Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science:

A First Look

December 2015

Korean version(v.2.0)

2017. 4. 21.

SRI Education™
A DIVISION OF SRI INTERNATIONAL

Authors

Marie Bienkowski, Eric Snow, Daisy Rutstein, Shuchi Grover **SRI International**

Acknowledgments

This work originated in a December 2011 workshop supported by an NSF planning grant (CNS-1132232). Attending the workshop was an interdisciplinary group of experts who gave us initial guidance on modeling the computational thinking domain for the purposes of our assessments. Meeting participants included experts in assessment and evidence-centered design: Irvin Katz, Educational Testing Service; Geneva Haertel and Britte Cheng, SRI International; and Gail Chapman, University of California at Los Angeles/Into the Loop. There were experts in computational thinking, computer science, and innovative ways of delivering content in school and afterschool settings: Maggie Johnson, Google; Alex Repenning, University of Colorado, Boulder; Chris Stephenson, Google (formerly at CSTA); and Melissa Koch, Anita Borg Institute (formerly at SRI). The Exploring Computer Science curriculum research and development project was represented by Joanna Goode, University of Oregon; Jane Margolis and Noelle Griffin, UCLA; Jenna Masser, CSTA; and Suyen Moncada-Machado, Los Angeles Unified School District. Experts in program evaluation, especially of information technology- and computer science-oriented curriculum, provided guidance on the use of assessments in research and evaluation: Girlie Delacruz, UCLA/CRESST; Jill Feldman, Westat; and Shaileen Pokress, Project Lead the Way.

After the planning grant and under a new NSF grant (CNS-1240625), in January 2013, early versions of the computational thinking practices patterns were reviewed by an expert panel consisting of Chapman, Goode, Katz, and Stephenson. They were also reviewed during a meeting in June 2013 by that grant's advisory board members, Dan Garcia, UC Berkeley; Jill Denner, ETR Associates; Drew Gitomer, Rutgers University; and Terry Vendlinski, TVATE (formerly at SRI).

We especially acknowledge Irvin Katz and Christian Schunn who conducted two rounds of review on the computational thinking practices design patterns (as well as those for collaboration and communication) in June 2015.

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1132232, CNS-1240625, and DRL-1418149. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or any collaborator or partner named herein.

Suggested Citation

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved from http://pact.sri.com/resources.html

Contents

번역 참여자(Translators)

김수환 / 총신대
강윤지 / 이태원초
김미영 / 이화여대 과학교육과
김성훈 / 청성초
김슬기 / 선부초
김인주 / 대전시교육청
김찬웅 / 부천원일초
김학인 / 한성과학고
박주연 / 이대부속초
송석리 / 한성과학고
서정보 / 청람초
안준별 / 광주광명초
유호진 / 삼보초
이상경 / 광주산정초
이왕렬 / 선린인터넷고
이영재 / 인천석전초
이인규 / 구정초
장재용 / 런투게더연구소 (L.T.Labs)
정효진 / 청원초
최용준 / 경기도과학교육원
천대건 / 송화초
최경희 / 맘잡고콘텐츠연구소

후 원(Acknowledgements)

한국컴퓨터교육학회 교육용프로그래밍연구회

CHONGSHIN MEDIA LAB

서 론

컴퓨터 과학은 비교적 최근에 형성된 학문분야이지만 무언가를 설계하고 탐구하는데 적용되고 있으며 사람들의 일상 속에까지 영향을 미치고 있다. 컴퓨터 과학은 파이프라이닝, 캐싱, CPU, 재귀와 같은 컴퓨터 소프트웨어, 하드웨어 작동에 관련된 새로운 용어와 컴퓨터 개념을 표현하는 새로운 언어를 포함하는 지식의 체계이다. 학생들이 오직 프로그래밍으로 만으로 전문성을 개발할 필요가 있는 지에 대한 의견 차이가 있더라도(Grover 2013; Resnick, 2013), 컴퓨터 과학 - 컴퓨팅 사고력을 형성하는 핵심 개념과 사고 방식에 노출되어야한다는 공감대는 계속 증가하고 있다(Gallup, 2015; Berdik, 2015; Shein, 2014; della Cava, 2015). 이러한 이유 중 하나는 이런 지식이 과학과 공학이 활용되는 방식을 바꿀 수 있기 때문이다(Nature, 2006). 예를 들어, 컴퓨팅 생물학자들은 자연선택과 같이 복잡한 생물학적 시스템 현상과 유전자 서열과 같은 방대한 데이터 문제를 모델링하기 위해 알고리즘을 사용한다. 컴퓨팅 사고력은 과학자가 시스템 모델링을 하고 대용량 데이터를 분석하도록 도울 뿐 아니라 컴퓨팅 상징을 사용해서 현상들을 설명하는 새로운 방식을 제공하고(Regav & Shapiro, 2005), 과학을 연구하도록 새로운 여러 연구 분야의 커뮤니티들을 만들어 갈 것이다(e.g., Tadmor & Tidor, 2005). 실제로, 컴퓨터는 "현대 생물학 연구에서 필수적인 요소가 되었고 과학자들은 컴퓨팅 생물학의 새로운 스킬에 적응하고 새로운 용어에 숙달해야 한다"(Loman & Watson, 2013, p. 996).

컴퓨터 과학 교육은 유초중등교육으로 확장하고 있고, 컴퓨팅 사고력을 배우는 것과 과학, 기술, 공학, 수학(STEM)을 배우는 것의 연관성을 밝히고 있는 연구도 있다 (Basawapatna et al., 2011; Basu et al., 2013; Grover, Pea, & Cooper (2015); Lewis & Shah, 2012; Wilensky & Reisman, 2006). 예를 들어, 센굽타와 동료는(2013) 물리학과 생물학 학습을 돕기 위해 에이전트 기반 모델링과 컴퓨팅 사고력을 사용했으며, 수학적 표현과 생태학에 관한 인과적 추론을 사용해서 학생들의 사고력의 발전을 발견했다. 컴퓨터 과학자들의 모델링 작업과 일치하는 바, 연구원들은 학생들이 과학 현상들의 모델 또는 시뮬레이션을 구축하기 위해 넷로고(NetLogo) 또는 에이전트시트(AgentSheets)와 같은 도구를 사용할 수 있는 방법을 연구했다(Basawapatna, Koh, Repenning, & Lewis, 2013; Wilensky & Reisman, 2006). 컴퓨팅 사고력을 STEM학습에 사용하는 프로젝트들은 유초중등 학생들의 성과 기대치를 달성하는데 요구되는 과학을 이해하는 방식으로 모델링 지원한다(NGSS, 2013).

컴퓨팅 사고력을 효과적으로 가르치기 위해 유초중등 교육자들은 학생들이 무엇을 알고 있으며 어떻게 배우고 있는 지를 이해해야 한다. 새로운 개념들과 프로그래밍 언어 명령어에 대한 학습자의 지식을 평가하는 것은 학습자들이 문제를 해결하고 창의적 컴퓨팅 산출물을 설계하기 위해 어떻게 컴퓨팅하고 프로그래밍 언어를 사용하는지 평가하는 것보다는 단순한 것이다.

학습자들이 자신의 지식을 컴퓨팅 사고력에 적용하는 방법을 평가한다는 의미는 문제해결과 코딩으로 만드는 해결책을 이해하고 산출물로 만드는 것에 대한 관계를 깊게 이해하고 있다는 증거를 찾는 것이다. 그렇다면, 학생들이 컴퓨팅을 활용하여 문제를 풀어가기 위해 어떻게 생각하고, 추상화를 적용하고, 다른 사람의 컴퓨팅 작업을 이해하고, 창의적 문제 해결 방법을 설계하고 구현하는 지를 가장 잘 평가할 수 있는가?

이 보고서는 컴퓨팅 사고력의 능력을 평가하는 유효한 증거를 제시할 수 있도록 평가 과제를 설계하는 원리 접근법에 대한 개관을 제공한다.
원리 접근법이란 학생들의 관찰가능한 행동으로부터 그들이 무엇을 알고 있는 지 평가할 수 있는 여러 가지 증거를 지정함으로 중요한 지식과 실행을 측정하는 평가 과제 설계를 의미한다. 평가에 대한 우리의 접근법은 디자인 패턴이라는 문서를 제공한다. 그 디자인 패턴은 어떻게 과제를 설계하는지와 과제 설계를 위한 템플릿, 관심있는 평가 영역에서 학생들의 능력에 대한 증거를 어떻게 이끌어 내는지에 대한 개요를 제공한다. 디자인 패턴은 특수한 학습 경험의 맥락에서 지식과 기능을 둘 다 평가하기 위한 과제들을 개발하도록 평가 전문가를 안내하고 다양한 과제들을 생성하도록 해준다.

이 보고서에 다루고 있는 디자인 패턴은 PACT (Principled Assessment Principled Assessment) 프로젝트에서 개발되었다. 이 프로젝트는 컴퓨팅 교육을 위해 누구나 접근할 수 있고 탐구 학습에 초점이 맞추어진 ECS(Exploring Computer Science)에서 제공하고 있는 특정한 고등학교 컴퓨터 과학 교육과정을 자세히 연구하였다.
현재 진행중인 PACT 프로젝트의 장기적인 목표는 다음과 같다(under NSF awards CNS-1132232, CNS-1240625, CNS-1433065, and DRL- 1418149):

- 지식과 기능의 평가 설계와 개발을 구성하기 위해 근본적인 지식과 기능, 평가 산출물-기준들의 매핑, 디자인 패턴, 과제-을 개발하여 컴퓨팅 사고력의 영역을 분석하고 모델링하기
- 온라인 평가에 중점을 두고 구현하는 것을 포함하여 평가 설계를 안내하기 위해 특별한 교육과정의 맥락에서 디자인 패턴의 예시를 개발함으로써 컴퓨팅 사고력의 평가요소를 개발하고 검증하기
- 평가와 다른 평가규준을 활용하여 고등학교 학생들의 컴퓨팅 사고력 학습에 영향을 주는 컴퓨터 과학 연계 교육과정과 ECS의 구현 요소들을 규명하기

이러한 목표를 달성하기 위해 아래와 같은 단기 목표를 실행한다:

- 커리큘럼을 개정할 때 평가를 설계하고 개선하기 위해 사용되는 주요 컴퓨팅 사고력 실행과 ECS 단원의 디자인 패턴 만들기
- ECS 맥락에서 컴퓨팅 사고력 실행을 위한 평가 과제 개발의 템플릿 개발하기
- 온라인 평가 공급을 포함하는 ECS 4가지 단원 평가(ECS 단원 1-4) 및 총괄 평가(ECS 단원 1-4)를 개발하고, 파일럿 테스트하고, 검증하기

본 보고서에서는 컴퓨팅 사고력을 증진하기 위한 다양한 커리큘럼을 관통하는 일반적인 디자인 패턴의 집합을 제시하는 것을 첫 번째 단기 목표로 한다. 이러한 디자인 패턴의 집합은 컴퓨터 과학 지식의 하위요소를 모델링한다: 이는 컴퓨팅 사고력의 실행, 혹은 컴퓨팅으로 문제를 해결하기 위한 탐구 및 설계의 적용, 컴퓨팅 산출물 만들기와 관련되어 있다. 이 디자인 패턴의 집합은 컴퓨팅 사고력 영역의 실행에 일반적으로 적용되며 컴퓨팅 문제를 해결하기 위한 설계와 탐구 기능(그런 기능의 적용이 필요한 기본적인 개념 지식 보다는)을 강조한다.

디자인 패턴의 개발은 컴퓨터 과학 교육 및 평가에서 다양한 전문가와 워킹 그룹 간의 상호작용을 통해 2011년 시작되었다. 동시에, ECS 및 새로운 대학교과선수이수제(AP) 컴퓨터 과학 원리를 개발하는 교육자 커뮤니티는 컴퓨팅 사고력 실행의 넓은 정의를 체계화했다(Arpaci-Dusseau et al., 2013). 컴퓨팅 사고력에 실행을 추가

한 것은 단지 학생들이 보여주는 개인 내부의 사고뿐만 아니라 수행방법의 방향을 반영하는 것이다. 이러한 방향은 과학 실행이 더 깊은 종류의 지식과 기능의 교육과정 또는 평가에 포함되어야 할 과학 실행 요구사항에 대한 중요한 의견을 반영하는 미국 차세대 표준과학교육과정(NGSS)과 일치한다(NGSS Lead States, 2013). 고등 컴퓨터 과학 교육에서 컴퓨팅 사고력 실행이 중요한 부분으로 다루어지고 있는 점과 실행을 평가하는 것이 어렵다는 이류로 인해 우리는 이러한 방향으로 개발작업을 조정했다. 원칙에 입각한 평가 설계를 만들었던 경험으로 우리는 결정적인 요구사항을 성공적으로 반영할 수 있는 입장에 있다.

본 보고서는 고등학교 수준에서 평가 개발을 가이드할 수 있는 네 가지 컴퓨팅 사고력 실행 디자인 패턴과 두 가지 지원 디자인 패턴(별첨 1)을 제시한다. 지원 구성체, 협업, 소통은 특정한 교과들을 관통하는 실행들을 제정하는 방법이다. 현재까지는, 이러한 여섯 가지 디자인 패턴이 평가 항목 개발에 사용되지 않았다. 각각의 컴퓨팅 사고력 디자인 패턴에 대해, 이 디자인 패턴의 유용성을 이해할 수 있도록, 어떻게 교육 프로젝트에 적용 가능한지 살펴볼 수 있는 프로젝트들이 있다: computational thinking in science in the Computational Thinking in STEM (CT-STEM) project (ct-stem.northwestern.edu), constructionist pedagogies using Scratch (scratch.mit. edu), game and simulation design using AgentSheets (www.agentsheets.com), and storytelling and game design using Alice (www.alice.org).

