

# Using stem in your classroom <sup>[0]</sup>

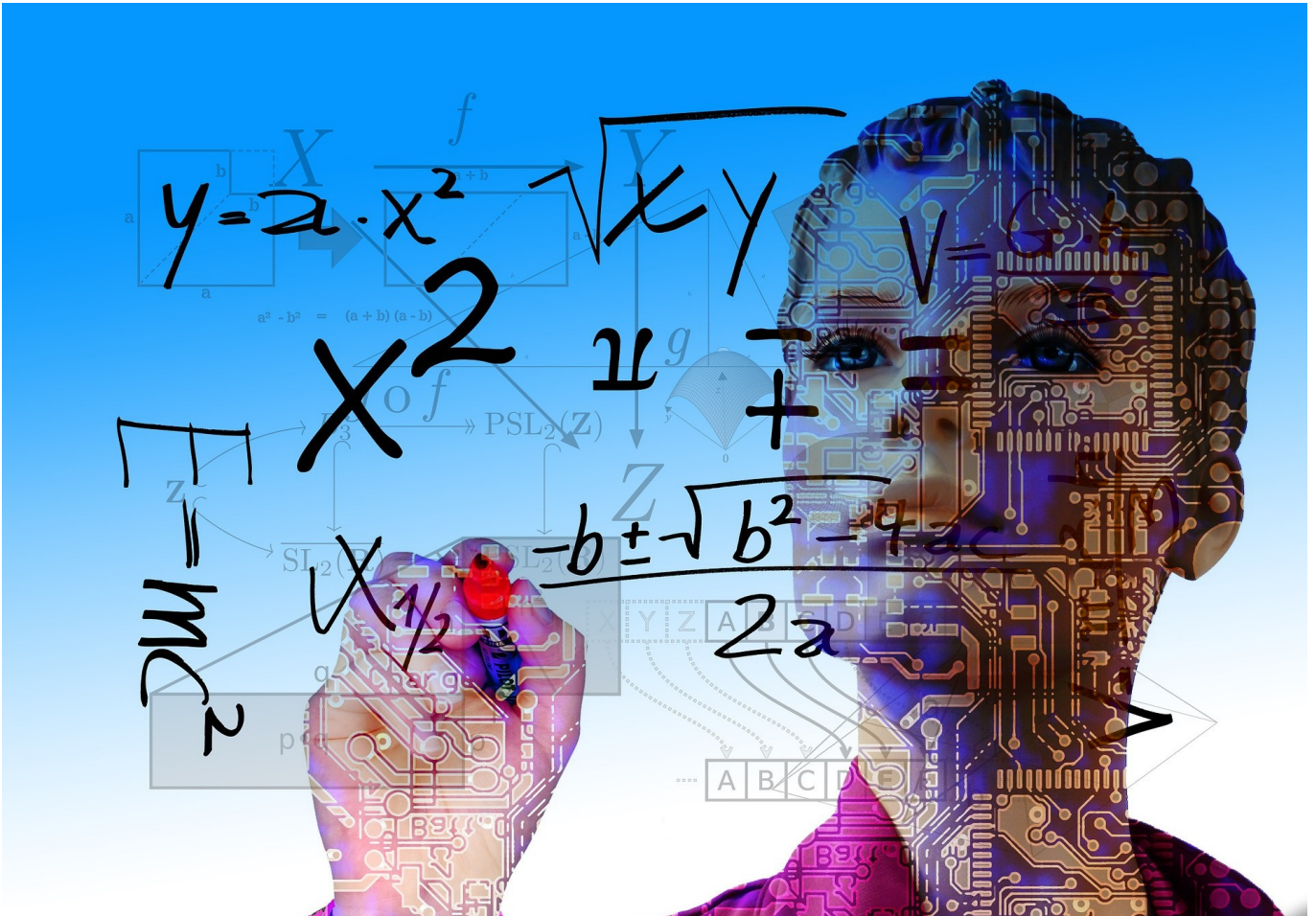


Image sources: [Gerd Altmann](#)<sup>[1]</sup>, [Gerd Altmann](#)<sup>[2]</sup> Pixabay. [cc0 / public domain](#)<sup>[3]</sup>

This article was first posted at 2015-11-11 at [robomatter.com](#)<sup>[4]</sup>

## Using your STEM Classroom to Teach What Matters

„computer science is no more about computers than astronomy is about telescopes“  
(Edsger Dijkstra)

Computers are an everyday part of life. We use them constantly in our personal lives and in the workplace. According to the U.S. Bureau of Labor statistics, over 50% of jobs today require some level of technology skills. And, that percentage is expected to grow to almost 80% in the next ten years.

