Critical Thinking Sheet (CTS) for design thinking in programming courses

J. C. Roberts[1] and P.D.Ritsos[1]

[1]Bangor University, UK

Abstract

We present a quick design process, which encourages learners to sketch their design, reflect on the main algorithm and consider how to implement it. In-depth design processes have their advantages, but often are not practical within the time given to the student, and may not fit the learning outcomes of the module. Without any planning students often jump into coding without contemplating what they will do, leading to failure or poor design. Our single-sheet method, allows the learners to critically think of the challenge and decompose the problem into several subproblems (the appearance, functionality and algorithmic steps of the solution). We have successfully used this technique for three years in a second year computer graphics module, for undergraduate degree students studying Computer Science. We present our method, explain how we use it with second year computer graphics students, and discuss student's experiences with the method.

Categories and Subject Descriptors (according to ACM CCS): Interfaces and Presentation [H.5.2]: User Interfaces—Graphical User interfaces (GUI). Computing Milieux [K.3.2]: Computers & Education—Computer Science Education

1. Introduction

Learners and developers often need a quick way to critically think about their program before implementing it. Especially when programming visual computing interfaces, developers need to contemplate what the interface will look like, what algorithmic functionality needs to be implemented, and what components are required. In this work we propose a single worksheet (the Critical Thinking Sheet, CTS), to encourage critical thought and sketched planning by the individual. By stepping away from the computer, and thinking critically about their task the students will create better programs: we advocate that students benefit from using the CTS and reflecting on the problem, and thoughtfully preparing their solution before jumping into code.

Our vision, is to enable teachers and learners to prepare for the future. We wish to upskill the next generation of computing developers and researchers: individuals who are confident designers, critical thinkers, reflective practitioners, and competent in visual computing development. Individuals who are critical computational thinkers. This work fits within this vision. In fact, Computational Thinking (CT) has become a widely cited approach to problem solving [Win06]. CT emphasises conceptualising and thinking like a computer scientist, being a skilled computer programmer, thinking creatively, and combining mathematical and engineering thinking. While the idea is sound there is little guidance how to apply computational thinking in reality.

In this paper we provide one way to help learners think critically about their work before coding. It is a quick method, meant to encourage critical thought, and to help students decompose the problem into subproblems. It fulfils a different purpose, but remains complementary to wireframing and UI development techniques or ideation methods such as the Five Design-Sheet [RHR16]. Indeed, it is not an in-depth design-study, but meets the challenge of providing a structure to help students reflect on the problem from different viewpoints, and move from receiving a "problem description" into understanding better possible "problem solutions".

In the background and Related Work (Section 2), we put the work into context and describe previous research. We explain the method (Section 3) and present a visual programming exercise scenario along with the results of students using the method (Section 4). Finally we report on reflections from our students, discuss student experiences and conclude.

2. Related Work

There are huge benefits to *visual thinking*. For example, pictures are more easily remembered in comparison to words [Pai75], visual imagery encourages learning associations [San77], and drawings of problems help to guide conjecture [Anz91]. Sketching and diagramming requires students to contemplate abstract ideas, it helps them slow down and think carefully about their programming and explicitly elicit the ideas in their mind [RRJH18]. Visual sketching can help students discuss their ideas better with their peers and thus learn different ways to do their tasks [RHR17]. In graphics teach-

DOI: 10.2312/eged.20201029

ing tutors often sketch ideas on paper to explain specific concepts, lecturers will draw diagrams on white boards to explain algorithms, and we use animated graphics in our slides to visually explain concepts such as ray tracing or radiosity. Teachers even get students to create visual artefacts, so why don't we (as teachers) ask the students to plan in a visual way first? Indeed, given the many benefits to sketching to critically think and plan, it seems surprising that most subjects and skills rarely use sketching, and it is not widely taught at an undergraduate level. Sketched planning, wire-framing and other low-fidelity techniques are certainly used in User Interface modules, and in some STEM subjects [GAOS17] and geology [JR05].

One solution, to help students analyse the task, is to get them to perform a design-study. Many methods exist in this space to help students address open-ended questions [Sim73]. Such as models by the Design Council [Cou07], Munzner's nested model for visualisation design [Mun09], the understand, ideate, make, deploy process of McKenna et al. [MMAM14], the nine-stage design-study model by Sedlmair et al. [SMM12] and the Five Design-Sheets method [RHR16]. Yet, these activities focus on creativity and alternative design solutions. While important aims, they are not necessarily the right learning outcomes for a graphics module, especially while the student is still developing their programming skills. Furthermore, it is not trivial to perform such studies, consequently students take many hours to complete the design-study, time that a second year student often does not have. What is needed is a method that can encourage students to think about the problem from different viewpoints [Dij82], which is not a full design-study, and does not take too long.