새로운 NSF 기금 프로젝트에서, SRI 연구재단은 정규 컴퓨터 과학 코스외에 다른 학습 상황과 과학, 수학의 평가에 대한 생각의 시작점으로서 이런 디자인 패턴을 사용하고 있다. 우리는 다른 컴퓨터 과학 교육 내용과 관련된 내용 영역에서의 평가 개발에 대한 이런 디자인 패턴의 적용가능성에 대해 컴퓨터 과학 교육과 평가 커뮤니티로부터의 조언을 환영한다.

본 보고서는 컴퓨팅 사고력 영역의 개관을 제공하며, 디자인 패턴을 개발하기 위한 방법론을 요약하고, 협력과 의사소통을 위한 디자인 패턴, 예시적인 적용사례가 나타난 컴퓨팅 사고력 실행의 디자인 패턴의 중요 부분을 보여준다.

별첨 1. 컴퓨팅 사고력 실행 및 지원 구성체

컴퓨팅 개발의 영향 분석
창의적 해결 및 산출물에 대한 설계 및 구현
추상화 및 모델에 대한 설계 및 적용
컴퓨팅 작업 및 다른 사람의 작업 분석
사고력 과정 및 결과에 대한 의사소통
컴퓨팅 활동에 대한 동료와의 협력

컴퓨팅 사고력

Wing(2006, 2011)은 컴퓨팅 사고력(Computational thinking)을 문제를 만들고(formulating) 그것을 컴퓨터에 의해서 효과적으로 실행될 수 있는 형태로 보여지는 해결책과 관련된 사고과정이라고 정의하였다. 현재 컴퓨팅 사고력은 알고리즘적 사고, 자동화, 분해, 모델링, 패턴, 재귀, 비례, 상징적 표현, 다른 위계 수준에서의 추상화와 같은 문제해결전략과 컴퓨터 과학 구조를 망라하는 개념으로서 인식되어 왔다(e.g., Cortina, 2007; Denning. 2009; Grover&Pea, 2013; Guzdial, 2008). 2010년에 NRC(the National Research Council)는 컴퓨팅 사고력의 범위와 본질을 연구하기 위한 연구팀을 모집했고, 그 결과 보고서(NRC, 2010)에서 컴퓨터적으로(computationally) 생각한다는 것은 컴퓨터 과학, 컴퓨터, 기술 소양(technology literacy), 프로그래밍과 완벽한 동의어가 아니고 수학적, 과학적, 양적인(quantitative) 생각과 다른 것이라고 강조했다. 이 작업 그룹은 컴퓨팅 사고력을 정의하지 못한 채 남겨두었지만, 후속 작업으로 교육과정 기준, 문헌의 종합, 인력 기술(workforce skills)을 검토함으로써 간결한 정의를 만들어갔다.

다양한 분야의 전문가를 위해 컴퓨팅 사고 기능은 협력, 창의성과 같은 21세기 기술의 보완일 뿐 아니라 확장이다. 2012년의 인적자원보고서(workforce study)에서 Malyn-Smith와 Lee는 컴퓨팅 사고력을 가진 전문가는 대게는 컴퓨터 사용을 통해서 문제를 해결하고, 산출물(products)을 설계하고, 시스템을 자동화하거나, 시스템, 절차(processes) 혹은 매커니즘(mechanisms)을 정의, 모델링, 평가(qualifying), 개선함으로써 이해를 향상시키는 창의적인 과정에 협력하고 참여할 수 있는 사람이라고 정의하였다(p.3).

부분적으로 컴퓨터 프로그래밍 학습은 컴퓨팅 사고력으로 전환하는데 영감을 준다. K-12에서의 컴퓨팅 사고력 실천의 현재 상태에 대한 최근의 비판적 연구를 통해 다음의 특징들을 종합하고 끌어냈다(Grover&Pea, 2013).

- 추상화와 패턴의 일반화(모델과 시뮬레이션을 포함한)
- 체계적인 정보 처리
- 상징 체계와 표현
- 제어의 흐름의 알고리즘적 개념(notions)
- 구조화된 문제 분해(모듈화)
- 반복적, 재귀적, 병렬적 사고
- 조건적 논리
- 효율성과 성능 제약조건
- 디버깅과 시스템 오류 발견

이러한 것들은 컴퓨팅 사고력의 포괄적인 관점을 형성하는데 중요한 특징들이다. K-12와 관련된 사람들에게 컴퓨팅 사고력에 대한 정의는 문제해결과 데이터 표현을 강조하는 대신에 프로그래밍을 경시하는 것처럼 보인다. 컴퓨터 과학과 정보 기술을 위한 전문적 기관인 ISTE와 CSTA는 컴퓨팅 사고력을 다음과 같이 정의하고 있다.

문제를 해결하는데 도움을 주는 다른 도구들과 컴퓨터를 사용하게 하는 방식으로 문제를 형성해 가는 것; 데이터를 논리적으로 조직화하고 분석하는 것; 모델과 시뮬레이션 같은 추상화를 통해서 데이터를 표현하는 것; 알고리즘적 사고(일련의 단계적 순서)를 통해서 해결책을 자동화하는 것; 가장 효율적이고 효과적으로 단계와 자원을 조합하기 위해 가능한 해결책들을 식별하고 분석하고 구현하는 것; 이 문제 해결과정을 다양한 문제들에 일반화하고 전이하는 것(Barr,

Harruson, & Conery, 2011).

또 다른 사람들은 이렇게 포괄적으로 시도하지 않고, 대신에 STEM과 다른 영역에서 산출물을 설계하는데 문제를 해결하는데 컴퓨터를 사용하는 그들의 방법과 일치하는 특징들을 강조하였다. 예를 들면, 교육자들을 위해 자원을 제공하는 기업 후원 웹사이트에서는 패턴, 추상화, 모델, 알고리즘, 데이터 분석과 시각화(컴퓨팅 사고력을 탐구하는)를 포함하는 요소를 강조하거나 또는 다른 과학에서 컴퓨팅 사고력의 예를 제공한다(Phillips, 2009; Microsoft Corporation, 2014). 게임과 시뮬레이션을 위한 몇몇 교육과정에서는 에이전트(agent)의 행동 및 확산, 충돌, 탐색, 투표와 같은 에이전트의 상호작용을 모델링(modeling)함에 있어 발생하는 현상을 캡쳐하는 컴퓨터적 패턴(computational patterns)을 강조한다("Scalable Game Design Wiki", 2014). 계산과학(Computational science)을 지향하는 프로젝트들은 데이터, 모델링, 시스템 사고를 강조하는 경향이 있었다(Computational Thinking in STEM: Lesson Plans, 2015). 또한 여전히 다른 사람들은 수학을 가르치기 위해서 컴퓨팅 사고력을 사용한다(Bootstrap Materials: Curriculum and Software, 2015).

대부분의 예나 정의들은 컴퓨팅 사고력을 그것이 사용되는 학문 분야의 맥락에서 지식의 적용과 통합을 요구한다. 이러한 구상은 CCSS(Common Core State Standards), NGA(National Governors Association, 2010)와 NGSS(NGSS Lead States, 2013)에 요약 서술된 기대와 일치하는 바, K-12학생들이 내용지식을 실제로 적용하기를 희망한다. Common Core Standards는 문제를 해결(수학적 실행 1: 문제를 이해하고 문제를 해결하는데 끈기를 갖는 것)과 추상화(수학적 실행 2: 추상적으로 그리고 정량적으로 추론하는 것)와 같은 수학에서 컴퓨팅 사고력의 실행을 언급한다(National Governors Association, 2010). NGSS는 컴퓨팅 사고력을 정의하지는 않았지만, 수학과 컴퓨팅 사고력을 사용하는 것은 모델링과 분석, 데이터 해석을 위한 중요한 실천이라고 기술하였다.

우리는 컴퓨팅 사고력을 위한 틀과 표준을 CC(Common Core)나 NGSS와 유사하게 정의할 수 있지만, 평가 개발을 지원하기 위해 설계와 탐구 실행으로써의 컴퓨팅 사고력에 대한 개념 정의와 교과 과정 표준이 필요하다(Bienkowski, 2015; Sykora, 2014). 다행히도 컴퓨팅 사고력과 관련된 수행 기준은 정보통신기술 유창성(information technology fluency) 표준을 대체하고 있고, 예비 대학생들이 알아야 하는 것을 재조정하고 있다. CSTA와 ISTE는 컴퓨터 과학 교육자와 전문적인 컴퓨팅 커뮤니티에 기초하여 K-12에서 지식의 3개의 수준으로 컴퓨터 과학 교육과정 표준을 마련하였고(CSTA, 2011), 이러한 컴퓨터 과학 표준은 특별히 컴퓨팅 사고력을 위한 요소를 포함시켰다. 현재 수정중인 2011 CSTA기준은 컴퓨터 과학을 지지하는 조직인 code.org가 주도하는 여러 관계자 팀(multistakeholder team)에 의해 공식화되어 K-12에서의 컴퓨터 과학을 위한 프레임 워크로 제공될 것이다.

이는 ECS와 AP 컴퓨터 과학 원리 과목과 같은 특정 교과를 위한 학습 목표 뿐 아니라 정의와 표준을 검토하는 컴퓨팅 사고력을 위한 평가 설계 도구 개발을 위해 중요한 출발점이다. 그러나 이러한 표준과 교육과정 학습 목표는 학습내용의 수준과 평가 항목과 과제의 설계를 위해 요구되는 자세한 측정 기준을 제공하지 않아서 평가를 개발하기에 충분하지 않다. 따라서 다음과 같이 보완적인 접근이 필요하다.

증거 기반 설계

컴퓨팅 사고력 실행에 대한 평가 개발을 위해, 우리는 타당하고 신뢰할 수 있는 측정이 이루어질 수 있도록 적합한(구체적인, 명확한) 수준의 실행에 대한 정의를 할 필요가 있다. SRI 교육팀은 증거 기반 디자인(ECD) 프레임 워크를 사용하여 평가하기 어려운 구조에 대한 평가를 개발하는 작업을 전문적으로 하고 있다. ECD는 측정하려는 지식과 기술이 컴퓨팅 사고력에서 요구되는 것과 같이 복잡하고 여러단계의 수행을 포함할 때 특히 유용하다. ECD는 구조를 중심으로 구성된 광범위한 학습 목표를 세분화하고 학생의 능력에 대한 증거를 이끌어 낼 수 있는 과제 및 상황을 설명하며 학생들의 능력에 관한 정보를 산출하기 위해 증거를 수집하는 방법을 설명하는 데 도움을 준다.[1] 이러한 평가가 어려운 영역에서의 우리의 경험은 ECD가 요구하는(필요로 하는) 선행 설계 작업을 통해 내용 유효성을 갖는 평가 항목들을 효율적으로 생성할 수 있게 해주었다.

ECD는 논증평가의 구축을 위해 명시적인 도구를 제공한다.(Mislevy & Riconscente, 2006; Mislevy, Steinberg, & Almond, 2003) Messick (1994)은 논증평가의 본질을 다음과 같이 요약했다.

> 구조 중심 접근은 지식, 기술 또는 다른 속성의 복잡성이 평가되어야 하는지를 묻는 것으로 시작한다. 이는 아마도 그들이 명시적 또는 암시적 교육 목표와 관련 있거나 혹은 사회에서 가치가 있기 때문일 것이다. 다음으로, 어떤 행동이나 수행이 그러한 구조들을 나타내고, 어떤 과업이나 상황이 그러한 행동을 이끌어내는가? 따라서 구조의 본질은 관련 과업의 선택 또는 구성은 물론 구조 기반의 채점 기준 및 루브릭의 합리적인 개발을 안내한다.(17 쪽)

ECD는 핵심 역량(Messick의 "지식, 기술 및 다른 속성의 복합체")에 대한 세부 사항과 측정 방법을 제공하는 디자인(설계) 문서(디자인(설계) 패턴 및 과업 템플릿)의 체계적인 생성을 통해 컴퓨팅 사고력과 같은 영역에 대한 보다 중요한 이해를 촉진함으로써 논증평가 구축에 있어 개발을 부분적으로 지원한다. 이러한 디자인 문서는 구조에 대한 자세한 설명에서부터 평가를 개발하고 제공하는 데 필요한 기술 사양(설명서)까지 작업의 모든 측면을 다룬다. 디자인 패턴은 평가 설계자가 과제을 생성하는 데 사용할 수 있는 방식으로 영역에 대한 근거 논증의 구조화된 서술적인 설명으로 나타낸다(Liu & Haertel, 2011).

ECD는 일반적으로 5가지 계층으로 설명된다.(Mislevy et al., 2002) 이 과정에서 생성된 주요 실체/아티팩트(사용흔적)의 이러한 계층과 예는 별첨 2와 같다. 이러한 계층은 순차적인 디자인 프로세스의 단계를 제시하지만 반복 및 세밀화 주기가 계층 내부 및 계층 간 모두에서 이루어지며 서로 다른 계층에서의 작업이 동시에 발생할 수도 있다. 이 보고서는 컴퓨팅 사고력 영역을 위한 영역 분석 및 모델링 계층에서의 작업을 제시하고 공식 및 비공식 교육 환경에서 컴퓨팅 사고력 실행 평가 작성을 위해 결과물을 어떻게 적용 할 수 있는지 제시한다. 영역 분석은 핵심 실체를 파악하고 정의하는데 도움이 되었으며, 영역 모델링을 통해 평가 디자인을 위한 요소를 정하는 디자인 패턴이 만들어졌다.

