CRA Computing Research Association

CRA-I Computing Research Association Industry

Key Findings Snapshot

What Computing Practitioners Are Saying About Undergraduate Computer Science Education

CRA Practitioner-to-Professor (P2P) Survey

December 2024

About the CRA P2P Survey

The CRA Practitioner-to-Professor (P2P) survey, launched by the Computing Research Association (CRA) in Spring 2024, is designed to gather actionable feedback from industry professionals to inform and improve computing curricula in academia. The survey aims to ensure university programs remain aligned with evolving industry needs, fostering a long-term partnership between academia and industry.

This initiative establishes a valuable feedback loop, helping universities stay relevant and globally competitive while equipping graduates to meet industry demands. The project is endorsed by ABET, the Association for Computing Machinery (ACM), CSAB, and the IEEE Computer Society (IEEE-CS).

Coming Soon

A full report with detailed survey findings and analysis will be available in early 2025.

Respondent Profile

The survey gathered 1,048 qualified responses, with participants providing valuable insights into the competencies and course experiences that best prepare students for the workforce. The responses were screened to ensure data quality through bot detection measures. The diverse and experienced respondent pool ensures the survey results reflect broad and actionable industry perspectives.

Who Responded?

Educational Background:

- 54% hold degrees in computer science.
- Bachelor's (40%) and Master's (33%) degrees were the most common.

Professional Experience:

- 61% have 21+ years of experience.
- 76% work in software development.
- 64% are in front-line technical roles or manage such teams.

Company Demographics:

- 50% work at mid-to-large companies (1,000+ employees).
- 61% are employed by companies primarily focused on computing technology.

Key Findings

How many CS courses, and which ones?

For each CS course topic, respondents indicated the ideal number of courses compared to the number actually completed during their undergraduate curriculum. The results showed broad support for "more CS courses," with an average recommendation of approximately four additional courses.

- The average recommended number of CS courses was 18.3 (equivalent to 3-4 semesters of coursework).
- The topics identified as most important for additional courses were Algorithms, Computer Architecture, and CS Theory.

For non-CS foundational courses such as science, math, and writing, respondents suggested an average increase of 1.7 courses across all areas.

- The average recommended number of non-CS foundational courses was 16.5 (equivalent to ~3 semesters of coursework).
- The areas identified as most important were written communication, probability and statistics, single-variable calculus, systems thinking, and co-op or internship experiences.

How should students approach learning programming languages?

The top five assertions regarding programming languages, ranked by importance, are:

1. Problem-solving is more important than programming skill.
2. It is important to know one language deeply.
3. Students should be exposed to multiple languages.
4. Students should have the ability to learn new languages independently.
5. Students should be familiar with an object-oriented language.

How important are soft skills in computing education?

Four key assertions emerged regarding the importance and development of soft skills:

- Soft skills can be taught, but universities do a poor job of teaching them.
- Soft skills grow with maturity and are often developed on the job.
- Undergraduate programs should include more oral-communication courses.
- A broad liberal-arts education is more effective at developing soft skills than a technically focused or engineering education.

The recommendation for more liberal arts courses stands in contrast to the earlier call for "more CS courses" but aligns well with the extra semester allocated for non-CS foundational courses (4 semesters for CS, 3 for other foundations).

Among communication skills, the highest-rated priorities were:

1. Speaking confidently in small technical groups.
2. Writing technical documentation.
3. Speaking confidently to leaders and decision-makers.
4. Speaking confidently to non-technical clients in one-on-one settings.

Key Findings

What is the connection between math and computer science?

Respondents reported a strong connection between mathematics and computer science: 65% enjoyed or pursued additional mathematics coursework, while only 6% stated that mathematical ability plays no role in software development.

Key reasons for mathematics' importance:

- Mathematics underpins growing areas such as AI, machine learning, data science, and quantum computing.
- It trains analytical thinking, which is valuable even in software-focused roles.
- A strong mathematical foundation supports adaptability as careers and the field evolve.
- Mathematical training reflects key attributes such as diligence, a willingness to tackle challenging material, and attention to detail.

Mathematics remains a critical foundational area, even for software engineering practitioners. Fields like probability, statistics, and linear algebra have increased in importance due to their relevance to modern computing disciplines.

Top mathematics areas recommended:

1. Statistics (56%)
2. Linear Algebra (40%)
3. Discrete Mathematics (37%)
4. Single-Variable Calculus (32%)
5. Logic (32%)

What should students focus on when learning about algorithms?

Respondents emphasized the importance of cultivating algorithmic thinking through theoretical reasoning and problem-solving approaches. Implementing algorithms was viewed as essential for gaining a deeper understanding of their functionality. Additionally, the specific areas of focus were considered less important than using algorithms as a tool to develop strong foundational skills in algorithmic reasoning.

The following topics were ranked as the most important:

1. Basic data structures (70%)
2. Analyzing running time (37%)
3. Advanced data structures (31%)
4. Sorting and searching (27%)
5. Basic graph algorithms (26%)

Key Findings

How do practitioners think databases should be taught?

Respondents highlighted the need for a balance between theoretical foundations and practical applications in database education. Understanding concepts like normal forms, indexing, and relational algebra was seen as just as important as gaining hands-on experience with SQL. This dual focus ensures that students develop both a conceptual understanding and practical skills to apply in real-world scenarios.

The following topics were ranked as the most important:

1. Basic SQL (70%)
2. 1st, 2nd, and 3rd normal forms (33%)
3. Index structures (33%)
4. Advanced SQL (31%)
5. Relational algebra (31%)

What should a computer architecture course focus on?

Respondents emphasized the importance of focusing on foundational concepts in computer architecture. These core areas, such as digital logic and the memory hierarchy, provide the building blocks for understanding more advanced topics. By mastering these fundamentals, students are better equipped to grasp the complexities of modern computing systems.

The following topics were ranked as the most important:

1. Digital logic, including Boolean algebra, arithmetic logic, gates, and circuits.
2. Memory hierarchy, with a focus on cache and virtual memory.

Full Report Coming in 2025

Insights to inform computing education and industry collaboration will be available at cra.org

This project is being partially supported by the Division of Undergraduate Education at the U.S. National Science Foundation under Award #2110815 under a larger umbrella project called DEAP.