There are many methods that we could consider to help students learn particular skills, but none of these fit our goals of a quick computational thinking method. For example, approaches using making [LÏ6] or token and construction [HCT*14] can help learners develop their computational thinking skills. Brainstorming tools such as VisitCards [HA17] can help learners consider novel solutions. Ad hoc sketching [Bux10] and wireframing design software focuses the students' mind on the design appearance of a solution. Unified Modelling Language (UML) [BRJ98] could be used to help define the underpinning components (however, like the design-study methods, UML diagramming takes time to master and create). Nelson's rules help with design competence [NS12], while the waterfall model (of requirements, design, implementation, verification, maintenance), and instructional models (e.g., ADDIE [BRC*75], Jonassen [Jon97]) help students frame their problems against real-world projects. Prototypes would enable students to explore solutions e.g., Lim et al. [LST08]) but like the design-study will take more time to perform.

Our goals are to get the students to (1) understand the question asked of them, (2) apply knowledge they have learnt to develop a solution, and (3) analyze and consider the task from different viewpoints [Dij82] and divide it into several sub-problems: to consider the appearance, underpinning algorithm, and functionality of the problem. We want a technique that was quick to perform, that will record their initial thoughts (that they can then submit for grading), which will demonstrate that they have considered all steps.

3. Design of the Critical Thinking Sheet

For the past four years we have been leading a second year Computer Graphics and Algorithms module, with over 80 students each year, which sits within the second year of a BSc Computer Science programme. The aim of the module is to "teach fundamental computer graphics algorithms and techniques for computer graphics, and enable students to gain skills to code graphics programs, and understand fundamental algorithmic concepts that can be applied across computer science". The content of this module includes: information about graphics libraries (OpenGL and Processing.org) and graphics standards, rendering algorithms (Z-buffer and Ray tracing), discrete algorithms such Bresenham's line drawing algorithm, various boundary fill (flood fill) and scanline algorithms, and local illumination models (Phong illumination). The module has weekly lectures with corresponding practical laboratories, where students are given coding tasks, ranging from drawing grids of squares, to developing simple line-drawing and flood fill discrete algorithms.

During one lab session, we realised that several students were confused about implementing a simple line-drawing algorithm. We therefore gathered all students around and illustrated the algorithm through sketching on a large sheet of paper. To follow up this activity we got the students to do their own sketches when they were considering their own tasks. Following weeks, it became a tradition to sketch the lab questions before starting to code. We brought pens and pencils and plain sheets of paper to the labs, such that students could sketch, make notes and try to think computationally about their tasks.

During the academic year we started to observe several other student behaviours, after following this method. First, some students were frightened to start sketching and planning on a blank sheet of paper. When asked, students were anxious: "we don't know where to start", "what if it is wrong", and "what if I make a mistake, how can I get it right?". We noticed this very same problem when using the Five Design-Sheet [RHR16] method; we and other researchers have discussed anxieties of starting (e.g., [RRJH18, RHR17]). Second, some students were not sure of what was required. Some wrote individual words, others bulleted lists. Third, some students did not use the sheet of paper at all, choosing to ignore any planning. Fourth, other students hacked the task first and afterwards copied their code onto the sheet, to fulfil any request to plan via a sheet of paper. On reflection, we realised that students were using the sheets for different purposes. Sometimes they sketched what the output would look like, on some occasions the students drew a flow chart of the stages of the algorithm, and on other occasions noted the main stages of the algorithm as pseudo code. Therefore, we decided to put structure to the sheet, to guide the students to consider the problem from different viewpoints, and we gave some instruction and several completed sheets as examples of its use.

We followed Dijkstra's approach to divide the challenge into several sub-problems [Dij82], and the ideology of "separation of concerns". We contemplated several different strategies to categorise these sub-problems. We considered mnemonics such as who, what, why, when, but felt that this was not suitable for this task because it does not encourage the students to thing about how the solution will be created, or what it would the sub-components would be.