1) ECD에 관한 더 자세한 정보는 Mislevy and Riconscente(2005, 2006)와 Haertel et al.(press-b에서). ECD가 적용된 영역의 예는 다음을 찾아보시오. DeBarger and Snow (2010) (생명 과학); Mislevy, Riconscente 및 Rutstein (2009)(모델 기반 추론) Cheng, Ructtinger, Fujii 및 Mislevy (2010)(시스템 사고)

별첨 2. 컴퓨팅 사고력을 위한 증거 기반 디자인의 다섯 가지 계층(단계)과 핵심 실체

ECD 계층	역할	핵심 실체 및 예들
영역 분석	평가에 영향을 미치는 컴퓨팅 사고력 영역에 관한 실질적인 정보 수집. 지식이 어떻게 구성되고, 획득되고, 사용되고 전달되는지	컴퓨터 사고력 영역 개념 (예 : 추상화, 자동화); 전문 용어 (디버깅); 도구 (프로그래밍 언어); 표현 (스토리보드); 사용 상황 (포식자-피식자 모델링, 시각적 스토리텔링), 커리큘럼 표준 및 매핑
영역 모델링	영역 분석을 통한 정보를 기반으로 서술 형식의 논증평가 표현	평가할 지식, 기술 및 기타 속성에 대한 (자세한)설명서(예:주어진 데이터에서 프로그램을 실행 한 결과를 설명); 증거를 불러일으킬 수 있는 상황의 특징(프로그램에서 오류를 발견); 증거를 전달하는 실행의 종류 (재귀 사용)
개념적 평가 체계	과업 및 평가, 평가 절차, 측정 모델을 위한 구조 및 (자세한)설명서의 논증평가 표현	학생, 증거 및 과업 모델; 학생, 관찰 가능 및 과업 변수; 루브릭; 측정 모델; 테스트 어셈블리(조립) 설명서; 과업 템플릿 및 과업 설명서
평가 구현/실행	프레젠테이션 준비 작업 및 보정/조정된 측정 모델을 포함한 평가 구현/실행	업무, 업무 자료 (보조 자료, 도구, 자산 포함); 평가 절차를 잘 다듬고 측정 모델을 개선하기 위한 파일럿 테스트 데이터
평가 전달	학생과 과업간 상호 작용 조정 : 과업 및 시험 수준 채점; 보고	제시된 과업; 만들어낸 산출물(창의적으로 만들어낸 작업물); 평가 된 점수

출처 : Haertel et al. (in press-a).

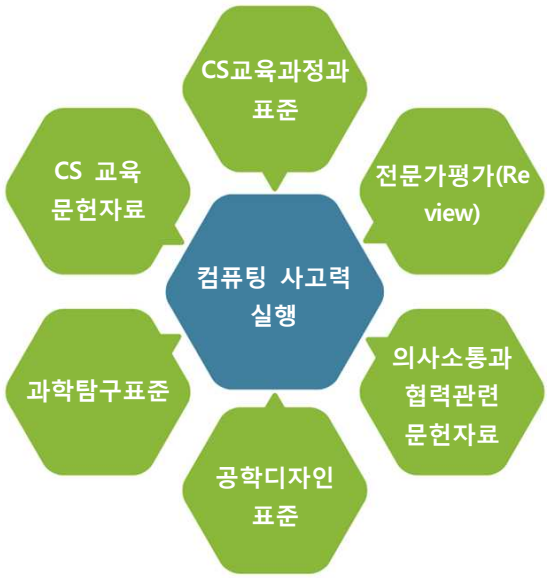
영역 분석

영역 분석의 주요 활동은 관심 주제에 대한 기존 자료의 검토이다.(별첨 3). 이 작업을 위한 자료에는 구조 정의; 표준; 교육과정; 컴퓨터 과학 교육, 과학 탐구, 공학 설계, 커뮤니케이션, 협업, 이전의 컴퓨터 과학 평가 프로젝트가 포함되어 있다. [such as Herman, Loui, & Zilles (2010), McCracken et al. (2001), and Tew & Guzdial (2010) for postsecondary and Werner, Denner, Bliesner, & Rex (2009), Nicholson, Good, & Howland (2009), Robertson & Howells (2008), and Brennan & Resnick (2012) for middle school)].[2)]

우리는 또한 경험 많은 컴퓨터 과학 교사의 의견을 듣고 전문가 및 고문으로부터 중요한 피드백을 받았다(감사의 말 참조). 우리는 CSTA 표준 (CSTA 2011), ECS 커리큘럼 (학습 목표 포함) 및 AP 컴퓨터 과학 원리 프레임 워크 (학습 목표 및 근거 진술 포함)에 특별히 주의를 기울였다. 특히, 4가지 컴퓨팅 사고력 실행과 2가지 지원 실행에 대한 ECS 및 AP 컴퓨터 과학 원리 채택은 디자인 패턴을 생성하는데 있어 구조의 기초 역할을 했다.

2) 2 원본 참고 문헌은 보고서의 끝에 있다.

별첨 3. 컴퓨팅 사고력 실행, 의사소통과 협력에 대한 영역 분석 소스



컴퓨터과학 교육과정의 실행 아이디어는 다른 프레임워크들과 일치한다. 예를 들면 NGSS 프레임워크(NRC, 2012)에서는 각각의 실행 기대치는 학문 핵심아이디어와 학문간 교차개념을 갖도록 과학 또는 공학의 실행을 적절하게 결합해야 한다고 공표하였다. 이것은 과학과 공학 실천의 활용 능력과 분리된 핵심아이디어에 대한 이해만을 측정하는 평가를 지양한다. 대신에 학생이 개념을 아는 것뿐만 아니라, 과학 탐구와 공학디자인 실천으로 의미 있는 문제를 풀기위해 학생들이 이해를 적용한 실행과 내용을 함께 평가하는 것을 지향한다. NGSS 프레임워크에서는 특별한 이유로 '과학 과정(science processes)' 또는 '탐구(inquiry)'기능보다 '실행(practice)'이라는 용어를 사용한다.

> 우리는 '기능(skills)'보다는 '실행(practice)'이라는 용어를 사용한다. 이는 기능뿐만 아니라 각각의 실천에 특화된 지식이 요구되는 과학적인 탐구(그리고 공학디자인)와 관련되어 있음을 강조하기 위해서이다.(NRC, 2012, p.30)

영역분석과 영역 모델링 과정을 통해 정제된 중요한 디자인 패턴의 요소는 개요 진술과 구조체의 요약에 나타나 있다. 핵심영역 구축에 관한 개요는 영역 모델링 과정의 디자인 패턴을 생성하는데 도움이 된다.

컴퓨팅 사고력 실행과 의사소통 및 협력에 대한 설명은 별첨 4에서 볼 수 있다. 또한 이전의 AP 컴퓨터과학 원리에 대한 초기 연구로부터 이러한 개요가 유사함을 보여준다. 이들 유사점은 컴퓨터과학 원리 프로그램에서 기본 정의한 실행과 컴퓨팅 사고 실행 정의가 어떻게 일치하는 지를 나타낸다.[3] 이들이 어느 정도 일치함에도 불구하고, 특정 교육과정보다는 컴퓨팅 사고력 실행을 따르는 교육프로그램의 평가개발 안내서에 제공하기 위한 디자인 패턴을 만들었다.

3)http://csprinciples.cs.washington.edu/sixpractices.html에서 서술한 컴퓨팅 사고력 실천의 오리지널 정의이다.

별첨 4. 컴퓨팅 사고력 실행에 관한 설명과 AP컴퓨터과학 원리의 유사점

구성	디자인 패턴 설명(개요)	CS 원리 유사점
컴퓨팅 개발의 효과 분석하기	이 디자인 패턴은 컴퓨터나 컴퓨팅에 적용한 문제들의 이해범위를 보이는 학생 과제를 평가 개발하는 것을 지원한다. 학생들은 컴퓨터와 컴퓨팅의 요소들을 인식하도록 요구될 것이다. 그들은 어떻게 컴퓨팅이 사회전체와 다양한 학문에서 혁신을 이루어 내게 되었는지, 동시에 어떻게 윤리적(예, 사생활(privacy) 침해), 사회적정의(동등한 접근(equal access))에 관한 이슈들을 야기하였는지 이해하게 될 것이다. 그들은 "지능화" 기계의 이해와 네트워크로 연결된 시스템에 대해 광범위하게 이해하게 될 것이다.	-컴퓨팅 기술로 인해 가능해진 기존 혹은 잠재적 혁신을 식별 -컴퓨팅 발전의 윤리적인 영향을 식별 -컴퓨팅 혁신에 사회에 미치는 (긍정적, 부정적)영향 인식 -디자인결정 시행의 영향분석 -컴퓨터적 산출물의 활용평가 -인간의 요구사항과 컴퓨터의 기능 간의 연결을 정의 -관련된 지적재산권 이슈의 설명
창의적 해법과 산출물을 디자인 하고 수행하기	디자인 패턴은 학생들로 하여금 참신한 아이디어와 문제 해법을 컴퓨터적 해법과 (로봇공학같은 컴퓨팅 혹은 컴퓨터프로그램으로 구현된 해법 같은) 산출물로 변환하도록 하는 과제개발을 지원한다. 학생은 주어진 목적이나 의도를 따라 디자인하고 수행한다.(이 경우 프로그램은 의사소통적인 행위로써 볼 수 있다.) 이 디자인 패턴은 문제해결과 창의적인 과정들의 이해, 분해, 탐험(즉 스토리보드에 문제를 발표하는 것과 다른 플로우차트, 창의성에 의해 유사코드)을 단계를 포함한다. 이러한 산출물은 하나 혹은 그 이상의 디자인 해법과 산출물을 생산한다.	-연관되고 흥미로 와서 학생들에 의해 선택된 산출물의 창조 -주어진 문제에서 해법 디자인 -문제를 풀기 위해 적절한 접근을 선택 -이미 정해진 알고리즘의 적절한 사용 -프로그래밍 구조체(programming constructs)와 자료 구조(data structures)의 적절한 사용 -다양한 표준을 사용한 산출물의 평가 -에러의 위치와 수정 -컴퓨터적 산출물 개발을 위한 적절한 기술의 사용
추상화와 모델을 디자인하고 적용하기	추상화를 위한 전략적 사고는 컴퓨팅 사고력의 특징이다. 이 디자인 패턴은 일상생활에서 특별한 것을 끄집어내는 아이디어와 표상을 사용하는 과제를 지원한다. 이것은 패턴, 실체 그 자체로, 과정 그 자체로, 또는 실체 또는 과정중의 관계/구조 같은 세부적인 수준을 포함한다. 여기에는 문제에 대한 일반적인 해결책을 설계하거나 보다 광범위한 문제(기능적 추상화)를 포괄하는 특정 솔루션을 일반화하는 것이 포함될 수 있다. 이러한 아이디어와 표현은 다른 맥락(문제 또는 분야)에서 사용될 수 있다. 학생들은 일상생활 속에서 아이템을 찾고, 수학, 모델, 다이어그램, 컴퓨터 프로그램(데이터 추상화), 자연 및 인공세계에서 발견되는 항목 등의 표현적 속성에 대한 지식을 나타낸다. 또한 학생들이 현상을 표현하기 위한 모델의 한계에 대한 이해와 모델 또는 추상화의 목적에 주목하는 것이 표현된다.	-어떻게 데이터, 정보 그리고 지식이 컴퓨터적 용도로 쓰이기 위해 어떻게 표현되는 지를 설명 -부과된/존재하는 질문을 조사하고, 새로운 질문을 개발하기 위해 시뮬레이션을 사용 -추상화의 적절한 수준에서 알고리즘 원리를 선택 -다른 수준의 추상화를 사용 -모델/시뮬레이션 디자인을 명확화 -데이터 추상화를 사용 -현상을 모형화하기에 적당한 데이터를 일반화하거나 수집 -경험적 예시에서 발생된 데이터와의 비교

별첨 4. 컴퓨팅 사고력 실행에 관한 설명과 AP컴퓨터과학 원리의 유사점(내용 이어짐)

구성	디자인 패턴 설명(개요)	CS 원리 유사점
컴퓨팅 작업 및 이외 작업들 분석	이 디자인 패턴은 학생들이 컴퓨팅 작업(프로그램, 프로그램 출력, 웹사이트 같은 산출물 또는 문제해결의 결과물)을 평가하고, 여러 컴퓨팅 산출물을 비교할 수 있음을 보여주는 작업 개발을 지원한다. 학생들은 문제해결 또는 컴퓨팅 목표성취에 어떻게 다른 기술들이 다른 방법들로 사용되는지 인식할 수 있다. 학생들은 산출물의 다음 항목들 - 만족도(요구사항을 충족하는지?), 정확성(올바르게 작동하는지?), 효율성(컴퓨팅 리소스를 현명하게 사용하는지?), 정밀함(모듈화, 절약성, 언어별 준수 및 언어별 코딩 관용구 준수, 단순성, 구조화 및 이해도에 대한 지침을 따르는지?)- 을 평가할 수 있다는 것을 보여준다. 그들은 컴퓨터가 취할 수 있는 형태(로봇공학을 포함하여)와 사용자 인터페이스가 사용성(사용자 대면 산출물에 대한 특수한 기준)에 어떻게 영향을 미치는지를 보여줄 것이다.	문제와 주어진 속성을 가진 산출물의 식별 문제해결에 유용한 툴을 비교하기 제안된 문제 해결법 및 해결법 사용에 대한 함축들을 평가하기 해결법의 절충점(trade-off) 분석하기: 가능한 해결법들을 적절한 정당성을 고려하여 가감하여 선택하기 프로그램의 결과 분석하기 문제의 특성 및 산출물 평가하기
사고과정 & 결과 소통하기	컴퓨팅 산출물에 대해 소통하는 것은 컴퓨팅 사고력의 여러 단계를 지원한다. 이 디자인 패턴은 학생들이 특정 대상에 적합한 방식으로 자신의 업무 프로세스와 결과를 전달할 수 있음을 보여주는 업무개발을 지원한다. 학생들은 컴퓨팅과 관련된 주요 주제와 아이디어를 그래프, 시각화, 컴퓨터 분석을 통해 서면 및 구두로 표현할 수 있다.	결과의 의미 설명 기술(technology) 또는 산출물의 영향을 기술하기(Description) 컴퓨팅 산출물의 동작을 요약하기 산출물과 기술의 설명 적절하고 올바르게 되었는지 정당화하기
컴퓨팅 활동에 대한 동료와의 협업	이 디자인 패턴은 학생들이 컴퓨터 과학의 협업 측면에서 공동으로 문제를 해결하고 컴퓨팅 산출물을 설계/창작함으로써 자신의 능력을 보여줄 수 있는 작업 개발을 지원한다. 학생들은 이해와 노력을 공유 할 수 있으며 페어 프로그래밍(pair programming) 또는 짝/팀 작업 같은 협업을 통해 동료 전문가 및 다른 사람들과 지식과 기술을 공유 할 수 있음을 보여준다.	효과적인 팀워크 실행의 적용 참여자들의 협업 여러 참가자들의 적극적인 기여에 의한 산출물 생산 산출물의 사용법, 기능 및 구현을 설명하는 문서화

영역 모델링 중에 지정된 추가 디자인 패턴 요소는 구성 범위를 설명하고 평가 개발자에게는 제안 및 지침을 제공한다. 디자인 패턴의 주요 요소가 다음에 설명된다.

