.NET Gadgeteer: A New Platform for K-12 Computer Science Education

Steve Hodges[a], James Scott[a], Sue Sentance[b], Colin Miller[c], Nicolas Villar[a], Scarlet Schwiderski-Grosche[a], Kerry Hammil[c], Steven Johnston[a]

a: Microsoft Research Cambridge
7 JJ Thomson Avenue
Cambridge, CB3 0FB
+44 1223 479700

b: Anglia Ruskin University
Bishop Hall Lane, Chelmsford
Essex, CM1 1SQ
+44 845 196 3519

c: Microsoft Corporation
Microsoft Way
Redmond, WA 98052
+1 425 703 4530

{shodges,jws,colinmil,nvillar,scarlets,khammil,v-stejo}@microsoft.com

sue.sentance@anglia.ac.uk

ABSTRACT

In this paper we present the features of a new physical device prototyping platform called Microsoft .NET Gadgeteer along with our initial experiences using it to teach computer science in high schools. Gadgeteer makes it easy for newcomers to electronics and computing to plug together modules with varied functionality and to program the resulting system's behavior. We believe the platform is particularly suited to teaching modern programming concepts such as object-oriented, event-based programming and it could be a timely addition to established teaching tools given the current interest in improving high school computer science education in some regions. We have run a number of pilot studies in the US and in the UK with students of varying age and ability. Our results indicate that the tangible and expressive nature of Gadgeteer helps to engage and motivate a diverse set of students. We were also pleasantly surprised by the level of polish and sophistication of the devices which were built. We hope to further explore the potential of Gadgeteer for teaching in future work and we encourage others to build on our experiences.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education–*computer science education, curriculum, information systems education, literacy*. D.2.6 [**Software Engineering**]: Programming Environments–*graphical, integrated & interactive environments*. D2.m [**Software Engineering**]: Miscellaneous–*rapid prototyping, reusable software*.

General Terms

Algorithms, Design, Experimentation, Languages, Theory.

Keywords

K-12 computer science education, prototyping platform, modular hardware, tactile learning, Microsoft .NET Gadgeteer.

SIGCSE'13, March 6–9, 2013, Denver, Colorado, USA.

1. INTRODUCTION

There is a growing acknowledgement that computer science teaching in high schools has not kept pace with the remarkable growth of computing and related technologies in society. For several years, bodies such as the Computing At School (CAS) group in the UK and the Computer Science Teachers Association (CSTA) in the US have actively promoted the need for a stronger emphasis on computer science in the curriculum and associated facilities and expertise in schools [3], [13]. A recent report by the UK Royal Society highlights an unsatisfactory level of computing education in many UK schools and states that computer science is sufficiently foundational that it should be recognized with equal importance as subjects like mathematics, physics and history [8].

As a result of this awareness, we are starting to see changes in the curriculum coupled with additional qualification options for students aged 14-18 along with increased adoption of computing-related content in schools for younger students [4]. Consequently, more than ever there is a need for engaging resources and vehicles for teaching computer science. Given the dramatic change in the way in which computer-related technologies are used in homes and workplaces – from smartphones and tablets to social networking – it is also timely to re-examine exactly what should be taught within the curriculum to "bridge the gap" between computer literacy and computer science [2].

A number of different programming environments and tools are well-established in a secondary computing education context. In particular systems such as Scratch[1] Alice[2], Greenfoot[3] and Kodu[4] have been shown to motivate and engage young people. The constructivist learning theory claims that knowledge is actively constructed by the student [1]. This suggests that active participation in problem-solving is very valuable, and critical thinking regarding a learning activity which the students find relevant and engaging is encouraged. Papert built on this concept by introducing the term 'constructionism' to indicate a combination of constructivism and hands-on construction [6].

We have recently been exploring how a new platform which we have developed – Microsoft .NET Gadgeteer[5] [11], [12] – might be used as a tool to support hands-on constuctionist computer

[1] http://scratch.mit.edu/

[2] http://www.alice.org/

[3] http://www.greenfoot.org/

[4] http://www.kodugamelab.com/

[5] http://netmf.com/gadgeteer/

science learning in high schools. Gadgeteer enables rapid creation of small electronic gadgets and digital devices. A key property of Gadgeteer is the use of physical hardware rather than the virtual environment approach taken by systems like Greenfoot, Logo and Kodu. This provides the excitement of a hands-on experience and the reward of physically building something, while also making it easy to introduce modern programming concepts like object-orientation and event-based programming.