There's no question that computer science skills are helping students succeed. But, computer science is about more than just learning to program. Students also need to learn how to think programmatically, to use programming as a problem-solving tool, and to understand the global impact of computer science and computing.

The most effective STEM programs include what are sometimes called the “Big Ideas” of computer science – foundational principles that are central to computing and help show students how computer science can change the world. Here's a quick overview of some of the big ideas we think are important,

and some tips on how you can incorporate them into your STEM Robotics or Computer Science classroom:

1. **Abstraction** – Abstraction is a key problem-solving technique that we use in our everyday lives and that can be applied across disciplines and problems. Abstraction helps students manage complexity by reducing the information and details of a problem, allowing them to focus on the main idea. But how do you teach students abstraction?

One way is to Implement a project that start with a complex problem but uses mini-challenges to break the problem into smaller pieces. Have students solve the mini-challenges, focusing on one aspect of the problem at a time, and then use those mini-challenge solutions to build a final solution to the larger, more complex problem.

2. **Algorithms** – Algorithms are used to develop and express computational problems and they're an important part of Computer Science. But, algorithmic thinking is a tool that students can apply across disciplines and problems. Algorithmic thinking means defining a series of ordered steps you can take to solve a problem. Therefore, it's important that students learn how to not only develop algorithms, but to also learn how to express algorithms in language, connect problems to algorithmic solutions, and evaluate algorithms effectively and analytically.

Here's one idea for introducing algorithms into your STEM Robotics or Computer Science classroom: Provide students with a list of numbers. Ask them to find the largest number and document the procedure they used. (This is also good pseudocode practice!) Next, tell students that they will be given a program that generates 10 random numbers between 0 and 30 and they will have to provide an algorithm to find the largest number from the list. Once students have generated the algorithm and seen it in action, discuss why the algorithm is valuable. While it may not be a big deal to find the largest number out of a group of 10, what if we increased the range of numbers from 0 to 10,000, and increased the amount of numbers from 10 to 1000? In a situation like that, an algorithm would be able to find the largest number much faster than a human.

„computational Thinking is the new literacy of the 21st century“  
(Jeannette Wing)

3. **Computational Thinking** – Computational thinking is a basic a problem-solving process that can be applied to any domain. This makes computational thinking an important skill for all students, and it's why our curriculum is structured to teach students how to use computational thinking to be precise with their language, base their decisions on data, use a systematic way of thinking to recognize patterns and trends, and break down larger problems into smaller chunks that can be more easily solved.

To learn more about implementing computational thinking in your classroom, read our blog post from last month, [What is Computational Thinking and Why Should You Care?](#)<sup>[5]</sup>

4. **Creativity** – People often think that science and creativity are two terms that don't belong together. However, that couldn't be further from the truth. Innovation and creativity are at the heart of STEM and Computer Science. Along with programming skills, students need to learn how to think creatively and need to get comfortable with the creative process. One great way to do this is by using structured problem-solving in your classroom. Structured problem-solving allows students to be creative, but within parameters. While students will still have opportunities to personalize their projects and justify their solutions, their creativity will still be structured. That way, teachers don't have to worry about students constantly losing focus.
5. **Data** – This “Big Idea” revolves around the fact that data and information facilitate the creation of knowledge. Over the past 50 years, the tasks that we perform on a routine basis have gotten more and more complex. According to an analysis done by Frank Levy and Richard J. Murane, the amount that employees are asked to solve unstructured problems and acquire and make sense of

new information has increased dramatically, by more than 40% .(1) Therefore, it's important to teach students how to analyze and interpret data.