영역 모델링

ECD의 영역 모델링은 측정을 위한 영역을 설명하기 위해 서술적 요소(narrative elements)를 만들어 내고, 서술은 디자인패턴으로 보존한다. ECD의 디자인패턴은 평가에서 측정해야 할 중요한 아이디어에 대한 사양을 제공하며, 다양한 형태의 평가를 생성하는데 도움이 되도록 사용, 재사용 및 정제 할 수 있다. 예를 들어 지필 테스트, 온라인 상호작용 테스트 또는 학생들이 제작한 컴퓨터 산출물을 점수화하기 위한 루브릭(rubric)을 만들기 위해 하나의 방식을 사용할 수 있다. 디자인 패턴(Design Patterns)은 전통적인 지필방식을 통한 학습 측정은 물론이고, 컴퓨터 기반 심지어 게임 플레이에서의 전략, 전술 및 움직임 관찰이나 시뮬레이션 기반 및 개인 학습 환경에서의 학생들의 움직임을 통한 평가까지도 안내하기에 충분하다.

디자인 패턴(Design Pattern) 요소

디자인 패턴(Design Pattern)은 측정되는 지식과 기술, 해당영역에서 지식과 기술의 증거로 사용될 수 있는 관찰이나 행동, 그리고 원하는 관찰이나 행동을 유도하는 작업이나 활동으로 구성된다. 별첨 5는 우리가 사용하는 모든 디자인 패턴(Design Patterns)에 나타나는 요소에 대한 일반적인 정의를 제공한다.

별첨5. 디자인 패턴(Design Pattern)의 요소들

주요 지식, 기술 및 기타 속성 (FKSAs)	디자인 패턴(Design Pattern)의 타겟이 되는 기본 KSA와 우리가 추론하고자하는 것. 우리의 컴퓨팅 사고력 연습에 대한 초기 연구에서는 지식보다는 기술에 중점을 두었다.
추가 KSAs	평가 업무에 성공적 수행을 위해서는 필수적일수 있지만, 평가하려고하는 목표 기술은 아닌 다른 KSA. 컴퓨터 과학의 경우 수학 또는 프로그래밍 언어와 도구에 대한 지식이 포함될 수 있다. 추가 KSA는 기술 간의 상호 의존성을 보여주기 위해 디자인 패턴(Design Pattern)을 연결하는데 사용될 수 있다.
가능한 관찰	학생들의 말이나 행동 또는 행동특징은 학생의 성과에 대한 추론 근거를 구성한다. 가능한 관찰은 정확도, 도달정도, 완전성 및 정밀도와 같은 질적 요소를 사용하여 설명된다.
가능한 작업 산출물	몇 가지 가능한 산출물 또는 볼 수 있는 관측보고 작업 산출물은 평가진행 중에 채점된다.
과제의 특징	기대하는(관찰가능한) 증거가 드러나도록, 과제를 지원하는데 필요한 평가 상황에 대한 것
가변적 기능	난이도나 중요사항을 바꾸기 위해 변하게 할 수 있는 평가 상황에 대한 것

주요 지식, 기술 및 속성; Focal Knowledge, Skills and Attributes (FKSA)

영역 분석을 통해 개요를 얻은 후, 우리는 구조의 기본 지식, 기술 및 속성을 모두 상세화 한다. FKSA에서의 focal이 의미하는 바는 우리가 평가하고자 하는 학생과 관련된 지식, 기술 및 기타 속성의 중심 또는 핵심을 의미한다. FKSAs는 관심구조 안에서 주요 아이디어를 다루어야 한다. "창의적 솔루션 및 산출물 설계 및 구현"을 위한 FKSAs의 전체 세트에 대해 다음 섹션에서 제시할 것이다. FKSA에서 사용 된 세부적 사항을 설명하기 위해, 우리는 뒤이어 별첨 6에서 5가지를

보여줄 것이다. 숙련에 초점을 맞추기 위해 FKSAs를 ".. 하는 능력"으로 의도적으로 언급했음을 유의해야 하며, 이것이 기술에 대한 언급으로 가장 좋은 표현이라 생각한다. 이는 지식 측면에 초점을 맞춘 FKSAs와는 대조적이다. 그러나 영역의 완전한 모델링(modeling)을 위해서는 둘 다 필요하며, 우리는 실행의 기초가 되는 컴퓨터 과학 개념적 지식에 초점을 두고 지식 중심 FKSAs를 포함하도록 디자인 패턴(design patterns)을 확장해서 생각한다.

우리가 FKSAs에서 설명하는 능력은 학생들이 배워야하는 실행들이다.

우리는 이러한 능력 (학생들이 할 수 있어야하는 것을 설명하는 동사)을 우리가 측정 할 수 있는 요소로 한정하려 한다. 예를 들어, 이름/식별, 설명, 디자인 / 개발 / 생성, 설명, 정당화, 표현, 제시, 게시, 요약, 토론, 유용한 피드백 제공, 듣기와 같은 용어를 사용했지만 통합, 인식, 이해, 결론 등과 같이 애매한 용어는 사용하지 않았다.

별첨 6. 창의적 해결책과 산출물의 설계와 구현을 위한 중심역량(FKSA) 예제

1. 제시된 문제에서 입력과 출력을 구분하여 설명할 수 있는 능력
2. 제시된 문제를 여러 하위 문제로 분해하고 해결하는 구체적인 방법까지 포함하는 능력은 문제 전반에 대한 해결책이 될 것이다.
3. 목적 또는 의도에 따라 컴퓨팅 산출물(computational artifact)을 생성 할 수 있는 능력
4. 컴퓨팅 작업(computational artifact)을 발전시키기 위한 적절한 기술을 선택할 수 있는 능력
5. 런타임 오류를 식별하는 능력
6. 컴퓨팅 활동에서 동료와의 협업

우리는 이러한 능력에 계층 구조를 두지는 않았지만, (우리가 학습의 진행 과정을 설명하지 않았으므로) 학생들이 다른 동사를 사용하여 지식을 적용 할 수 있는 정도를 구분하였다. 예를 들어, 일부 기술의 경우 우리는 현상을 단순묘사 할 수 있는 것으로 충분하다고 생각했지만 다른 내용의 경우 학생들이 설명 할 수 있기를 원했다. 이를 위해 학생들의 설명은 관계를 그려 해석하고 비교 분석하는 것을 포함한다. 우리는 평가를 위한 FKSAs를 설명하여 학생들이 더 높은 수준의 학습에 참여하도록 유도했다. 평가는 설명, 정당화 및 비교를 포함한다.

가능한 관찰(Potential Observations)과 작업 산출물(Work Products)

구조체의 근간을 이루는 FKSAs를 명시한 후에 우리는 학생이 수행하거나 관찰 할 수 있는 것, 그리고 그러한 행동이나 산출물이 어떻게 FKSAs의 증거를 제공하는지를 명시해야한다. 가능한 관찰 및 관련 작업 산출물은 ""창의적 해결책과 산출물의 설계와 구현"구조 중 FKSA를 위한 별첨 7에 나와 있다. 이러한 "잠재성(potential)"을 통해 연구와 실습을 기반으로 구조 지식(knowledge of the constructs)이 진화함에 따라 이러한 요소가 발전 할 수 있었고 또한 발전해야한다는 아이디어를 나타내고자 한다.

우리가 학생들의 말과 행동으로부터 관찰하는 것은 단지 작업 산출물(work product)뿐 아니라 산출물과 그 특성(characteristics)이다. 예를 들어, 우리는 작업 산출물의 정확성과 적합성으로 설명되는 품질(quality)을 추구한다(별첨 7). 또한 정확한 답을 찾을 수는 있지만 정답이 여러 개인 경우에 주의해야한다. 이러한 경우 학생의 프로세스와 그 프로세스를 사용한 이유를 확인해야 한다. 그리고 작업 산출물이 주어진 특성을 나타내는 적합성의 정도 및 범위를 확인한다. 가능한 관찰은 이러한 특성을 나열하고 루브릭(rubrics)이 개발 될 때 더 구체적으로 기술된다.

별첨 7. 가능한 관찰과 산출물 for the Construct "Design and Implement Creative Solutions and Artifacts"

"창의적인 해결책 및 산출물 설계 및 구현"에서 FKSA에 대한 잠재적 관찰 예
1. FKSA : 문제의 입력과 출력을 식별하기 위해 문제를 기술 할 수 있는 능력
1.a. 문제 설명의 정확성
1.b. 식별 된 입력과 출력의 적정성
"창의적인 해결책 및 산출물 설계 및 구현"에서 FKSA의 가능한 작업 산출물 예
1. FKSA : 문제의 입력과 출력을 식별하기 위해 문제를 기술 할 수 있는 능력
1.a. 문제의 진술
1.b. 문제의 입력 및 출력 목록

특징적이고 가변적인 특징

우리가 학생들의 행동과 산출에 대해 관찰 가능한 것을 알고 그들을 관측하는 작업의 중요한 특성을 생각하는 것은 자연스러운 것이다. 또한 언어 나 문화 또는 다른 문제로 인한 장벽을 제거하기 위해 작업을 다양화할 수 있는 방법을 생각하는 것이 당연하다. ECD 디자인 패턴은 이러한 특징적이고 가변적인 특성에 초점을 둔다.

과제의 특징(Characteristic features)은 디자인 패턴에서 개발된 모든 작업이 통합되어야 하며 관심 구조를 측정하는 모든 과제에 있어야 한다는 특성이다.
가변적 특성(Variable features)은 다양 할 수 있고 관심 구조를 측정하는 특정 과제에 존재하거나 존재하지 않을 수 있는 특성이다.
특성의 차이점은 작업의 측정 목표에 따라 달라지며 항목의 난이도를 변경하거나 추가적인 KSAs를 측정하거나 지원할 수 있다. 사전에 가변적인 특성 지정하면 항목을 개발할 때 결정해야하는 사항을 강조 표시하는 데 도움이 된다. 과제의 특성은 작업의 필수적으로 요구되는 특성을 말하며 작업이 FKSAs의 증거를 이끌어 내기 위해 필요하다.

디자인 패턴 개발 및 평가

영역 모델링(domain modeling)에서 우리의 컴퓨터 사고 실습 디자인 패턴은 컴퓨터 과학 교육, 정보 기술 문해력 평가 디자인, 조사 평가 및 커리큘럼 개발에 대한 외부 전문가 및 고문의 비평을 포함하는 여러 방면의 검토를 거쳤다.

모든 검토자는 커리큘럼과 표준 자료를 제공 받았다.
각 디자인 패턴에 대해 검토자는 다음과 같은 질문을 받았다.

- 컴퓨팅 사고력 실행 방식의 전반적인 설명이 정확하고 이해가능 한가요?
- FKSAs를 대표하는가?
- 가능한 관찰 (우리가 증거로 충분히 가늠 가능한)이 맞는가?
- 가능한 작업 제품 (우리가 학생들이 할 수 있다고 기대하는 것)이 적절한가?
- FKSAs는 가능한 작업 산출물 및 가능한 관찰과 잘 일치하는가?

검토이후 FKSAs 및 기타 디자인 패턴 요소를 더욱 세분화했다.

컴퓨팅 사고력 실행(practices) 디자인 패턴과 예들

다음 4개의 하위 섹션은 각각 4개의 컴퓨팅 사고력 핵심 실행에 대한 디자인 패턴 요소를 제시한다.

1. 컴퓨팅을 활용한 개발의 효과 분석하기
2. computational 작업과 다른 사람들의 작업을 분석하기
3. 추상화, 모델을 설계하고 적용하기
4. 창의적인 해결방법과 산출물을 디자인하고 구현하기

우리는 위의 각 디자인 패턴에 대해 중심역량(FKSAs)에서 정한 분류에 따라 입문 수준의 컴퓨터과학 학습 경험의 예를 제시한다. 이 분류(alignment)는 각 패턴의 폭넓은 적용 가능성을 보여주며 각각의 맥락에 따라 어떤 것을 측정할 수 있는지 제안한다. 그런 다음 의사소통과 협업을 위한 두 가지 디자인 패턴에 대해 추가로 제시한다.