The use of physical hardware in the context of K-12 computer science education has been explored previously through systems such as Logo-based turtles, Lego MindStorms and a variety of robot-based education products. These constructionist tools have engaged many students but are not yet established in support of the complete K-12 CS curriculum. The ScratchBoard/PicoBoard provides a tangible, hardware-based extension to Scratch which can be useful for younger students but is still quite limited both in terms of hardware flexibility and the programming experience. Modkit [5] extends the Scratch environment with a 'code view' and supports a wider variety of Arduino-based hardware but does not have the flexibility of Gadgeteer or the ability to create stand-alone electronic devices. The Sense system [7] similarly builds on Scratch and provides a modular tangible hardware experience through the SenseBoard, but is primarily aimed at adult learners.

In this paper we describe the Gadgeteer system and explain the features which we feel make it suitable for teaching. We present two initial pilots where Gadgeteer was used to teach aspects of computer science to students aged 11-18 in schools in the UK and US. We give an overview of our methodology and report on classroom experiences. We end by presenting avenues for future work which our pilots have uncovered. Our aim is to inform others who are active in the field of high-school computer science education of the potential of Gadgeteer as part of a portfolio of tools which teachers can use to engage a diverse set of students.

2. MICROSOFT .NET GADGETEER

The Microsoft .NET Gadgeteer system is an open and modular platform we created to facilitate the design, programing and construction of digital devices [12]. The ease of building new hardware coupled with powerful software development results in a very low hurdle for creating simple projects, but also supports more sophisticated applications. The open source nature of the platform[6] enables multiple manufacturers to create a diverse set of interoperable hardware. We now discuss the particular features we feel make Gadgeteer an interesting platform for CS education.

2.1 Hardware

At the heart of each Gadgeteer project there is a central 'mainboard' which includes a processor and a number of electrical sockets. The sockets may be used to attach a variety of 'modules' – different sensors, actuators, displays, communication and storage elements. Gadgeteer supports well-established protocols and formats such as USB, TCP/IP, FAT file systems, JPG/BMP/GIF images and WAV/MP3 audio files. Consequently modules can support quite sophisticated functionality including audio playback, color image display, Ethernet, WiFi, SD cards, USB keyboards, mice and cameras etc. Of course, Gadgeteer also supports simpler modules such as LED lights, buttons, joysticks, buzzers, light sensors etc. A growing number of modules are commercially available, around 100 at the time of writing.

[6] http://gadgeteer.codeplex.com/

The use of modules creates a very flexible platform with which many different devices can be built and programmed, promoting creativity. The solder-less composability of the elements means that students can construct sophisticated devices in minutes, and these can subsequently be re-configured and extended just as quickly. By using modules rather than the underlying hardware components such as individual LEDs or ICs, Gadgeteer is easier and faster to use and is more robust. It also encourages students to approach the corresponding software in a modular way.

2.2 Software development

From a programming point of view, code may be written in either C# or Visual Basic (VB). These high-level languages enable relatively sophisticated yet robust programs to be created quickly and as a result are very popular with professional programmers. The Gadgeteer system is tightly integrated with the Visual Studio (VS) integrated development environment (IDE) which is freely available as the "Express Edition" in addition to its commercial variants. While Visual Studio has a steeper learning curve than bespoke educational programming environments, it familiarizes students with the kind of professional tools used in modern software development. In particular, students can take advantage of features like automatic inline API documentation, dynamic syntax checking, real-time debugging and auto-completion of variable names, method calls, code snippets etc., see Figure 1. Also, there are no artificial ceilings in terms of the complexity of the projects and the development experience since any of Visual Studio's features can be used. For example, a shared library can be created for use by many students and projects, source control systems can be used for group project work, and so on.

2.3 Visual designer experience

Gadgeteer support is integrated into the VS IDE via Visual Studio's "plug-in" architecture. We have created a new visual representation for Gadgeteer projects which we call the "designer view", see Figure 2. This acts as the bridge between hardware and software and is the first screen that a user sees on creating a new Gadgeteer project. It specifies to the software which hardware modules are included, provides editable names for each of these, and assists the user with connecting the cables from modules to compatible mainboard sockets. The user can then go on to write C# or VB code to specify what this device actually does. Figure 2 shows the designer view during the creation of a digital camera.