You can do this by having students use coordinate data to code precise movements. Or, ask students to design a short, school-appropriate survey to collect data and answer specific questions. Then, have students write a program to input and analyze their data and calculate basic descriptive statistics such as mean, mode, range, and frequency. You can also ask students to plot their data on a chart or graph, and identify subgroups within the dataset to explain response patterns. Finally ask students to draw conclusions or make generalizations from their data and present their results to the class.

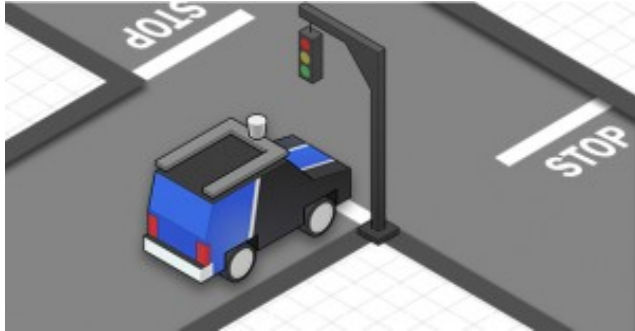


image rights: [robomatter.com](http://robomatter.com)<sup>[6]</sup>

6. **Impact** – Computers have had a global impact on the way we think and live. The way we work, play, collaborate, communicate, and do business has changed dramatically in recent years and will likely continue to change. It's important for students to understand the global impact of computing in everyday life, and the numerous ways computing helps enable innovation in other fields.

One way to help students understand the impact of computer science is to use activities that involve things like the internet, cybersecurity, internet searches, and the power of programming within advertising. You can also create activities that ask students to connect their programming skills to content from other classes (science, math, etc.). Or, you can ask students to think about and report on the less obvious ways they use technology every day, such as making breakfast, driving in a car, using the self-checkout line at the grocery store, etc.

7. **Precision** – Programming is precise. It's important for students to learn that a computer program will do exactly what they tell it to do. This is especially evident with robots. If you aren't precise about what you tell your robots to do, they probably won't do what you want. However, precision does not need to be complex. Even simple programming activities can require precise, thoughtful communication – How far should the robot move? How far should it turn?

Youtube Video: <https://youtu.be/gdkIgxWvsSk>

Ultimately, we're asking students to change the way they think about giving directions. So, a great activity is to have students create a set of instructions explaining how to do a task like following a recipe, drawing a house, or making a paper airplane. Have one student provide the instructions and a second student act as a robot, doing exactly what student #1 is telling him or her to do. Most times, it quickly becomes apparent that students have not fully considered the level of detail required for programming and that they need to be more precise with how they provide instructions.

If you're looking for more ideas on how to integrate "Big Ideas" into your STEM classroom, we've embedded these "Big Ideas" into our [research-based curriculum](#)<sup>[7]</sup>, which is available for free online, or through the purchase of a classroom edition that comes with the benefits of:

- Guaranteed Uptime – Keep your classroom up, even if your internet is down.

- Zero bandwidth requirements – 30 kids accessing the same curriculum can really slow things down.
- High Quality Support – Have a question or need help getting started? You’ll have access to our best-in-class support team.
- Individual curriculum access for each student or group – With individual access to the curriculum, students can move at the instructional pace that’s right for them.

(1): <http://content.thirdway.org/publications/714/Dancing-With-Robots.pdf>

about this article **Author:** [Lee Ann Baronett](#)<sup>[8]</sup>

**Contact:** [@LeeAnnBaronett](#)<sup>[9]</sup>

**Shorturl:** <http://goo.gl/wwbYsT><sup>[10]</sup>

**Hashtag:** #STEM

**Fork / improve:** [on Github](#)<sup>[11]</sup>

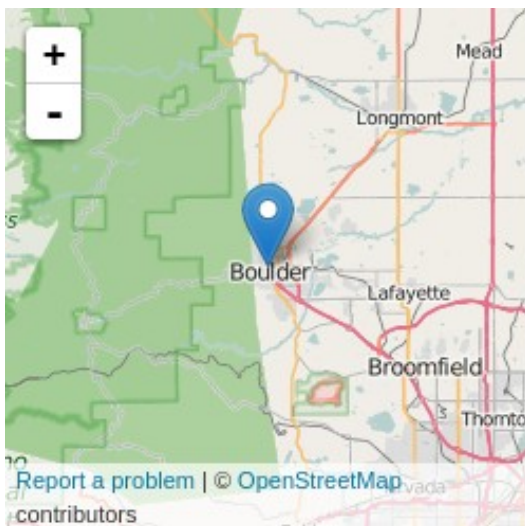
**Extras:** [original article at robomatter.com](#)<sup>[12]</sup>