컴퓨팅을 활용한 개발의 효과 분석하기

개요

이 디자인 패턴은 학생들이 컴퓨터 및 컴퓨팅을 활용하여 해결할 수 있는 문제의 범위를 이해하고 있는지 보여주는 작업의 개발을 지원한다. 학생들은 컴퓨터와 컴퓨팅에 대한 다양한 측면들을 인식하도록 요청 받을 것이다. 학생들은 컴퓨팅이 다양한 분야와 사회 전반에 어떻게 혁신을 가져왔는지, 그리고 그와 동시에 사생활 침해 문제와 같은 윤리적 이슈와 기회 평등과 같은 사회 정의에 대한 이슈를 발생시켰는지에 대한 이해를 보여줄 것이다. 그들은 “지능화” 기계와 네트워크로 연결된 시스템에 대해 광범위하게 이해하게 될 것이다.

중심역량(지식, 기술, 속성)

컴퓨팅과 컴퓨터에 대한 일반적 지식

1. 컴퓨터가 프로그램을 실행하는 장치임을 인식하는 능력
2. 컴퓨터가 일을 수행하는 방법과 사람이 일을 수행하는 방법의 차이를 설명할 수 있는 능력. (컴퓨터는 명령을 따르며 그 명령은 다양한 해석을 할 수 없는 명확한 것이어야 한다는 점을 포함)
3. 컴퓨팅적 산출물의 특징을 설명할 수 있는 능력
4. 컴퓨터 프로세서와 같이 컴퓨터의 동작 특성을 가능케하는 컴퓨터의 구성 요소들을 설명할 수 있는 능력(무엇이 컴퓨터를 컴퓨터로 만드는가와 연관됨)

컴퓨터와 컴퓨팅의 사용

5. 컴퓨팅의 혁신으로 인한 컴퓨팅(연산) 능력, 저장 용량과 네트워킹 기능 및 형태 요인(예: 랩탑, 스마트폰, 태블릿)의 주요 변화에 대해 설명할 수 있는 능력(하드웨어와 소프트웨어의 변화를 모두 포함)
6. 컴퓨팅의 혁신으로 인해 컴퓨터 사용이 어떻게 변하였는지 설명할 수 있는 능력(하드웨어와 소프트웨어의 혁신을 모두 포함)
7. 컴퓨팅이 어떻게 혁신을 가능케하는지 설명하는 능력. 즉, 관련 영역의 발전을 가져오는 새로운 제품, 프로세스, 또는 서비스를 디자인 및 창작하거나 새로운 방식으로 무언가를 실행하거나 이루어내는데 컴퓨팅이 어떻게 기여하는지에 대해 설명하는 능력.
   7.a. 컴퓨터 프로그램을 사용해 정보를 처리하여 통찰력과 지식을 얻는 방법(인간이 처리할 수 없는 많은 양의 데이터를 컴퓨팅 도구를 이용하여 처리하고 이해하는 것을 포함)의 예를 설명할 수 있는 능력.
   7.b. 컴퓨터 프로그램이 새로운 제품, 프로세스, 서비스 개발을 통해 다양한 분야의 발전을 어떻게 가져왔는지에 대한 예를 설명할 수 있는 능력.
   7.c. 컴퓨팅의 발전이 새로운 제품, 서비스 또는 서비스의 보급에 어떻게 도움을 주었는지에 대한 예를 설명할 수 있는 능력.
   7.d. 컴퓨팅의 발전이 새로운 제품, 프로세스, 서비스를 개발하기 위한 의사소통의 증가를 어떻게 가능케 했는가에 대한 사례를 설명할 수 있는 능력.

인터넷과 인터넷에 기반한 시스템에 대한 지식

8. 인터넷과 인터넷에 구축된 시스템의 다양한 용도에 대해 설명할 수 있는 능력.
9. 계층 구조와 중복성이 인터넷 구현에 있어서 왜 중요한지에 대해 설명할 수 있는 능력.
10. 인터넷과 인터넷에 구축된 시스템 사용의 확산을 가능하게 한 인터페이스와 프로토콜에 대해 설명할 수 있는 능력.
11. 네트워크로 연결된 시스템에서의 작업이 프라이버시(개인정보, 행동, 위치 등)에 어떤 영향을 미치는지에 대해 설명할 수 있는 능력

일상생활에서의 컴퓨팅

12. 컴퓨팅이나 컴퓨터적인 솔루션이 (컴퓨터과학이나 정보기술 분야가 아닌)다른 분야에 적용되고 있음을 인식할 수 있는 능력
13. 컴퓨터와 컴퓨팅 보조 도구가 다양한 진로와 직업(컴퓨터과학이나 정보기술 분야가 아닌 의학, 영화, 정치, 예술 등)에서 어떻게 사용되는지 설명할 수 있는 능력
14. 창의성과 자기표현을 위해 컴퓨팅을 활용할 수 있음을 인식하는 능력.
15. 자동화된 데이터 분석을 통해 복잡한 자연과 인간 시스템에 대한 이해를 높일 수 있음을 인식하는 능력.
16. 컴퓨팅이 어떻게 의사소통을 향상시키고 의사소통과 협업의 새로운 방법을 만들어내는지에 대해 설명할 수 있는 능력
17. 다양한 형태의 컴퓨팅(사이버-물리 시스템이나 보조 기술)이 인간의 능력을 어떻게 향상시키는지 평가할 수 있는 능력.
18. 일상의 문제나 행동에 컴퓨팅 사고력을 적용할 수 있는 능력.
19. (추상화된)컴퓨터적 구조들을 적용해 주변의 사물이나 모델의 기능적 특성을 설명할 수 있음을 인식하는 능력.

컴퓨팅과 사회

20. 경제적, 사회적, 문화적 맥락에서 사회에 미치는 컴퓨팅의 다음과 같은 영향을 분석하는 능력 :
 20.a 컴퓨팅 자원과 컴퓨팅 분야의 맥락에서 하위집단(subgroups)에 대한 형평성 및 접근성 문제를 설명하는 능력
 20.b 인터넷과 웹이 사회 및 하위집단(subgroups)에 미친 긍정적이고 부정적인 영향을 평가하는 능력
 20.c 컴퓨팅 기반 혁신으로 제기된 법률 및 윤리적 문제를 평가하는 능력
21. 컴퓨팅 산출물이 개발되는 문화적 맥락(context)와 청중(audience)이 설계와 사용에 미치는 영향을 설명할 수 있는 능력

추가적인 중심역량(지식, 기술, 속성)

- 컴퓨터나 컴퓨터 분야의 전문용어와 특정분류에 대한 지식
- 특정한 컴퓨터 혁신에 대한 지식
- 직업 지식을 포함한 다른 영역/학문에 대한 지식
- 사회적, 문화적, 윤리적, 법적 요인에 대한 지식
- 네트워크와 정보교환에 대한 지식

과제의 특징

- 모든 작업은 컴퓨터의 사용이나 컴퓨팅 작업이 포함되어야 한다.
- 모든 작업은 사람이 컴퓨터나 컴퓨팅 작업과 상호작용하는 것과 관련 있어야 한다.
- 모든 작업은 네트워크로 연결된 시트템이어야 한다.
- 모든 작업은 실제 시나리오와 관련 있어야 한다.
- 모든 작업은 경제적, 사회적, 문화적인 것과 같은 사회적이거나 개인적인 특징이 포함되어야 한다.
- 모든 작업은 프라이버시, 법적, 윤리적인 특성에 영향을 주어야 한다.

가변적 특성

- 컴퓨터의 내부 작업에 대한 지식이 필요한 정도
- 컴퓨터나 컴퓨팅 작업의 설명, 비교, 평가가 필요한지 여부
- 제시된 상황(일상생활/직업생활)의 유형과 제시된 학문과 상황의 친숙함
- 제시되고 있는 문제의 유형 : 법적, 윤리적, 프라이버시 또는 기타
- 문제에서 제시되는 청중이나 문화
- 문제가 장점이나 단점(긍정적인 표현이나 부정적인 표현)을 요구하는지 여부

가능한 관찰

- 사용 설명의 타당성과(이나) 완성도
- 상황 사용에 대한 서술/설명의 타당성과(이나) 완성도
- 혁신에 대한 설명의 타당성과(이나) 완성도
- 이점과 문제의 타당성과(이나) 완성도
- 디자인의 효과에 대한 설명의 타당성과(이나) 완성도

가능한 작업 산출물

- 일상생활에서 컴퓨팅이나 컴퓨팅 해결책을 인식한다.
- 컴퓨터와 해당 구성 요소의 사용에 대한 서술
- 다양한 상황에서 컴퓨터와 컴퓨팅이 사용될 수 있는 방법에 대한 서술/설명
- 컴퓨팅 혁신(예를 들면, 네트워킹과 인터넷)에 대한 설명
- 예시를 들어 컴퓨팅 혁신의 장점이나 문제에 대해 서술 또는 설명
- 상황이 컴퓨팅 혁신의 설계에 미치는 영향에 대해 서술

컴퓨팅 개발의 효과 분석

예시 어플리케이션 : Computational Thinking in STEM

컴퓨팅 사고력은 과학과 수학 교실, 특히 게임, 모델, 시뮬레이션을 통하여 과학과 수학 교실에 적용되고 있다. Northwestern University의 컴퓨팅 사고력 기반 과학기술교육 프로젝트는 (CT-STEM) 고등학교 학생들을 위한 기존의 STEM 교육 과정의 일환으로 제공되도록 고안된 60개의 컴퓨팅 사고 활동을 개발하고 있다. 이러한 융합된 활동은 차세대 과학 표준(NGSS)에서 컴퓨팅 사고력 및 모델링 요구 사항을 다루며, (현재 시점에서는) 물리학, 화학, 수학, 생물학 수업을 통해 광범위한 학생들에게 영향을 주려고 한다. 사용 된 도구에는 에이전트 기반 시뮬레이션 도구 NetLogo, PhET 시뮬레이션, Desmos 수학 도구 등이 포함된다.

본 프로젝트는 다음과 같은 4가지 분야의 실무 영역을 정의한다: 데이터 수집으로부터 데이터 분석 및 시각화, 모델링 및 시뮬레이션(모델을 사용, 평가 및 설계), 컴퓨팅 문제 해결(아래 참조), 시스템 사고.

증거 기반 설계는 모델 기반 추론 및 시스템 사고를 분석하기 위해 적용되어왔으며, 이로부터 CT-STEM 실행에 대한 평가 개발을 이끌어 낼 수 있다. CT-STEM 컴퓨팅 문제 해결을 위해, 컴퓨팅의 발달의 효과를 분석하는 디자인 패턴의 적용이 가능해야 한다. CT-STEM의 컴퓨팅 문제 해결 방법은 다음과 같다.

- 컴퓨팅 솔루션에 대한 문제 해결
- 프로그래밍
- 효과적인 컴퓨팅 툴 선택
- 문제에 대한 다양한 접근법/해결책 평가
- 모듈식 컴퓨팅 솔루션 개발
- 컴퓨팅 추상화 생성
- 문제 해결 및 디버깅

CT-STEM 연구는 컴퓨팅에서의 개발 효과를 분석하는 디자인 패턴에서 중심역량(지식, 기술, 속성)(FKSAs)을 보여준다. CT-STEM 활동을 완료한 학생은 다음과 같은 디자인 패턴에서 중심역량(지식, 기술, 속성)(FKSAs)을 보여줄 수 있다.

7.a. 컴퓨터 프로그램을 사용해 정보를 처리하여 통찰력과 지식을 얻는 방법(인간이 처리할 수 없는 많은 양의 데이터를 컴퓨팅 도구를 이용하여 처리하고 이해하는 것을 포함)의 예를 설명할 수 있는 능력

12. 컴퓨팅이나 컴퓨터적인 솔루션이 (컴퓨터과학 기술 또는 정보 기술분야가 아닌) 다른 분야에 적용되고 있음을 인식할 수 있는 능력

18. 일상생활의 문제나 행동에 대해 컴퓨팅 사고력을 적용하는 능력

다른 디자인 패턴의 FKSAs도 이 연구에 적용된다. 예를 들어, CT-STEM 문제 해결 평가에서는 학생들이 몇 가지 선택 사항 중에서 출력을 선택하도록 코드를 해독한다. 이것은 그들의 컴퓨팅 작업과 다른 사람들의 작업을 분석하고 창의적인 해결책과 산출물을 설계하고 구현하는 디자인 패턴과 관련이 있다. CT-STEM 활동은 이러한 중심역량(지식, 기술, 속성)(FKSAs)들과 연관 될 수 있으며, CT-STEM 실행을 측정하기 위한 평가를 개발하기 위해 ECD 프로세스를 사용할 수 있다.

컴퓨팅 작업과 다른 사람들의 작업 분석

개요

이 디자인 패턴은 학생들이 컴퓨팅 작업(프로그램, 프로그램 출력, 웹사이트 같은 산출물 또는 문제 해결의 결과물)을 평가하고, 여러 컴퓨팅 산출물을 비교할 수 있음을 보여주는 작업 개발을 지원한다. 학생들은 문제해결 또는 컴퓨팅 목표성취에 어떻게 다른 기술들이 다른 방법들로 사용되는지 인식할 수 있다. 학생들은 산출물의 다음 항목들 - 만족도(요구사항을 충족하는지?), 정확성(올바르게 작동하는지?), 효율성(컴퓨팅 리소스를 현명하게 사용하는지?), 정밀함(모듈화, 절약성, 언어별 준수 및 언어별 코딩 관용구 준수, 단순성, 구조화 및 이해도에 대한 지침을 따르는지?)- 을 평가할 수 있다는 것을 보여준다. 그들은 컴퓨터가 취할 수 있는 형태(로봇공학을 포함하여)와 사용자 인터페이스가 사용성(사용자 대면 산출물에 대한 특수한 기준)에 어떻게 영향을 미치는지를 보여줄 것이다.

