  - Reverse engineering is a key component of program understanding.
  - Reverse engineering concepts should be taught to encourage the development of widely adopted and automated RE tools.



# Agile Development

- Static development methodologies don't work
  - Requirements cannot all be known before development begins
  - Often requirements will change during development
  - Clients like to see prototype software over documents
  - Initial system design may not work.
- Instead develop in cycles
  - Focus on creating working software over any documentation.