John Dewey, suggests to recognise the problem, define it, suggest solutions, reason each solution, and then believe or disbelieve each solution [Dew10]. From this we learn that it is important first to acknowledge the problem, and then break the solution down into parts that can be solved. Subsequently, we wanted to have a section where the students can explain (in their own words) what is the problem they are set. George Polya, in his book titled "how to solve it" similarly describes that to solve a problem you must understand it, then devise a plan from your own experience, carry out that plan, and then reflect on your answer [Ṕ73]. We were also keen to have a section for students to sketch, to encourage them to think about the problem visually. In addition, we wanted to separate the analysis part into high-level concerns, and low level algorithmic steps. Subsequently, after much deliberation, we decided on five sub-problems, shown in Figure 1, each one focusing on a separate concern:

- In the panel labelled **"what is the challenge?"** students are asked to summarise the challenge. They should articulate the goals of the tool, and list any assumptions that need to be made by the developer to achieve the goal. This should be the first section they complete. It is important that the student understands and can articulate the challenge in their own words. Without knowing the challenge, they will not be able to find a suitable solution.
- On the **sketchpad** area we ask the students to make a sketch of the algorithm output. In a graphical task, this would be a technical illustration of the output. In other words, the output is sketched along with annotations of sizes, lengths, part names, etc. If the challenge is an interface, we encourage a picture of the user-interface along with similar labels. If the task is to develop an algorithm, then a conceptual diagram could be sketched that would explain the algorithmic solution.
- Students are asked to consider **what are the parts/components.** One strategy would be to consider how it could be implemented. Another strategy can be to apply knowledge of design patterns [GHJV95, HA06]. We encourage the students to ask themselves: "can you explain the work in terms of a class, method or a design pattern?"
- Consider **what are the algorithmic steps?** What is the high-level set of processes? What are the main steps that the algorithm needs to take? What are the start conditions, the main loop and consider how the algorithm will terminate or complete? When the main steps have been considered the student should then consider if there are any detailed steps that need to be added, and then review the whole algorithm.
- Finally, they should **reflect how to take the idea forward**. Is there anything to consider when implementing it? Do they need to do more research? How can the ideas be implemented? Have any assumptions been made? Is the algorithm complete; is it too simple or too complex?

In addition, and when using the sheet we encourage the students to follow Polya's four stage problem solving strategy (see [RHR17, Ṕ73]) as follows: (1) make sure they understand the problem, (2) use the CTS to ideate a plan that will help develop a solution, (3) reflect on the CTS plan, and carry it out by implementing the algorithm in code and (4) reflect on their coding solution and the use of the CTS.

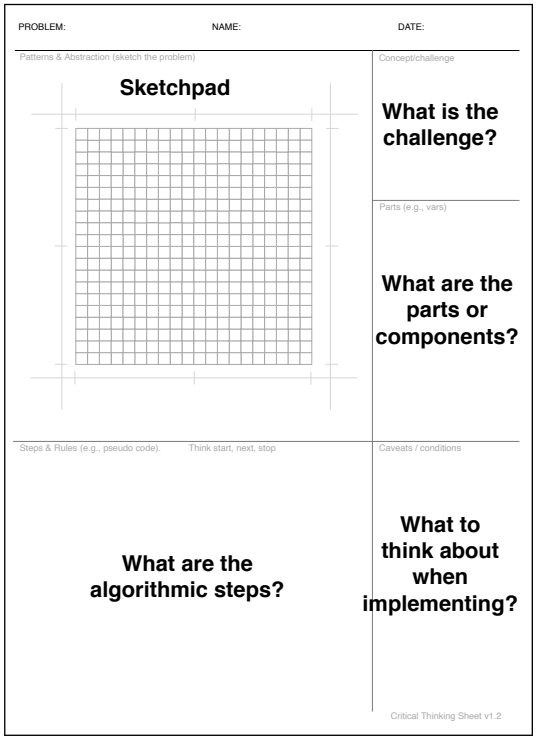

Figure 1: *Critical Thinking Sheet, with five areas: sketchpad, what is the challenge, components, algorithmic steps, and reflection.*

4. The Critical Thinking Sheet (CTS) in our teaching

In the second-year Computer Graphics and Algorithms module we first get the students to use the CTS in five classroom sessions each with three tasks, and one coursework programming assessment. We expect students to spend a few minutes on each CTS sheet. Once they have read through the task they should spend enough time to think about the problem, work out a solution, and put this down on the CTS sheet. They should do this task as quickly as possible, without wasting time to make the sheet look perfect. The actual length they spend on the CTS can vary, and depends on the complexity of the task.