중심 역량(지식, 기술, 속성)

컴퓨팅 작업 평가를 위한 기준 파악 및 생성

1. 컴퓨터 산출물(프로그램, 웹사이트 또는 문제 해결법)과 구현 요구 사항(비용 및 배송 플랫폼 등)의 목표 및 결과를 설명 할 수 있는 능력
2. 다양한 관점(일반적인 목표 혹은 원하는 결과, 구현 방법, 의도된 사용자, 사용 맥락 등)으로 컴퓨팅 산출물(프로그램, 웹 사이트 또는 문제 해결법 등)에 대한 여러 평가 기준을 개발할 수 있는 능력
3. 형식과 설계가 인간 - 컴퓨터 상호 작용에 있어서 가용성에 어떻게 영향을 미치는지 설명할 수 있는 능력.

기준을 고려하여 컴퓨팅(전체적/블랙 박스 또는 분석적으로) 작업 평가하기

4. 정확도, 충분성, 효율성 및 우아함을 포함하는 평가 기준에 따라 컴퓨팅 작업(프로그램, 웹 사이트 또는 문제 해결법)을 평가할 수 있는 능력
   - 4.1. 컴퓨팅 작업(예: 프로그램, 웹 사이트 또는 문제 해결법)의 오류를 식별할 수 있는 능력
   - 4.2. 다음을 포함하여 의도된 청중 또는 사용자를 위한 컴퓨팅 산출물 적합성을 평가할 수 있는 능력
     - 4.2.1. 모든 요구 사항이 충족되거나 경쟁 요구 사항이 적절하게 우선 순위로 지정된다.
     - 4.2.2 접근성 (즉, 장애인에 의한 컴퓨터 인공물과의 상호 작용 또는 액세스를 방지하는 장벽을 제거하는 포괄적 인 관행)이 고려되었으며,
     - 4.2.3. 모든 가능한 입력이 고려된다.
   - 4.3. 다양한 기준을 최적화하기 위해 컴퓨팅 산출물을 다르게 구현할 수 있다는 점을 인지할 수 있는 능력(예를 들어, 메모리 사용량, 속도의 면에서 효율성 또는 코드 리팩토링과 같이 보다 쉽게 파악할 수 있는 컴퓨팅 접근 방식 등)
   - 4.4. 간소화, 구조화(well-structuredness), 모듈화 및 이해력 포함)한 구현 설계의 미학에 근거하여 컴퓨팅 산출물을 평가할 수 있는 능력

4.5. 컴퓨팅 산출물의 모습이나 느낌(사용자 인터페이스나 사용성)에 대하여 미학적 관점에서 산출물을 평가하는 능력

기준에 따라 다수의 해결책을 비교하기

5. 몇몇 평가 기준(충분성, 효율성, 정확성 및 간결함)을 기반으로 동일한 문제에 대한 여러 해결법(컴퓨팅 산출물) 간의 절충점을 평가할 수 있는 능력.

추가적인 중심역량(지식, 기술, 속성)

- 컴퓨터 시스템의 하드웨어와 소프트웨어 구성요소에 관한 지식
- 컴퓨팅 해결책들과 관련 있는 평가 기준들(예를 들어 속도, 메모리 사용량, 전력 요구량, 코드의 재사용 가능성 등)에 관한 지식
- 컴퓨팅 산출물을 나타내는 능력

과제의 특징

- 과업들은 학생들에게 하나 혹은 그 이상의 컴퓨팅 작업(혹은 산출물) 예시들을 제공하여야 한다.
- 컴퓨팅 산출물은 다양한 잠재적 구현과 다양한 사용성을 포함하는 명백한 기준들을 가져야 한다.
- 과업들은 산출물의 목표나 의도된 용도를 알려주어야 한다.

가변적 특성

- 제공된 컴퓨팅 작업 예시의 수
- 컴퓨팅 작업에 대해 제공된 세부 사항의 양
- 컴퓨팅 작업의 유형과 기준의 뚜렷함
- 평가 기준에 대해 제공된 세부 사항의 양
- 판별 기준들(오류, 사용성, 수행) 의 난이도 수준
- 산출물이 본인의 것인가 타인의 것인지 여부
- 산출물이 유용성 정도

가능한 관찰

- 컴퓨팅 산출물들에 대한 다른 기준들을 사용하는 서술, 평가, 그리고 비교의 타당성과 완전함
- 요구사항 우선순위에서의 차이, 기준들 간의 절충점 및 구현 방법의 차이를 고려하는 응단의 정도
- 컴퓨팅 산출물을 평가하는 기준의 타당성. 고려된 관점들의 폭
- 필요 요건에 대한 참조를 포함하는 컴퓨팅 산출물을 위해 서술된 목표의 타당성
- 설명이 산출물의 목표와 관련되는 정도

가능한 작업 산출물들

- 양적 및 질적 기준들을 사용한 글이나 도표 등의 형식으로 된 컴퓨팅 산출물의 서술, 평가, 비교
- 평가 기준들과 관점들에 대한 서술
- 필요 요건에 대한 참조를 포함한 컴퓨팅 산출물을 위한 목표의 서술
- 평가와 비교를 위한 기준들이 선택된 이유에 대한 설명

컴퓨팅 작업과 다른 이들의 작업 분석하기
예시 어플리케이션: 스크래치 리믹싱

Source: https://scratch.mit.edu/projects/69988504/

컴퓨팅 사고력은 단순히 문제해결에 대한 것은 아니다. 그것은 또한 창의성이나 자기표현을 지지하는 환경이 필요하다. 구성주의 교육의 맥락에서 스크래치(http://scratch.mit.edu) 환경은 사용자들에게 코드의 블록들을 화면상의 캐릭터들, 움직임들, 그리고 소리들로 바꿀 수 있는 시각적 캔버스를 제공한다. 스크래치 프로젝트 웹사이트는 창작자들이 그들의 이야기들, 시뮬레이션들, 그리고 작품을 다른 사람과 공유할 수 있도록 해준다. 이렇게 공유되었을 때, 이들을 만들기 위해 사용된 코드들을 다른 이들도 사용할 수 있게 된다.

스크래치는 다른 사람들의 작업을 공유하고 이를 토대로 작업하는 일반적인 프로그래밍 방식에 기반하여 컴퓨팅 사고력 실행으로 "재사용 및 리믹싱"을 지원한다. 이는 코드 제작자가 코드 이해, 코드 소유권에 기여하는 실제적 상황을 만들어주며, 프로그래머가 더 복잡한 코드를 만들 수 있도록 해주기도 한다. 다른 컴퓨팅 사고력 실행들은 점증과 반복 테스팅과 디버깅, 그리고 추상화와 모듈화를 포함한다.

재사용과 리믹싱은 컴퓨팅 작업 디자인 패턴 분석에서 다음의 FKSAs 사용한다.

1. 컴퓨터 산출물(프로그램, 웹사이트 또는 문제해결법)과 구현 요구 사항(비용 및 배송 플랫폼 등)의 목표 및 결과를 설명 할 수 있는 능력
4. 정확도, 충분성, 효율성 및 우아함을 포함하는 평가 기준에 따라 컴퓨팅 작업(프로그램, 웹 사이트 또는 문제 해결법)을 평가할 수 있는 능력

다른 디자인 패턴의 FKSAs 또한 이 작업에 적용될 수도 있을 것이다. 예를 들어, 재사용과 관련하여 법적 혹은 윤리적 이슈들이 있기 때문에 효과 분석의 FKSAs가 적용 가능하다.

20.c 컴퓨팅 기반 혁신으로 제기된 법률 및 윤리적 문제를 평가하는 능력

추상화, 모델의 설계와 적용하기

개요

추상화를 위한 전략적 사고는 컴퓨팅 사고력의 특징이다. 이 디자인 패턴은 일상생활에서 특별한 것을 끄집어내는 아이디어와 표상을 사용하는 과제를 지원한다. 이것은 패턴, 실체 그 자체로, 과정 그 자체로, 또는 실체 또는 과정중의 관계/구조 같은 세부적인 수준을 포함한다. 여기에는 문제에 대한 일반적인 해결책을 설계하거나 보다 광범위한 문제(기능적 추상화)를 포괄하는 특정 솔루션을 일반화하는 것이 포함될 수 있다. 이러한 아이디어와 표현은 다른 맥락(문제 또는 분야)에서 사용될 수 있다. 학생들은 일상생활 속에서 아이템을 찾고, 수학, 모델, 다이어그램, 컴퓨터 프로그램(데이터 추상화), 자연 및 인공 세계에서 발견되는 항목 등의 표현적 속성에 대한 지식을 나타낸다. 또한 학생들이 현상을 표현하기 위한 모델의 한계에 대한 이해와 모델 또는 추상화의 목적에 주목하는 것이 표현된다.

중심 역량(지식, 기술, 속성)

컴퓨팅에서 추상화에 관한 기초지식

1. 기능적 추상화와 데이터 추상화가 무엇인지 설명할 수 있는 능력
2. 세부적인 여러 단계에서 문제를 추론 할 수 있는 능력
3. 문제 해결에 있어 추상화의 효과를 설명할 수 있는 능력(복잡성 관리 혹은 패턴 일반화)
4. 알고리즘은 특정 시작 상태가 주어지면 마지막 상태 혹은 출력하는 명령어를 포함하는 추상화 양식임을 설명할 수 있는 능력
5. 추상화가 유용한 문제의 특성을 설명할 수 있는 능력
6. 컴퓨터가 실제 세계를 표현하는 방법을 설명할 수 있는 능력
7. 컴퓨팅과 모델링을 위해 컴퓨터가 어떻게 수학적 객체나 논리적 실행을 표현하는지 설명 할 수 있는 능력
8. 컴퓨터가 어떻게 데이터로서의 객체 및 객체로서의 데이터(미디어 파일, QR코드)를 표현하는지 설명 할 수 있는 능력
9. 이진수, 논리, 집합 및 함수를 포함 한 수학 및 컴퓨터 과학 요소간의 연관성을 설명할 수 있는 능력

모델이나 추상화의 분석

10. 데이터를 분석하여 모델링 및 시뮬레이션을 통해 패턴을 구별할 수 있는 능력
11. 모델이나 시뮬레이션이 현상을 추상화하는데 어떻게 도움이 되는지 평가하고 다양한 입력에 어떻게 반응하는지와 같은 현상을 컴퓨팅적으로 이해하는 능력
12. 추상화의 목적에 따라 추상화가 적절한 이유(또는 적절하지 않은 이유)를 설명할 수 있는 능력
    - 12.a. 개체 또는 프로세스의 어떤 기능이 추상화에서 포착이 되는지 그리고 추상화가 주어진 목적을 위한 또는 프로세스의 필수적인 기능을 포착하는 방법을 설명하는 기능
    - 12.b. 특정 목적에 대한 문제의 복잡성을 관리하기 위해 추상화의 가치(세부정보를 숨김으로서)를 설명할 수 있는 능력

12.c. 기능적으로 중요한 요인/특징이(모델 또는 시뮬레이션) 포착되거나 생략되었기 때문에 추상화가 주어진 목적에 적합한지 분석 할 수 있는 능력
12.d. 문제를 정확하고 단순화하고 쉽게 해결할 수 있는 올바른 가정이 있음으로 인해 추상화가 목적에 부합하는지 인지하거나 분석할 수 있는 능력

산출물 그리고/또는 모델 사이의 맵핑으로써 추상화와 모델의 생성

13. 복잡성을 단순화하기 위해 문제 혹은 프로세스를 세분화 할 수 있는 능력
14. 문제를 분해하거나 복잡성을 단순화 혹은 하위 문제들로 나눌 수 있는 능력
15. 복잡한 프로그래밍 해결책을 세부사항을 숨겨 쉽게하는 어플리케이션의 인터페이스나 라이브러리의 목적과 쓰임을 설명할 수 있는 능력
16. 특정 해결방법을 일반화하기 위해 매개변수가 사용되는 방법을 설명할 수 있는 능력
17. 복잡한 문제를 단순한 부분으로 나누는 분류 및 방법을 정의 할 수 있는 능력

해결방법 디자인

18. 문제 또는 해결 방법을 표현하기 위해 추상화 설계할 수 있는 능력
19. 특성, 특색, 특징, 관계 그리고/또는 특정 목적을 달성하기 위해 문제 구조를 포착하는 알고리즘을 만들 수 있는 능력
20. 알고리즘의 특성들이 어떻게 문제의 특성에 맵핑되는지를 설명할 수 있는 능력
21. 각 단계의 구성요소와 구성요소 간의 관계 및 단계간의 설명을 포함 하여 세부 사항에서의 문제점의 해결책을 만들 수 있는 능력
22. 구현과 명세를 분리하는 수단으로 추상화를 사용할 수 있는 능력
23. 표준 설계 도구를 사용하여 해결책의 각 개체 또는 프로세스 간의 관계 그리고/또는 해결책의 개체, 프로세스의 특정 측면을 전반적으로 표현 할 수 있는 능력

컴퓨팅적 모델링, 시뮬레이션, 데이터 조작 사용에 대한 분석

24. 다양한 방법으로 플로팅 될 수 있는 텍스트의 집합이나 대용량 데이터 세트의 유용한 패턴을 찾기 위해 다양한 분석기법을 적용 할 수 있는 능력
25. 컴퓨터를 사용하여 데이터에 대한 가설을 테스트 할 수 있는 능력
26. 컴퓨터가 지능형 행동 모델(로봇동작, 음성 및 언어 인식, 컴퓨터 비전)을 사용하는 방법을 설명할 수 있는 능력
27. 디지털 데이터에 대한 추론에서 추상화가 사용 되는 방법을 식별/설명 할 수 있는 능력(데이터 표현 방법 및 이진 시퀀스 해석 방법 포함)

다양한 표현

28. 그래프, 다이어그램, 문화적 산출물, 스토리 보드 또는 컴퓨터 프로그램과 같은 다양한 표현을 사용하여 추상화를 표현 할 수 있는 능력
29. 문제 상태, 구조 및 데이터를 시각적(그래프, 차트, 네트워크 다이어그램, 순서도 등)으로 표현하는 능력

30. 텍스트, 사운드, 그림 및 숫자를 포함한 다양한 방식으로 데이터를 표현하는 능력
31. 자동화 된 분석에 적합한 대용량 데이터 세트의 기능을 설명하는 능력
32. 데이터 교환의 형태로서 사람, 컴퓨터 등의 의사 소통을 분석 할 수 있는 능력

추가적인 중심역량(지식, 기술, 속성)

- 수학적 개념과 표현에 대한 지식
- 문제에 대한 해결책을 코드화하는 능력
- 다양한 컴퓨팅 모델에 대한 지식
- 특정 알고리즘에 대한 지식
- 알고리즘의 특성에 대한 지식

과제의 특징

- 추상화가 적절히 포함된 컴퓨팅 작업 또는 과제가 제시되어야 한다.
- 과제는 실제 상황에 연결되어야한다.