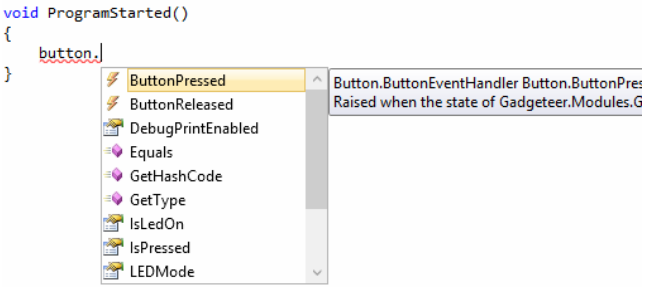

Figure 1. Real-time auto-completion in Visual Studio lists object methods, properties and so on in real-time as you type.

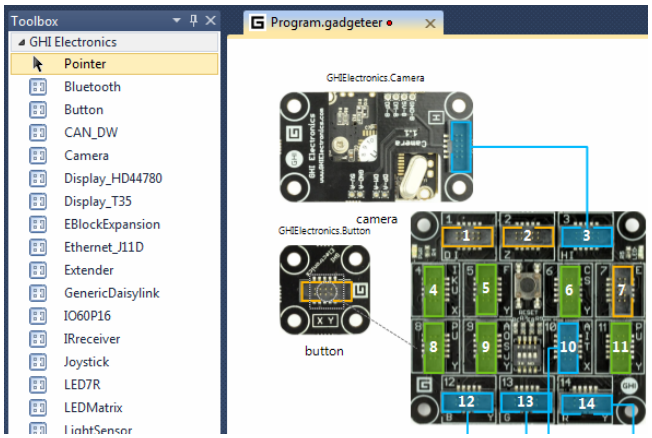

Figure 2. A 'designer view' in Visual Studio lets students create a graphical representation of the modules in a project and how they are connected to each other.

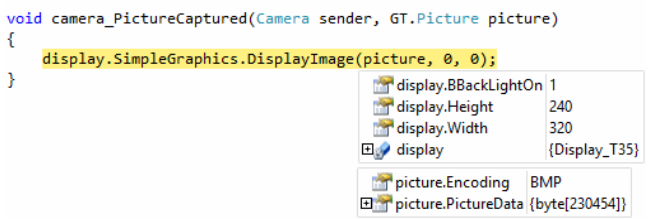

Figure 3. Real-time debugging allows students to pause program execution and examine and edit variables.

2.4 Real-time remote debugging

Through Visual Studio and the underlying .NET Micro Framework, Gadgeteer supports real-time debugging of whatever code is running on a mainboard. This debugging channel is established via USB and it enables a number of features. "Breakpoints" can be set up to cause execution to pause at specific places in a program. The state of variables can then be examined (see Figure 3) and even modified during execution; the program can be "stepped through"; and "immediate" commands can be executed as if the device was interpreting code rather than running compiled code. These features, commonly found in desktop environments, are much less common on embedded platforms but are immensely useful in a teaching context. For example, it is possible to "step through" algorithms to see how the variables interact and how control structures operate. It also gives students a realistic professional software development experience.

2.5 Programming model

Gadgeteer uses an event-driven programming model. This is in line with many modern programming environments such as Javascript, and lends itself to programming parallel behaviors like blinking two LEDs at different rates in a thread-safe manner.

The functionality of each Gadgeteer module is wrapped up in a software class which acts as an object-oriented "device driver" for the relevant hardware. This hides the underlying complexity of initializing, communicating with, and controlling that module. For example, an accelerometer driver can provide a high-level API that might include a "GetReading()" method which returns an "Acceleration" class with floating-point fields "X", "Y" and "Z" and a "Magnitude" property. The driver might also provide a "HighAcceleration" event. In this way drivers expose module functionality in a very accessible manner. Of course, it is possible for more capable students to develop their own drivers if appropriate. Note that objects can be instantiated multiple times if several identical hardware modules are used in a single project.