The tasks in our labwork gradually increase in difficulty. Five assessed labs are given, each matching with the lecture of that topic. Each lab has three core tasks to complete. Good students finish in about 40 minutes, whereas weaker students take the full two hours that is allocated to each session. They are able to ask for help from the lab assistants. Students submit a logbook of their labwork demonstrating their work, which includes scanned copies of their CTS sheets, along with code snippets, full code, result screenshots and reflection of work done. This is graded and accounts for 10% of the mark for the module. It is graded using a simple three-stage marking strategy (above average, average, and poor). The idea is that students learn how to use the CTS sheets, gain formative and summative feedback on their work, improve their coding skills and develop good practices. They start by drawing coloured grids and triangular patterns, which gets them using the graphics library. Develop their own line-drawing algorithm (through plotting rectan-

gles), and then create gridded, circular and hierarchical patterns, before focusing on a small plotting program where they create their own rectangle fill algorithm, and circle plotting algorithm (plotted in rectangles, such to mimic a frame buffer). Figure 2 shows three labwork examples, from three different students.

In addition they receive a large self-study assessment. We get the students to develop an interactive pixel-pattern tesselator. The idea is to design and implement a wallpaper pattern editor; a pattern in a small tile of pixels is replicated across the main design space. The tile can be rotated, translated, mirrored and so on, to create different patterns. To achieve this, students need to understand transformations, hold the state of the wallpaper patterns in an appropriate data structure, and develop a simple interface. They are encouraged to follow the Model View Controller (MVC) pattern to achieve their implementation. We have run this process for three years, one year using OpenGL, and the final two years with the Processing library (processing.org). This assessment is worth 30% of the module.

We asked the students to write a critical reflective report on their work that explains their tool, includes at least one screenshot, code snippets, and makes an in-depth critical reflection of how they used the critical thinking sheet. Figure 3 shows the results of two students. They also submit their code, and a movie of their tool working. Finally they take an exam, worth 60% of the 5 ECTS credit bearing module.

4.1. Reflection on student feedback

Because we asked every student to write a reflective report and comment on the processes and learning that they achieved, we have a broad set of comments. Below we report on a representative sample of their feedback.

James said "*normally, I don't use any sort of planning material, as code I write can undergo drastic changes at any moment. However, using the Critical Thinking Sheet for this task was a great benefit, as the Steps and Rules section allowed me to plan it out rather effectively, as well as scribble down any ideas I thought could be included in the final version.*" This is a positive outcome and shows that he thought about how the sheet had been used. James did not complete the work, but was able to sketch his ideas on the sheet, getting some marks for his design. He went on to say "*a few ideas I had weren't included within the final version due to my own inability to effectively code them, such as the possible inclusion of a UI or inverting the colour scheme from RGB to CMY.*" Another student, Lisa, obviously wanted to go further in her implementation, writing "*More advanced patterns, . . . could have been added to make design more creative, as explained in the Critical Thinking Sheet.*" This is again positive, because she has reflected on the sheet, and understood that she could design something different to her final implementation.

It was also good to read how the sheets were used to design the interface. For example, Raj wrote "*the critical thinking sheets were useful in laying out my design before starting my code, it also allowed me to work out all the measurements in advanced rather than trial and error each time you run the code etc.*"

Frank added many positive comments about his experience with the assessment. He submitted several sheets, and clearly enjoyed the planning and the coding tasks: "*the first critical sheet I created was when I just started working on the project. I had an idea of how to create the tool, but actually making a quick design and noting down some initial ideas helped me. By completing this sheet, I was able to see how the pattern could be repeated.*" He understood our ideology, saying "*if I started by going straight to writing the code, it's likely that I would have forgotten about the push/pop matrix, meaning I would be wasting time trying to fix errors.*" In fact, this is what we had hoped for. He had carefully considered the challenge, drawn on his experiences, designed something that worked well and reflected on his work. Another student, let's call her Gillian, made a very similar comment: "*The main benefit of using a critical thinking sheet was in solving this problem by allowing me to visually demonstrate where the tiles and squares needed to be rather than making guesses about the offset formula which should be used*".