가변적 특징

- 학생에게 주어진 추상화인가, 아니면 학생이 제시하는 추상화인가?
- 추상화의 난이도
- 실제 문제에서 추상화가 얼마나 많이 제거 되었는가?
- 추상이 수학을 포함하는 정도 (논리 포함)
- 추상화의 표현 (또는 표현의 유형)
- 주어진 시나리오의 유형과 복잡성 (시나리오가 없는 것을 포함)
- 학생들이 일반화 하도록 요구되는 상황의 수
- 주어진 문제가 분해 될 수 있는 정도
- 문제 공간 분석에 사용하기 적합한 기술 유형
- 제공된 데이터 세트의 크기
- 문제 공간이 친숙한 정도
- 문제 공간이 잘 정의된 정도
- 그룹 또는 개별화된 일
- 추상화와 문제 사이의 요소들의 대응
- 그들이 일반적으로 추상화에 대해 이야기하는지 또는 특정 문제에 적용되는지 여부 잠재력 관찰

가능한 관찰 요소들

- 추상화가 문제의 필요성과 일치하는 정도
- 표현의 정확성
- 사용된 표현의 수
- 표현의 적합성
- 설명의 타당성
- 구현이 추상화와 일치하는 정도

- 구현에 추상화가 적용되는 정도
- 맵(the map)이 문제와 요소에 적합한 정도
- 추상화에 대한 설명이나 문서의 명확성 정도
- 분석의 적절성 정도
- 분석의 정확성
- 응용 프로그램의 적합성 또는 추상화 적용의 정확성
- 구현의 정확성 또는 잠재력 있는 작업 제품의 추상적 구현

가능한 작업 산출물들

- 추상화, 문제, 문제 영역 또는 분석에 대한 하나 이상의 표현
- 추상화에 대한 설명 또는 관련 사항 (적용 방법, 왜 적절한가 등)
- 추상화 구현
- 문제의 요소와 추상화 요소 간의 맵핑
- 구현 또는 추상화에 대한 설명 또는 문서화
- 추상화 또는 모델 분석
- 추상화의 응용

추상화 및 모델의 개발 및 적용
적용 예제 : 게임 디자인에서의 추상화

Scalable Game Design (SGD)프로젝트는 시각적이고 드래그 앤 드롭 방식이며 에이전트 기반 언어인 AgentSheets를 사용하여 게임 및 과학 시뮬레이션을 디자인하는 학생을 지원하는 교육학과 강력한 교사 교육이라는 도구의 조합을 사용한다. 기존의 컴퓨팅 교육 및 STEM 과정에서 제공되는 커리큘럼은 학생들에게 게임디자인 혹은 나아가서 STEM주제에서의 모의실험 디자인을 통해 컴퓨팅 사고력을 소개하고 있다. (예 : 산불, 포식자 - 먹이 모델). SGD는 실행 가능하고 오류가 없는 유형의 컴퓨팅 결과물을 산출한다는 최종 목표를 위해 적시에 기술을 습득하여 즉각적으로 적용할 수 있도록 하는 "프로젝트 우선" 접근 방식을 지지한다.

SGD 연구원은 게임 센터에 제출한 수천 건의 학생 제출물에 대하여 잠재 의미론적 분석을 적용했으며 게임과 시뮬레이션에서 에이전트/객체의 상호 작용을 설명하는 일반적인 패턴 집합을 식별하였다. 충돌, 확산, 경로 발견 및 요구 계층과 같은 컴퓨팅 사고 패턴은 학생들에게 명확하게 가르쳐지고, 프로그램에 이런 패턴의 사용은 학생들의 기교(sophistication)가 향상된 증거로 간주된다(Repenning, Webb, & Ioannidou, 2010).

SGD의 패턴은 특정 상황에서 설계를 통해 문제를 이해하고 해결할 때 추상화를 사용하는 좋은 예이다. 그들은 다음과 같은 FKSAs를 예시로 든다 :

2. 세부적인 여러 단계에서 문제를 추론 할 수 있는 능력
3. 문제 해결에 있어 추상화의 효과를 설명할 수 있는 능력(복잡성 관리 혹은 패턴 일반화)
12. 추상화의 목적에 따라 추상화가 적절한 이유 (또는 적절하지 않은 이유)를 설명할 수 있는 능력
20. 알고리즘의 특성들이 어떻게 문제의 특성에 맵핑되는지를 설명할 수 있는 능력

이러한 FKSAs 및 SGD 활동을 사용하여, 학생들이 이러한 패턴에 대해 더 깊이 이해하고 추상화를 수행하는 평가 과제를 만들기 위해 추상화 및 모델링 디자인 패턴을 예시로 들 수 있다.

창의적인 해결방법 및 산출물 설계 및 구현

개요

디자인 패턴은 학생들로 하여금 참신한 아이디어와 문제 해법을 컴퓨터적 해법과 (로봇공학같은 컴퓨팅 혹은 컴퓨터프로그램으로 구현된 해법 같은) 산출물로 변환하도록 하는 과제개발을 지원한다. 학생은 주어진 목적이나 의도를 따라 디자인하고 수행한다.(이 경우 프로그램은 의사소통적인 행위로써 볼 수 있다.) 이 디자인 패턴은 문제해결과 창의적인 과정들의 이해, 분해, 탐험(즉 스토리보드에 문제를 발표하는 것과 다른 플로우차트, 창의성에 의해 유사코드)을 단계를 포함한다. 이러한 산출물은 하나 혹은 그 이상의 디자인 해법과 산출물을 생산한다.

중심 역량(지식, 기술, 속성)

프레임 / 문제 탐색 / 접근

1. 실용적, 개인적 또는 사회적 측면, 개인 호기심 또는 창의성 표현과 같이 여러 목적으로 설계되는 컴퓨팅 해결을 이해하는 능력
2. 컴퓨팅 산출물 및 산출물 자체를 설계하는 데 사용되는 절차의 창의적인 측면을 설명하는 능력
3. 주어진 문제와 최종 결과 또는 필연적 결과에 관해 문제 / 접근 방법을 기술 할 수 있는 능력
4. 새로운 문제 또는 목적에의 측면에서 기존 프로그램 및 해결책 (학생 자신의 창조물 포함)을 평가할 수 있는 능력
5. 현재 계획 중인 프로그램이 예상되는 입력의 결과로 무엇을 출력할 것인가에 대하여 기술 할 수 있는 능력
   5.a. 경계조건(경계사례)과 어떻게 프로그램이 그것들을 처리하는가에 대해 알 수 있는 능력
6. 문제/접근 방식에 대한 설명을 반복 할 수 있는 능력

문제/접근방식을 분해하고 해결책/산출물을 설계하기

7. 문제를 다루기 쉬운지 어려운지, 혹은 컴퓨팅으로 해결할 수 있거나 해결할 수 없는 것으로 분류할 수 있는 능력
8. 문제를 보다 쉽게 다루거나 이해하기 쉽게 하기 위해 하위 부분으로 나누는 능력
9. 팀과 공동으로 해결방법을 구축하기 위해 문제를 나눌 수 있는 능력
10. 전체적인 문제 해결책을 이끌어 낼 수 있는 하위 부분들을 결합하여 해결책을 구성 할 수 있는 능력
11. 컴퓨팅 해결책에 대하여 무엇이 논리적이고 비논리적인 입력인지 알아 낼 수 있는 능력
12. 컴퓨팅 해결책을 만들어 낼 때 염두에 두어야 할 경계조건을 알아 낼 수 있는 능력
13. 원하는 입력 범위를 처리하고 경계조건/경계사례를 처리 할 수 있는 컴퓨팅 해결책을 설계 할 수 있는 능력
14. 문제 해결을 위한 여러 가지 접근 방식을 생성할 수 있는 능력
15. 문제 해결을 위한 여러 가지 접근 방식을 비교할 수 있는 능력
16. 컴퓨팅 해결책의 설계를 반복적으로 개선할 수 있는 능력(구현 결과를 기반으로)

17. 문제해결방안 또는 산출물의 구현에 대한 명세서(specification)를 작성할 수 있는 능력.

해결책/산출물 구현

18. 문제에 대한 효과적인 해결방안을 만드는 데 사용할 수 있는 도구들(프로그래밍 언어 및 소프트웨어 개발 환경), 접근 방식(재귀) 및 방법들(프로그램과 별도의 말이나 표시로 문제에 대한 해결방안을 기술하는 것)을 이해한다.
19. 주어진 집합내에서 어떤 컴퓨팅 도구가 컴퓨팅 산출물을 생성하는 데 가장 적합한 도구인지 평가할 수 있는 능력.
20. 특정 평가 기준에 근거한 문제를 구현하기 위한 여러 접근방식/기법 간의 절충점(trade-off)을 비교하는 능력. 예를 들어 재귀를 통해 프로그램을 더 우아하게 (코드는 더 복잡하지만) 해결할 수 있고, 코드 일부분을 함수 또는 메소드로 만들 수도 있으며(사전 계획이 필요하지만 코드가 보다 모듈화 됨), 특정 데이터 구조를 사용하면 메모리가 더 많이 필요하지만 실행 시간은 단축 될 수 있다.
21. 코딩, 테스트 및 디버깅 과정을 반복하여 프로그래밍 언어로 (원하는 입력 범위를 처리하고 경계조건을 다룰 수 있는) 문제에 대한 완전한 해결방안을 코딩하는 능력
22. 실행 가능한 문제해결방안(코드)을 만들기 위해 다음과 같이 프로그래밍 구조를 사용할 수 있는 능력:
    22.a. 알고리즘 명령어에서 순차적 구조를 사용할 수 있는 능력
    22.b. 조건문을 사용하여 선택 기준에 따라 다른 경로를 통합할 수 있는 능력,
    22.c. 부울 논리 표현식을 사용하여 선택 및 루핑 기준을 통합 할 수 있는 능력 및
    22.d. 루핑 구조를 사용하여 반복적인 코드 블록들을 통합할 수 있는 능력
23. 특정 컴퓨팅 산출물과 해당 요구사항 사이의 정렬(alignment)을 설명하는 능력(산출물이 문제 요구사항이나 접근방법과 다른 점을 포함)

테스트/디버그/해결책 개선/산출물

24. 런타임 오류의 원인을 효율적으로 식별할 수 있는 능력.
25. 런타임 오류의 원인을 설명할 수 있는 능력.
26. 오류 탐지를 위한 체계적인 방법 (예를 들어, 테스트 케이스, 유닛 테스트, 화이트 박스, 블랙 박스 및 통합 테스트)(such as test cases, unit testing, white box, black box, and integration testing)을 설명하는 능력.
27. 컴퓨팅 해결책을 점검하고 수정하기 위한 테스트 및 디버깅 방법을 구현할 수 있는 능력.

추가적인 중심역량(지식, 기술, 속성)

- 기술용어에 대한 지식
- 특정 프로그래밍 언어에 대한 지식
- 특정 디자인 환경에 대한 지식
- 다른 영역/분야에 대한 지식

과제 특징들

- 컴퓨팅 해결책이 필요한 문제 또는 상황을 제시해야 한다.