2.6 Bringing it all together

One short example which simultaneously illustrates the simplicity and the sophistication of Gadgeteer is the construction of a self-contained digital camera. Until very recently this would have required considerable hardware and software development skills. With Gadgeteer the first step is to use the VS Designer to specify the hardware components and how they are connected. In the case of a digital camera, an image sensor, an SD storage card, a touchscreen and a push button are needed along with the mainboard and power supply modules. Having "wired these up" graphically on-screen, it takes just a couple of minutes to construct the corresponding physical hardware, Figure 4. The basic operation of taking a photo, displaying it and saving it to a memory card can be specified in essentially just five lines of code. Auto-completion and dynamic syntax highlighting make coding incredibly easy. Figure 5 shows the resulting application in its entirety. When this application starts up, two methods which serve as event handlers are registered. The first executes when the "shutter" button is pressed, and this requests the camera to take a picture. Once the picture is captured, the second method is triggered and the picture is displayed and saved to the SD card.

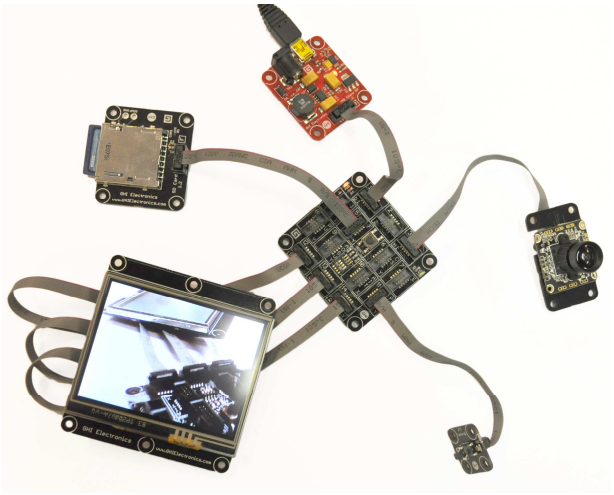

Figure 4. An example digital camera built with Gadgeteer.

```
void ProgramStarted()
{
    button.ButtonPressed +=
        new Button.ButtonEventHandler    (button_ButtonPressed);
    camera.PictureCaptured +=
        new Camera.PictureCapturedEventHandler(camera_PictureCaptured);
}

void camera_PictureCaptured(Camera sender, GT.Picture picture)

{
    //Show the picture on the display
    display_T35.SimpleGraphics.DisplayImage(picture,0,0);

    //Save the picture to the SD card
    sdCard.GetStorageDevice().WriteFile("picture.bmp",
                                        picture.PictureData);
}

void button_ButtonPressed(Button sender, Button.ButtonState state)
{
    camera.TakePicture();
}
```

Figure 5. The entire camera application is just 5 lines of code.

3. CLASSROOM EXPERIENCES

In the 2011/2012 school year we trialed Gadgeteer in the classroom to gain experience of its suitability for high school computer science education. We ran a total of five pilot studies: two in the UK with 11-17 year-olds and three in the US with 14-18 year-olds. These pilots were implemented differently in the two regions, and are described in more detail below.

3.1 UK Pilots

The two UK pilots involved 16 secondary schools, mostly state-funded and mixed gender. Teachers willing to trial Gadgeteer in the classroom were recruited and given initial training either in groups or 1:1. They were also provided with session plans and example projects; C# was used throughout. Gadgeteer was then taught for around 12 weeks, mostly in extra-curricular 'lunchtime' or 'after-school' clubs attended by students ranging from 11 to 17 years old. The students typically worked in groups of 3-4 with one Gadgeteer Fez Spider[7] starter kit from GHI electronics per group.

The session plans consisted of 8 separate modules with approximately one hour's teaching material in each. They contained all the instructions for carrying out the tasks including code samples. Following the first UK pilot all the materials were refined based on feedback and observations and they are available online[8]. Towards the end of the 12 week duration of each pilot, students were invited to invent and program a device of their own design, and enter this into a competition. The students enjoyed being creative and implementing their own ideas. Prizes were given for originality and technical merit.

Approximately 150 students were involved in the UK pilot, attending an average of 8 sessions each, although this varied significantly from school to school. More details of the first pilot are presented in [10]. Feedback gathered from students and staff is presented in Section 3.3.