But not all students liked sketching or planning using the sheet, some did not like to perform any planning at all. Jack said "*I didn't use any critical thinking sheets for this assignment. While I am sure they are useful for some, I found them more hindrance than help in the previous labs, and so in the interest of time ignored them. People learn and work in many ways; for me personally, the critical thinking sheets were not helpful. I would work through the critical thinking sheet but by the end of the assignment I had changed so much, often due to an initial oversight or new idea, that it became barely relevant.*" In one respect, this student has a correct understanding: certainly, some ideas (on some sheets) will need to be thrown away. The CTS is a tool to help students develop their thoughts, and should be treated as such and not an arduous task. It is designed to move students from a basic understanding of the task into a deep understanding. In addition, we acknowledge that it also does take time to complete. However, learning takes time, and careful critical thinking takes time too. Francis also wrote negatively, saying "*I struggle to gain anything much of value before sitting down and trying to bash some code out . . . and if I get stuck . . . I do scrawl some working out in my notebook*". In fact, she did submit some sketches, but these did not follow the CTS structure. Even with these sketches he exhibited critical thought and ideation.

5. Discussion

There are clearly some students who enjoy using the sheet, while others do not. We found it difficult to encourage every student to use it, and some insisted that they were fine without it. At times we felt exasperated with our students in the 2nd year graphics course. We are not alone with these frustrations, as exemplified by books such as "getting the buggers to think" [Cow07]. Not only does this title express the huge exasperation of the teacher, but it provides practical advice and exercises to help teachers get students to think. We have had some students argue with us that they should not use the critical thinking sheet, that they knew exactly what was expected of them, and that they knew how to code it. But then these very same students (half and hour later) were still typing and guessing different coordinates to move a triangle vertex left a little bit more. If only they had thought about the problem, used the sheet, they would have realised a simple solution — to use inbuilt variables of