**License of text:**



[cc-by-sa](#)<sup>[13]</sup>. Attribute to: [robomatter.com/big-idea-stem-classroom](http://robomatter.com/big-idea-stem-classroom)<sup>[14]</sup>

**Location:** [40.0178, -105.2737](#)<sup>[15]</sup>



**about the author(s):**

LeeAnn has always been interested in the space where technology and people meet. She has worked in education since 2011 and is especially interested in ed tech and [STEM](#)<sup>[16]</sup> education. **Donate to the authors:** No donation address given yet

explained by Wikipedia:

[Science, Technology, Engineering and Mathematics \(STEM, previously SMET\)](#)<sup>[17]</sup> is an education grouping used worldwide. The acronym refers to the academic disciplines of science, technology, engineering and mathematics. The term is typically used when addressing education policy and curriculum choices in schools to improve competitiveness in science and technology development. Read the [full Wikipedia entry](#)<sup>[18]</sup>...

[Edsger Wybe Dijkstra](#)<sup>[19]</sup> (Dutch: 11 May 1930 – 6 August 2002) was a Dutch computer scientist. A theoretical physicist by training, he worked as a programmer at the Mathematisch Centrum (Amsterdam) from 1952 to 1962. He was a professor of mathematics at the Eindhoven University of Technology

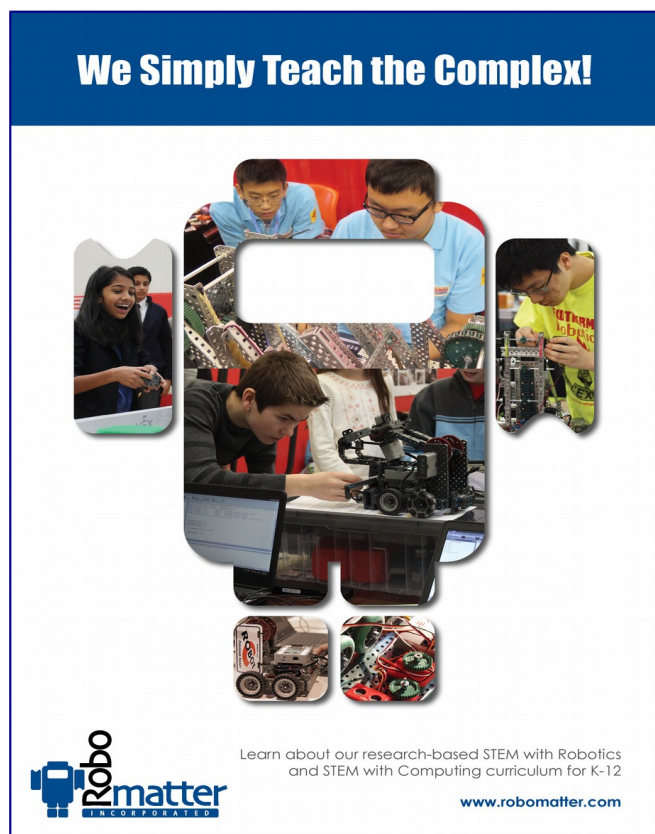
(1962–1984) and a research fellow at the Burroughs Corporation (1973–1984). He held the Schlumberger Centennial Chair in Computer Sciences at the University of Texas at Austin from 1984 until 1999, and retired as Professor Emeritus in 1999.

One of the most influential members of computing science's founding generation, Dijkstra helped shape the new discipline from both an engineering and a theoretical perspective. His fundamental contributions cover diverse areas of computing science, including compiler construction, operating systems, distributed systems, sequential and concurrent programming, programming paradigm and methodology, programming language research, program design, program development, program verification, software engineering principles, graph algorithms, and philosophical foundations of computer science and computer programming. Many of his papers are the source of new research areas. Several concepts and problems that are now standard in computer science were first identified by Dijkstra and/or bear names coined by him.