3.2 US Pilots

In the US Gadgeteer was used in direct support of computer science curriculum teaching in three suburban public high schools around Seattle, WA. Two courses were offered, 'Introduction to Computer Science' and 'High School Computer Programming'. The latter class provided an option for college credits. Both courses were one semester long, about 75 class hours of instruction, and as a result covered much more depth than the UK pilots. One course was taught by an accredited teacher and one was taught under supervision of an accredited teacher. Both classes were open to students from freshman through senior (14-18 years old).

The courses were intended to raise the students' proficiency in computer science concepts and computer programming to allow them to progress to more advanced high school classes like Advanced Placement (AP) and International Baccalaureate (IB) or to pursue engineering at college level. In addition to material on computer organization, data structures, algorithms and programming constructs (looping, conditional logic, variables) the courses focused on 'computational thinking' [14]. This provided students with exposure to generally applicable problem decomposition and analysis techniques.

[7] http://www.ghielectronics.com/catalog/product/297

[8] http://netmf.com/gadgeteer/forum/

In the 'Introduction to Computer Science' course, the students spent several weeks becoming familiar with programming constructs by using Scratch to create several games, one of which they controlled using an Xbox Kinect gaming controller.

Following the introductory period, for the remainder of the semester the students worked in Visual Studio C# Express Edition and used Gadgeteer. They completed several small projects and one larger one. The large project involved creating a fully functional digital camera. The students programmed the camera so that it could take multiple pictures, store them to an SD card, scroll through them, etc. User interaction with the camera was supported via a large Gadgeteer touch screen, requiring the students to create an on-screen menu system and interpret user input. The hardware modules were encased in a laser-cut wooden enclosure provided to the students which allowed them to take the camera out of the classroom to take part in a photo contest.

3.3 Feedback and outcomes

Students were generally very positive about the way that Gadgeteer gave them the freedom to develop their own devices:

> *"You're allowed to be sort of creative and sort of like make anything – so you weren't really limited to what you can make. You're in control; you can take an idea anywhere and use the hardware that's available..."* (Student, 1st UK pilot)

> *"It's nice when you finish a product …and you're always looking to improve in certain areas, like can I make this camera function better, could I make it hook to an SD card, could I enable it so that it has internet connectivity. It's asking those sort of questions .. that's really enabled me to enjoy such projects."* (Student, 1st UK pilot)

> *"I thought overall this semester was a success. It got really confusing sometimes but once you got it, it was awesome."* (Student, 1st Semester US Pilot)

> *"Overall it was a great learning experience for me and has helped me to want to get into the computer field more."* (Student, 1st Semester US Pilot)

> *"Now I have an idea of how most of the electronics I own works. It gave me higher expectations for the future."* (Student, 1st Semester US Pilot)

Feedback from the teachers confirmed the students' views, they felt that the Gadgeteer kits were engaging to use and in particular developed determination and resilience in students because they were motivated to get their devices to work and to extend them:

> *"The technology of gadgets inspired them to work a little harder & play with ideas more."* (Teacher, second UK pilot)

> *"… they actually wanted to solve it and they would keep trying to solve it … that kind of resilience you need to have to be able to keep going"* (Teacher, first UK pilot)

Interestingly, several students indicated that they would like to gain a deeper understanding of and experience with Gadgeteer:

> *"I would much rather this class be a year long instead of a semester."* (Student, 1st Semester US Pilot)

> *"Now that I know how to use it, I want to learn more with it. I think that it is amazing how you can make something work with simple computer code."* (Student, 1st Semester US Pilot)

Devices built by our high school participants included iSpyder and Gadgesketch. The iSpyder (Figure 6) is a double-sided device

in a wooden case built by the students. One side has a touch screen which displays the temperature when a button is pressed. The other side acts as a coaster for a hot drink. The iSpyder tests the temperature of the drink; when it reaches a temperature cool enough to drink an internal buzzer will sound to alert the user.

Gadgesketch (Figure 6) is a remake of a classic children's sketching toy, by a pair of 17 year-olds. The students programmed the device by reading potentiometer values and translating them to on-screen graphics. The housing was designed in the Pro/Desktop CAD package and then vacuum formed from acrylic. The modules were mounted inside using engineering materials. The students are now planning to revise the housing and add a tilt switch via the extender module allowing the iconic "shake to erase" feature.