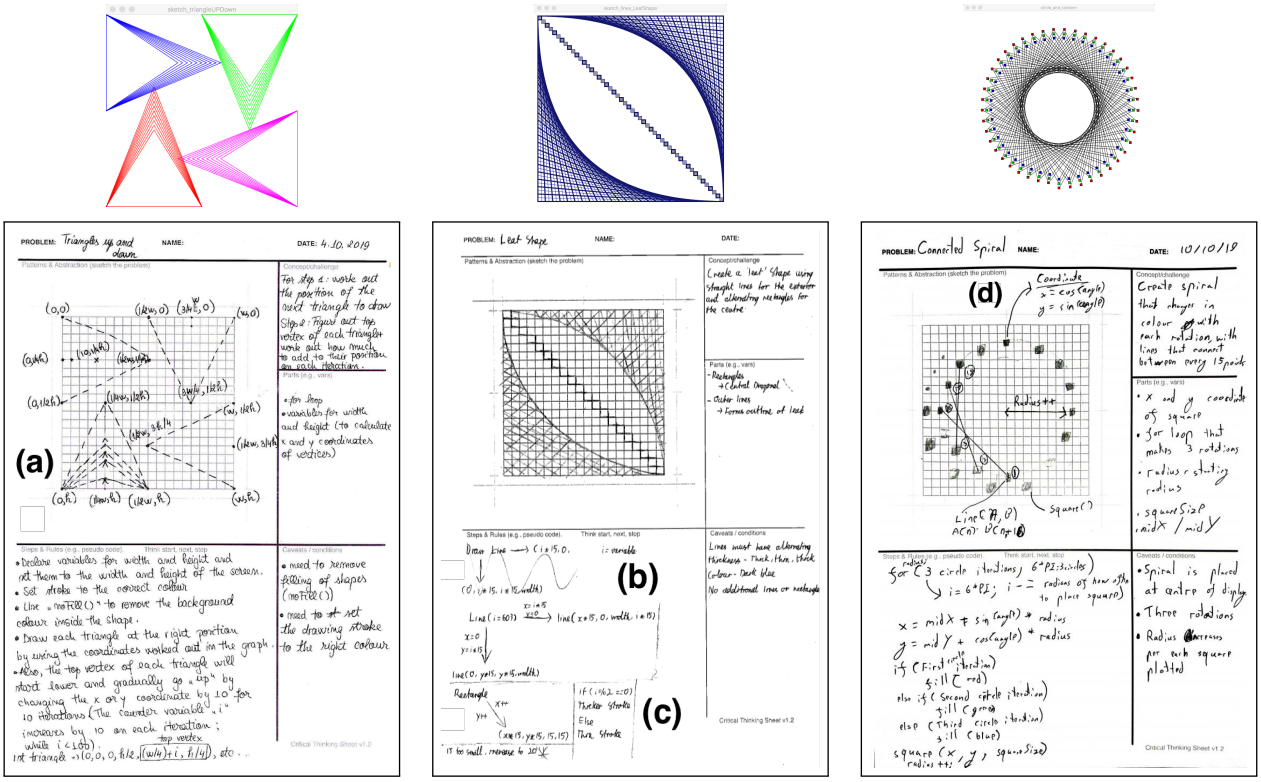

Figure 2: *Three labwork examples. Notice how (a) the student calculated the triangle coordinates based on their proportion to the whole square, such as* $(\frac{1}{2}width)$*. In the Leaf example (middle) the student noticed that the design can be broken into three distinct steps, first the lines at the top and then the bottom to create the smooth arc shape (b), and then add the squares down the centres, alternating by checking if* $(i\%2 == 0)$ *(c). Finally, this student worked out how to place the rectangles in a spiral using* $cos(\theta)$ *and* $sin(\theta)$ *(d), before drawing the joining lines.*

width and height, and multiplied by $\frac{1}{2}$ or $\frac{1}{4}$ — to get exact positions. Even when we drew the values on the sheet of paper, they still argued that the sheet was not useful. In one particular case, it was only when their peer said "there you go, if you had used the sheet then it would have saved you 50 minutes of wasted time". In some cases, the benefits of the tasks were only really understood after failing, or wasting time. While the journey, in this case, was hard, the student has now realised the benefits of the process.

Indeed, peer pressure, the reactions and acceptances of the task by peers, and the learning environment all help (or hinder) the uptake of any learning method. Having a positive attitude to the work is important. Both for the student and the teacher. If a student wants to think critically then they will; if they don't want to do the exercise they will distract themselves with other thoughts and actions. Approaching the problem with the right attitude is a skill in its own right. It is important to develop critical thinking and creative skills, therefore as teachers "we need to create environments – in our schools, in our workplaces, and in our public offices – where every person is inspired to grow creatively" [Rob09].

We have seen similar denial and non-engagement with activities before, in particular when using the Five Design-Sheet method, where students cry "we don't think this is useful". However, this specific 'cry' is becoming very infrequent. However, we do hear the "but I cannot draw" complaint. To which we answer "yes, but we are asking you to sketch your ideas, not draw a masterpiece". However, even with this statement, anecdotally we believe that these complaints are continually reducing. Maybe it is because of the background, knowledge and experiences of today's students. Today, generation Z students are comfortable with computers; they are digital natives who enjoy retro concepts such as shown by the increase of vinyl record sales. And they still seem to be comfortable with pen and paper.

However, recently we have had more students wishing to perform the critical thinking sheet using computers. Analysing the results we discovered that three years ago we had one student complete their CTS using a tablet, two years ago we had two, and this last year we had four. While not an exact statistic, it is clear that we are seeing more students wishing to complete this task on their computer, or tablet. Certainly during the past few years technologies have become cheaper and faster, and many allow direct touch input. It is now much easier to sketch on a tablet or laptop PC, and pen interfaces are becoming more prominent. Especially with the rise of ePaper, eInk or digital-paper systems (such as reMarkable, Wacom Bamboo Folio or Sony DPT-RP1) it is becoming easier to digitally sketch in a way that feels similar to the experience a user