Computer programming in the 1950s to 1960s was not recognized as an academic discipline and unlike physics there were no theoretical concepts or coding systems. Dijkstra was one of the moving forces behind the acceptance of computer programming as a scientific discipline. A training background in mathematics and physics led to his applying similar disciplines of mathematical logic and methodology to computer programming. In 1968, computer programming was in a state of crisis. Dijkstra was one of a small group of academics and industrial programmers who advocated a new programming style to improve the quality of programs. Dijkstra coined the phrase "structured programming" and during the 1970s this became the new programming orthodoxy. Dijkstra's ideas about structured programming helped lay the foundations for the birth and development of the professional discipline of software engineering, enabling programmers to organize and manage increasingly complex software projects. Read the [full Wikipedia entry](#)<sup>[20]</sup>...

## original advertisement from robomatter.com

Please bundle this original advertisement image together with the article if you republish or spread the article:



**We Simply Teach the Complex!**

Learn about our research-based STEM with Robotics and STEM with Computing curriculum for K-12

[www.robomatter.com](http://www.robomatter.com)

The advertisement features a central graphic of a stylized Android robot head. The head is composed of several images: the top part shows two young men in blue shirts looking at a circuit board; the left side shows a young woman and a young man; the right side shows a young man in a yellow shirt working on a robot; the bottom part shows a young man working on a robot on a table. Below the robot head are two small images of a robot and a circuit board. The Robomatter logo is in the bottom left corner, and the website URL is in the bottom right corner.

[image rights: Robomatter.com](http://www.robomatter.com)<sup>[21]</sup>

## Hyperlinks:

- [0] [http://internationalopenmagazine.org/2015-12-10-stem\\_classroom.html](http://internationalopenmagazine.org/2015-12-10-stem_classroom.html)
- [1] <https://pixabay.com/en/robot-artificial-intelligence-woman-507811/>
- [2] <https://pixabay.com/en/mathematics-formula-physics-school-757566/>
- [3] <https://creativecommons.org/publicdomain/zero/1.0/deed.en>
- [4] <http://robomatter.com/big-idea-stem-classroom/>
- [5] <http://robomatter.com/computational-thinking/?ref=email111115>
- [6] <http://robomatter.com/big-idea-stem-classroom/>
- [7] [http://robomatter.com/stem-solutions/for-teachers/?ref=blog\\_111115](http://robomatter.com/stem-solutions/for-teachers/?ref=blog_111115)
- [8] <https://twitter.com/LeeAnnBaronett>
- [9] <https://twitter.com/LeeAnnBaronett>
- [10] <http://goo.gl/wwbYsT>
- [11] [https://github.com/horstjens/internationalopenmagazine/blob/master/content/blog/2015-12-10-stem\\_classroom.md](https://github.com/horstjens/internationalopenmagazine/blob/master/content/blog/2015-12-10-stem_classroom.md)
- [12] <http://robomatter.com/big-idea-stem-classroom/>
- [13] <https://creativecommons.org/licenses/by-sa/4.0/>
- [14] <http://robomatter.com/big-idea-stem-classroom/>
- [15] <http://www.openstreetmap.org/?mlat=40.0176&mlon=-105.2737#map=12/40.0178/-105.2737>
- [16] [https://en.wikipedia.org/wiki/Science,\\_Technology,\\_Engineering,\\_and\\_Mathematics](https://en.wikipedia.org/wiki/Science,_Technology,_Engineering,_and_Mathematics)
- [17] [https://en.wikipedia.org/wiki/Science,\\_Technology,\\_Engineering,\\_and\\_Mathematics](https://en.wikipedia.org/wiki/Science,_Technology,_Engineering,_and_Mathematics)
- [18] [https://en.wikipedia.org/wiki/Science,\\_Technology,\\_Engineering,\\_and\\_Mathematics](https://en.wikipedia.org/wiki/Science,_Technology,_Engineering,_and_Mathematics)
- [19] [https://en.wikipedia.org/wiki/Edsger\\_W.\\_Dijkstra](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra)
- [20] [https://en.wikipedia.org/wiki/Edsger\\_W.\\_Dijkstra](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra)
- [21] <http://robomatter.com/big-idea-stem-classroom/>