Other devices built by students demonstrated their ability to design graphics for the 3.5" touch screen and build engaging and interactive games, for example the "How wide is your goal" game, "FaceBooth" and "Rainbow Press", all shown in Figure 6.

3.4 Discussion

The pilots were successful in their primary objective of assessing whether Gadgeteer has potential for computer science education in the classroom. In our work with high-school students of a range of ages from both the US and the UK its tangible nature seemed to engender curiosity and motivate the students to push themselves. They enjoyed the creativity the platform supports and gained a great deal of satisfaction from building real, tangible devices. We also saw how Gadgeteer lends itself to collaborative working, whereby students with a range of skills and abilities can support and learn from each other. In this way we believe that Gadgeteer engenders valuable inter-personal skills in addition to the specific technical skills which are actively being taught.

We were pleased how quickly the students mastered the VS professional development environment. One 13 year-old student commented that he was using a "*proper programming language*" and appreciated that he was using real tools rather than an educational environment. Initial feedback from the first UK pilot suggested that the younger students (aged 11-13) found the programming difficult unless they had had some prior experience. More engagement was experienced with students in the 14-18 age range. We also saw more engagement from those pupils who were able to use Gadgeteer within the curriculum rather than solely as an extra-curricular activity.

As with the UK experience, in the US the age of the students played a role in the level of difficulty that they encountered mastering programming. Students who had at least been taught geometry seemed to be better equipped for the challenge. However, all students were able to complete the projects with support from the instructors, in part because of the larger time commitment in these courses.

In summary, the qualitative feedback to date indicates that Gadgeteer is a motivating environment for teaching programming to 14-18 year-olds. Recent changes in the computer science curriculum [9] mean that this will now be possible in the UK, as it was during our US pilot. With Gadgeteer we hope to engage students of all abilities, many ages and both genders. In addition to teaching computing, we hope to give students a better understanding of modern devices and technology in general.

Figure 6. Example devices built during our high-school pilots. The "i-Spyder" (top left) alerts you when your drink is cool; "How wide is your goal" is a football game (top right); the GadgeSketch drawing device with its vacuum-formed case (middle left); "FaceBooth" (middle right) lets you take a photo and then annotate it with cartoon eyes, nose and mouth; "Rainbow Press" is a reaction game. The bottom row shows digital camera construction in the US trials.

4. FUTURE PLANS

As an initial pilot study with a new platform, our work with Gadgeteer has inevitably highlighted a large number of interesting avenues for future research. We discuss some of these here.

Initial feedback from teachers in indicates that Visual Basic would have a much wider appeal than C# because it is relatively well-established for high-school teaching. Following recent technical improvements to the Gadgeteer software framework, the platform now supports VB and we are currently developing new teaching materials which cover key programming concepts in the K-12 CS curriculum such as variables, functions, conditionals,

file handling, looping and arrays. We would like to get experience with VB in the classroom and compare this with C#.

We hope to leverage the appeal of building real, tangible devices to engage a more diverse set of students than has traditionally been the case in computer science. We would also like to gain experience of a more holistic approach to education which cuts across the wider curriculum by including Gadgeteer in the teaching of subjects beyond computer science. Perhaps the most obvious example is the creation of enclosures for devices built with Gadgeteer, which could be integrated with design and technology teaching. Each Gadgeteer module includes standardized mounting holes which facilitate the use of a number of physical construction tools ranging from hot melt glue and self-adhesive tape right up to 3D CAD tools and rapid prototyping equipment such as laser cutters and 3D printers. Other examples of potential cross-curriculum integration include building an environmental sensor and recorder which could be deployed and subsequently analyzed in an environmental studies session, or the creation of experimental equipment such as a light-beam-triggered digital timer as part of a physics experiment.

We want to create a wider range of example projects that will inspire students and teach deeper computer science principles in a way which is particularly relevant to today's sophisticated consumer electronics devices. Examples would include a portable games console, an MP3 player and even a smartphone – the elements of which are all available as Gadgeteer modules.