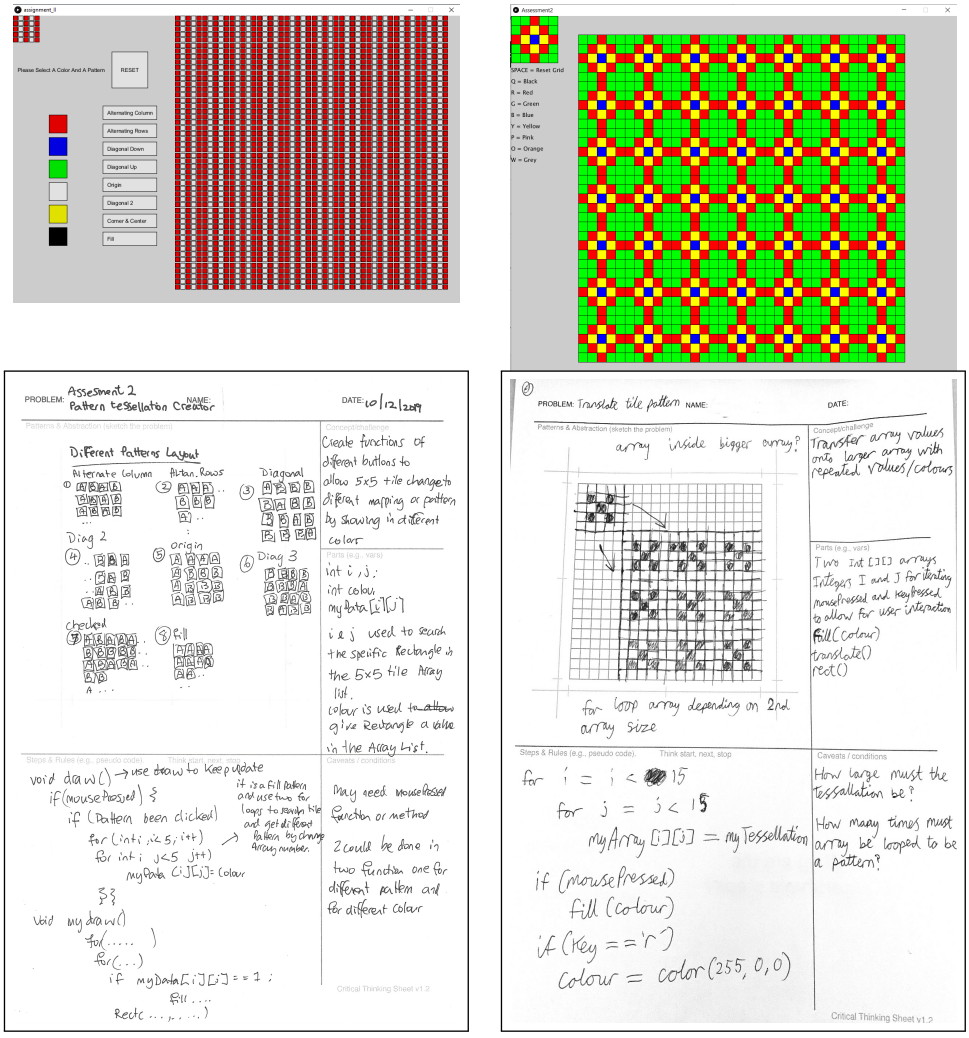

Figure 3: *The final tesselator tool and corresponding CTS sheets that have been created by two different students.*

would have with pen and paper. This is definitely a growth area, that we plan to utilise in more detail the future.

While there are some negative voices, there are many students who did get the idea. This is clear from the many positive comments we received in their critical reflective reports. They instantly understood the value of sketching and externalising their thoughts. That they could try out the ideas on paper much faster than they could code, and that they could check the algorithm with the tutor quicker on paper than debugging it on the IDE.

With the rise and need for workers to be competent in computational thinking, design thinking, visual computing and creative design, we believe there is a need to continue to develop techniques to help students learn and develop these skills. The CTS is one such tool. In addition, we believe it was useful to get the students to write a report and reflect on the use of the critical thinking process. Students need to reflect such to help them internalise and build upon their skills.

We have demonstrated how we have used the idea in our teaching, in our second year computer graphics module, for computing students. It is clear that students need to develop confidence in using new techniques, consequently we believe it was useful to have the initial labwork. We believe that it was important to keep the weighting of this coursework low in comparison to other activities in the course, such to encourage students to try out the method and to learn how to use it effectively. The single critical thinking sheet works well because it guides the users to consider the challenge from different perspectives. It helps learners organise their thoughts and direct their mind to create a set of smaller sub-tasks that will solve the problem.

References

[Anz91] ANZAI Y.: *Learning and use of representations for physics expertise*. Cambridge University Press, 1991, pp. 64–92. 1

[BRC*75] BRANSON R. K., RAYNER G. T., COX J. L., FURMAN J. P.,

KING F.: *Interservice procedures for instructional systems development. executive summary and model*. Tech. rep., DTIC Document, 1975. 2

[BRJ98] BOOCH G., RUMBAUGH J., JACOBSON I.: *The Unified Modeling Language User Guide*. Addison Wesley, MA, USA, 1998. 2

[Bux10] BUXTON B.: *Sketching user experiences: getting the design right and the right design*. Morgan Kaufmann, 2010. 2

[Cou07] COUNCIL D.: Eleven lessons: managing design in eleven global companies desk research report, 2007. www.designcouncil.org.uk. 2

[Cow07] COWLEY S.: *Getting the buggers to think*. Continuum, 2007. 4

[Dew10] DEWEY J.: *How we think*. D. C. Heath & Co., Boston, New York, Chicago, 1910. 3