At the same time as extending the depth and breadth of our Gadgeteer teaching materials, we think it is also very important to investigate the ways in which Gadgeteer can facilitate the students' understanding and acquisition of programming concepts. We want to maintain the exploratory and experiential approach which seemed to work so well in our pilots to date, and enhance it with more staged support which leverages the sense of challenge and creativity we saw in our participants. We hope to explore whether Gadgeteer can assist and motivate students who typically find it harder to engage with computer science and in particular address the gender imbalance which has traditionally been seen in the uptake of computing. Ultimately we hope to facilitate a secure and robust understanding of computer science.

Finally, we would like to trial Gadgeteer at university level for teaching undergraduate computer science courses. There is a lot of headroom in Gadgeteer for supporting sophisticated programming concepts such as threading, inter-process communication, device drivers, algorithms, data structures, and advanced networking.

5. CONCLUSIONS

This paper presented the Microsoft .NET Gadgeteer platform, highlighting in particular the various features which lend themselves to teaching computer science at high school level. In particular, Gadgeteer exposes students to many of the concepts which are an intrinsic part of modern software development. It facilitates a pedagogy based around teaching events first and object orientation naturally follows the modular hardware. The platform is growing rapidly and the increasing range of modules means it can be incorporated in cross-curricular contexts.

Our experience of using Gadgeteer and the initial feedback from teachers and students alike has been favorable. In addition to technical skills, Gadgeteer also appears to support the development of personal and inter-personal skills such as creativity, problem-solving, independent learning and collaborative working.

The Gadgeteer platform is still in its infancy and there are clearly a great many unanswered research questions relating to its use as a tool for teaching. Nonetheless, we hope that others will find our initial experiences valuable and build on them in future work.

6. ACKNOWLEDGMENTS

We would like to thank to the many teachers and students who supported our pilots so enthusiastically and the large number of people who have contributed to the Gadgeteer platform.

7. REFERENCES

[1] Ben-Ari, M. Constructivism in Computer Science Education. In proceedings of the 29th SIGCSE Technical Symposium, ACM. 1998.

[2] T. Brinda, H. Puhlmann and C. Schulte. Bridging ICT & CS - Educational Standards for Computer Science in Lower Secondary Education. In Proceedings of ITICSE, July 2009.

[3] T. Crick and S. Sentance. Computing At School: Stimulating Computing Education in the UK. In Proceedings of the 11th Koli Calling International Conference on Computing Education Research, pages 122-123, ACM, 2011.

[4] O. Hazzan, J. Gal-Ezer and L. Blum. A Model for High School Computer Science Education: The Four Key Elements That Make It! In Proceedings of the 39th SIGCSE Technical Symposium, pages 281–285. ACM, 2008.

[5] A. Millner and E. Baafi. Modkit: Blending and Extending Approachable Platforms for Creating Computer Programs and Interactive Objects, In Proceedings of the 2011 ACM Interaction Design and Children Conference.

[6] S. Papert. Mindstorms: Children, Computers and Powerful Ideas. Basic Books, 1993. 92053249.

[7] M. Richards, M. Petre and A. K. Bandara. Starting with UbiComp: Using the Senseboard to Introduce Computing. In Proceedings of the 43rd SIGCSE Technical Symposium, pages 583-588, ACM, 2012.

[8] The Royal Society. Shut Down or Restart? The Way Forward for Computing in UK Schools. Technical Report, January 2012 DES2448, The Royal Society, 2012.

[9] S. Sentance. Changes Afoot in the UK. CSTA Voice, Volume 8 Issue 3. July 2012. Available at: http://csta.acm.org/Communications/sub/CSTAVoice.html

[10] S. Sentence and S. Schwiderski-Grosche, Challenge and Creativity: Using .NET Gadgeteer in Schools, In Proceedings of the 7th Workshop on Primary and Secondary Computing Education, WIPCSE 2012.

[11] N. Villar, J. Scott and S. Hodges. Prototyping with Microsoft .NET Gadgeteer. In Proceedings of the 5th International Conference on Tangible, Embedded, and Embodied Interaction, TEI '11, pages 377-380. ACM.

[12] N. Villar, et al., .NET Gadgeteer: A Platform for Custom Devices. In Proceedings of Pervasive 2012, Lecture Notes in Computer Science.

[13] C. Wilson, L. A. Sudol, C. Stephenson and M. Stehlik. Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age. Technical Report CSTA. 2010.

[14] J. Wing "Computational Thinking", Communications of the ACM, March 2006.