[Dij82] DIJKSTRA E.: On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*. 1982, pp. 60–66. 2

[GAOS17] GAGNIER K. M., ATIT K., ORMAND C. J., SHIPLEY T. F.: Comprehending 3d diagrams: Sketching to support spatial reasoning. *Topics in Cognitive Science 9*, 4 (2017), 883–901. doi:10.1111/tops.12233. 2

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J. M.: *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 3

[HA06] HEER J., AGRAWALA M.: Software design patterns for information visualization. *IEEE Trans. Vis. Comput. Graphics 12*, 5 (Sept 2006), 853–860. doi:10.1109/TVCG.2006.178. 3

[HA17] HE S., ADAR E.: Vizitcards: A card-based toolkit for infovis design education. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan 2017), 561–570. doi:10.1109/TVCG.2016.2599338. 2

[HCT*14] HURON S., CARPENDALE S., THUDT A., TANG A., MAUERER M.: Constructive visualization. In *DIS 2014: Proc Designing Interactive Systems in 2014* (2014), ACM, pp. 433–442. doi:10.1145/2598784.2598806. 2

[Jon97] JONASSEN D. H.: Instructional design models for well-structured and ill-structured problem-solving learning outcomes. *Educational Technology Research and Development 45*, 1 (1997), 65–94. 2

[JR05] JOHNSON J. K., REYNOLDS S. J.: Concept sketches – using student- and instructor-generated, annotated sketches for learning, teaching, and assessment in geology courses. *Journal of Geoscience Education 53*, 1 (2005), 85–95. doi:10.5408/1089-9995-53.1.85. 2

[LÏ6] LÖWGREN J.: On the significance of making in interaction design research. *Interactions 23*, 3 (Apr. 2016), 26–33. doi:10.1145/2904376. 2

[LST08] LIM Y.-K., STOLTERMAN E., TENENBERG J.: The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Transactions on Computer Human Interaction 15*, 2 (July 2008), 7:1–7:27. doi:10.1145/1375761.1375762. 2

[MMAM14] MCKENNA S., MAZUR D., AGUTTER J., MEYER M.: Design activity framework for visualization design. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2191–2200. doi:10.1109/TVCG.2014.2346331. 2

[Mun09] MUNZNER T.: A nested process model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics 15* (Nov 2009), 921–928. doi:10.1109/TVCG.2009.111. 2

[NS12] NELSON H. G., STOLTERMAN E.: *The Design Way: Intentional Change in an Unpredictable World*. MIT Press, 2012. 2

[PÓ73] PÓLYA G.: *How to solve it: A new aspect of mathematical method*. Princeton university press, 1973. republished 2014. 3

[Pai75] PAIVIO A.: Perceptual comparisons through the mind's eye. *Memory & Cognition 3*, 6 (1975), 635–647. 1

[RHR16] ROBERTS J. C., HEADLEAND C., RITSOS P. D.: Sketching designs using the five design-sheet methodology. *IEEE Transactions on Visualization and Computer Graphics. 22*, 1 (Jan 2016), 419–428. doi:10.1109/TVCG.2015.2467271. 1, 2

[RHR17] ROBERTS J. C., HEADLEAND C. J., RITSOS P. D.: *Five Design-Sheets – Creative design and sketching in Computing and Visualization*. Springer, 2017. doi:10.1007/978-3-319-55627-7. 1, 2, 3

[Rob09] ROBINSON K.: *The Element – how finding your passion changes everything*. Viking Penguin, 2009. 5

[RRJH18] ROBERTS J. C., RITSOS P. D., JACKSON J. R., HEADLEAND C.: The explanatory visualization framework: An active learning framework for teaching creative computing using explanatory visualizations. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2018), 791–801. doi:10.1109/TVCG.2017.2745878. 1, 2

[San77] SANTA J. L.: Spatial transformations of words and pictures. *Journal of Experimental Psychology: Human learning and memory 3*, 4 (1977), 418. doi:10.1037/0278-7393.3.4.418. 1

[Sim73] SIMON H. A.: The structure of ill structured problems. *Artificial intelligence 4*, 3-4 (1973), 181–201. 2

[SMM12] SEDLMAIR M., MEYER M., MUNZNER T.: Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2431–2440. 2

[Win06] WING J. M.: Computational thinking. *Commun. ACM 49*, 3 (2006), 33–35. doi:10.1145/1118178.1118215. 1