가변적 특성

- 필요한 컴퓨팅 해결책의 난이도
- 컴퓨팅 해결책의 유형
- 제공된 문제 유형
- 컴퓨팅 해결책의 표현
- 컴퓨팅 해결책이 제시되거나 요구되는지 여부
- 컴퓨팅 해결책이 해결방안의 문제/상황/요구 사항을 해결하는 정도
- 학생이 해결방안을 제시하는지/해결방안을 평가하는지 또는 여러 가지 해결책을 비교하는지 여부

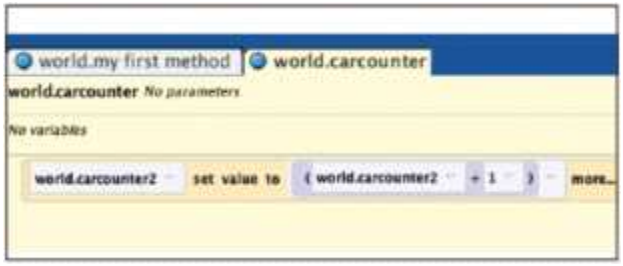
가능한 관찰 요소들

- 식별된 목적이 컴퓨팅 해결책과 관련된 정도
- 설계 과정에 대한 설명의 정확성
- 설계 과정에 대한 설명의 완전성
- 컴퓨팅 해결책이 문제를 해결하는 정도
- 컴퓨팅 해결책의 유연성
- 컴퓨팅 해결책의 복잡성 수준
- 컴퓨팅 해결책의 정확성 (예를 들면 지정된 입력에 따라 예상된 출력을 제공하는 정도, 오류없이 실행되는 정도)
- 컴퓨팅 해결책에서 프로그래밍 구조 사용의 적절성
- 컴퓨팅 해결책의 효율성
- 컴퓨팅 해결책의 설명 또는 묘사의 정확성
- 컴퓨팅 해결책의 설명 또는 묘사의 완전성 (설명 또는 기술이 컴퓨팅 해결책의 여러 측면을 다루는 정도)
- 문제 및 문제 공간(problem space)에 대한 설명의 정확성
- 문제 및 문제 공간에 대한 설명의 완전성 (또는 설명이 문제 또는 문제 공간의 여러 측면을 다루는 정도)
- 다른 도구에 대한 설명의 정확성
- 디버깅 과정에 대한 설명의 완전성 (또는 설명이 디버깅 과정의 여러 측면을 다루는 정도)
- 디버깅 과정의 효율성
- 디버깅 과정에서 오류를 발견하거나(하고) 수정한 정도
- 두 해결방안 또는 두 전략 간의 차이점을 비교한 정도
- 해결방안 또는 전략에 대한 비교의 정확성

가능한 작업 산출물들

- 컴퓨팅 해결책의 목적(들)에 대한 식별
- 설계 과정에 대한 묘사
- 컴퓨팅 해결책
- 컴퓨팅 해결책의 묘사 또는 설명
- 문제 및 문제 공간(space)에 대한 설명(문제의 하위부분 및 경계 조건(boundary condition)에 대한 설명 포함)
- 산출물을 생성하기 위해 어떻게 다른 도구들이 사용될 수 있는지 또는 어떻게 사용되었는지 설명하기
- 디버깅 과정(특정한 컴퓨팅 해결책에 적용되거나 일반적으로 적용되는)을 묘사하기
- 디버깅 과정을 추적하기
- 여러 개의 컴퓨팅 해결책이나 전략을 비교하기

창의적인 해결책과 산출물의 설계 및 구현
적용 예시: Game Creation in Alice



게임 프로그래밍은 복잡한 문제 해결에서 학생들을 참여시켜 창의적인 컴퓨팅 산출물을 설계하고 구현하는 방법으로써 중고등학교에서 점점 더 많이 소개되고 있다. Alice와 Storytelling Alice를 사용하여 ETR 협회와 UC Santa Cruz 연구팀은 역사적으로 컴퓨팅 분야에서 소외되었던 여학생들과 소수민족학생들에게 중점을 두고 어린이와 중학교의 게임프로그래밍을 10년 이상 연구했다.

중학생들이 만든 게임에는 일반적으로 추리하기, 수집하기, 객체 은닉(hiding objects)과 같은 게임 메커닉(mechanic)이 포함되어 있다. 게임은 간단한 이벤트 핸들러에서 (학생이 만든)방법, 조건문, 반복문, 변수(리스트를 포함한), 이진 논리에 이르는 다양한 프로그래밍 구조를 필요로 한다. 학생들은 '카운터' 정수 변수 생성, 초기화 및 증가, 변수 값에 따라 게임 상태 변경 등의 프로그래밍 패턴을 게임을 만드는 과정에서 사용한다.

게임프로그래밍은 창의적인 해결책과 산출물을 설계하고 실행하는 아래의 FKSAs를 따른다.

1. 실용적, 개인적 또는 사회적 측면, 개인 호기심 또는 창의성 표현과 같이 여러 목적으로 설계되는 컴퓨팅 해결을 이해하는 능력
10. 전체적인 문제 해결책을 이끌어 낼 수 있는 하위 부분들을 결합하여 해결책을 구성 할 수 있는 능력
11. 컴퓨팅 해결책에 대하여 무엇이 논리적이고 비논리적인 입력인지 알아 낼 수 있는 능력
16. 컴퓨팅 해결책의 설계를 반복적으로 개선할 수 있는 능력(구현 결과를 기반으로)
21. 코딩, 테스트 및 디버깅 과정을 반복하여 프로그래밍 언어로 (원하는 입력 범위를 처리하고 경계 조건을 다룰 수 있는) 문제에 대한 완전한 해결방안을 코딩하는 능력
22. 실행 가능한 문제해결방안(코드)을 만들기 위해 다음과 같이 프로그래밍 구조를 사용할 수 있는 능력:
    - 22.a. 알고리즘 명령어에서 순차적 구조를 사용할 수 있는 능력
    - 22.b. 조건문을 사용하여 선택 기준에 따라 다른 경로를 통합할 수 있는 능력,
    - 22.c. 부울 논리 표현식을 사용하여 선택 및 루핑 기준을 통합 할 수 있는 능력 및
    - 22.d. 루핑 구조를 사용하여 반복적인 코드 블록들을 통합할 수 있는 능력

추상화와 모델링과 같은 다른 설계 양식에서 FKSAs는 게임 프로그래밍도 적용된다.

컴퓨팅 사고력에서 의사소통과 협력

이 두 가지 실행은 다른 컴퓨팅 사고력과 연계되는 것(예를 들어 창의적인 해결책과 산출물을 설계하고 구현하는 부분에서 의사소통하고 협력할 수 있음)은 물론 본질적으로 수행을 기반으로 한다. 이 디자인 패턴은 우리가 크로스커팅 컴퓨팅 사고력 실행들을 만들 때 의사소통 혹은 협력의 파트를 찾을 수 있는 가이드를 제공해 준다. 의사소통과 협력은 매우 중요하지만(e.g., OECD, 2013), 교육과정에서는 교사들에게 그것을 어떻게 가르치고 평가할 것인지에 대해 매번 제시해 주지 않는다. 예를 들면 우리가 작업한 ECS에서 제시된 학습 목표들을 살펴보면 협력 요소가 불명확하게 되어 있는 것을 볼 수 있다. 비록 이 교육과정의 여러 활동에서 협력 프로젝트들이 있어도(다른 목표들과 마찬가지로 이 영역도 ECS 교사 전문 개발 과정에서 이미 기술된 목표들보다 더 많이 다루어질 수 있다), 직접적으로 협력이 연계된 ECS 목표는 하나밖에 찾을 수 없다. 의사소통과 협력의 디자인 패턴에 대한 우리의 설계는 ECD를 적용한 시나리오 기반 학습에서 영감을 받았다. 관련된 작업은 이런 실행들이 포함된 작업환경에서 제시된 협력 프레임워크 안에서 콘텐츠 중심 기능(우리의 컴퓨팅 사고력 수행들과 같이)을 평가하기 위한 접근 방법들을 검토한다(Davier & Halpin, 2013).

사고 과정과 결과를 의사소통하기

개요

컴퓨팅 산출물에 대해 의사소통하는 것은 컴퓨팅 사고력의 여러 단계를 지원한다. 이런 디자인 패턴을 통해 특정 청중들에게 적합한 방법으로 그들의 작업의 과정과 결과를 전달할 수 있다는 것을 보여주는 과제를 개발할 수 있다. 학생들은 컴퓨팅과 관련된 주요 테마와 아이디어들을 그래프, 시각화, 컴퓨팅 분석을 통해 말과 글로 명확하게 전달할 수 있다.

컴퓨터 과학 원리

- 결과에 대한 의미 설명
- 기술이나 산출물의 영향에 대한 설명
- 컴퓨팅 산출물 작동에 대한 요약
- 산출물의 설계에 대한 설명
- 기술이나 산출물에 대한 설명
- 적합성과 정확성에 대한 정당화

중심 역량(지식, 기술, 속성)

1. 글이나 시각 문서들(리포트, 에세이, 프리젠테이션 슬라이드, 비디오와 같은)을 생성하는 능력. 이런 문서들은 컴퓨팅 산출물, 컴퓨팅 문제나 의도, 문제 해결, 혹은 컴퓨팅 산출물이나 문제 해결(프로그램 문서 포함)의 개발을 지원하기 위한 과정을 설명한다.
2. 컴퓨팅 산출물, 컴퓨팅 문제, 컴퓨팅 문제/의도나 컴퓨팅 문제 해결방법/설계에 대한 생각을 다양한 표현(스토리보드, 그래프, 플로우 차트, 코드문서)을 사용하여 의사소통 할 수 있는 능력

3. 컴퓨팅 산출물, 컴퓨팅 문제나 의도, 문제 해결, 다양한 목적을 위한 컴퓨팅 산출물이나 문제 해결의 개발을 지원하기 위한 과정을(의사결정이나 결과 평가과 같은 청중의 다양한 목적에 부합하는 모든 핵심 정보를 포함하여) 설명할 수 있는 능력
4. 기술적 이해도가 있는 청중이나 일반적인 청중에게 명확하게 컴퓨팅 산출물이나 해결방법의 정보에 대해 말로 조직하고 설명하고 표현할 수 있는 능력. 청중은 다음을 포함한다.
   4.a. 사용자
   4.b. 팀구성원
   4.c. 개작자, 다른 개발자
5. 말, 행동, 시각적 단서를를 활용하여 청중에게 아이디어를 명확하게 표현하고 그들을 콘텐츠에 참여시키는 능력

추가적인 지식, 기술, 태도

- 도메인 속 기술적 어휘에 대한 이해(도메인 특정 언어)
- 아이디어 의사소통을 위한 적합한 전략에 대한 지식(말, 행동, 시각적 단서)

가변적 특성

- 의도된 청중은 기술적 혹은 비 기술적일 수 있다.

컴퓨팅 활동에서 동료와의 협업

개요

이 디자인 패턴은 학생들이 공동으로 문제를 해결하고 컴퓨터 산출물을 설계 / 생성함으로써 컴퓨터 과학의 협업 측면에 참여할 수 있는 능력을 입증하는 작업 개발을 지원한다. 학생들은 짝 프로그래밍 및 그룹 또는 짝을 포함한 프로젝트 팀에서의 작업과 같은 공동 작업을 통해 동료, 전문가 및 다른 사람들과 함께 작업 할 수 있음을 보여준다.

더 넓은 정의는 PISA 협동 적 문제 해결 프레임 워크에서 비롯됨:

> 협력적 문제 해결 또는 협력적 설계 역량은 솔루션/설계에 필요한 지식, 기술에 대한 이해 및 솔루션/설계에 도달하기 위한 노력을 공유함으로써 팀(두명 이상으로 구성)으로 문제를 해결하거나 산출물을 디자인하는 프로세스에 효과적으로 참여할 수 있는 능력을 말한다(PISA, 2013, 7 페이지에서 수정).

컴퓨터 과학 원리들

- 효과적인 팀워크 방식의 적용
- 참가자들의 협조
- 여러 참가자의 적극적인 기여에 따른 산출물 생산
- 산출물의 사용법, 기능 및 실행을 설명하는 서류

중심역량(지식, 기술, 속성)

1. 주어진 시나리오에서 필요조건(requirements) 및 문제에 대한 공통된 이해(shared understanding 의역)를 개발하고 논의할 수 있는 능력
2. 그룹 또는 짝 활동 안에서 문제에 대한 해결방법을 적극적으로 찾고, 모으고, 다른 해결 방법을 논의하는 능력
3. 정확하고, 이해하기 쉽고, 재치 있는(tactful) 피드백을 팀원들에게 제공하는 능력
4. 문제와 그 해결방법들에 대한 다양한 관점들을 이해하고, 가치있게 여기고, 받아들이는 능력
5. 컴퓨팅 사고력 문제에 대한 해결책을 개발하기 위해 여러 관점의 의견을 통합하는 능력 (예를 들면 코드 리뷰, 고객과 사양에 대해서 논의함, 사용성 테스트)
6. 다른 시차(different time spans)와 지역을 넘어서 협력하고 참가하는 능력
7. 공동 작업의 도구(Dropbox auto-sync, Google docs와 같은)와 기술(짝 프로그래밍, 코드 리뷰와 같은)을 사용하는 능력

대인관계 관련

8. 여러 사람들 사이에서 일어나는 갈등과 여러 아이디어 간의 충돌을 해결하는 능력
9. 상호 합의를 위해 노력하고, 팀원과 신뢰를 구축하는 일을 위해 노력하는 능력
10. 모든 사람이 팀 내에서 역할을 수행하도록 만들고, 모든 필수적인 과업이(task) 다루어지도록 하며, 함께하는 모든 과업이 전반적인 목표를 달성하도록 만드는 능력
11. 팀에서 유용한 역할 즉, 문제、의도、접근 방식에 대한 이해도를 높이고, 결과를 개선시키는 역할을 담당하는 능력
12. 팀 전체에 분포되어있는 전문성을 인식하고, 구축할 수 있는 능력

추가적인 중심역량(지식, 기술, 속성)

- 다른 학생과 일하는 데 노력을 기울이는 것을 기꺼이 하는 마음
- 소프트웨어 산출물(software products)의 개발과 디자인에 협동이 어떻게 영향을 미치는지 발견하는 능력
- 의사소통 능력(다른 디자인 패턴 참조)
- 무엇이 효과적인 팀이나 짝을 만드는지 발견하는 능력과 팀이 얼마나 잘 했는지를 반영시킬 수 있는 능력 (예를 들면 경쟁이 아닌 협력 ; 공헌, 뒤로 물러서지 않기)
- 최고의 짝을 찾는 능력 (컴퓨팅 작업을 위한)
- 효과적인 협력을 위해 물리적인 공간을 구성하는 능력 (예 : 두 학생이 모니터를 함께 볼 수 있는 방)

가능한 관찰 요소들

- 공동 관심 : 학생들은 같은 것을 찾는다.
- 상호 약속 : 학생들은 서로 적극적으로 참여한다.
- 개별 활동 : 학생들은 각자 팀 내에서 활동하고, 그 활동에서 배울 수 있는 책임과 기회를 가지고 있다.

- 그룹 활동과 책임 : 학생들은 협의하거나 만들거나 문제를 해결에 함께한다.
- 그룹 조직 : 학생들은 역할과 책임을 맡고, 진행 상황 측정법을 설계했다.
- 건설적인 담화 패턴들 : 그룹에 대한 이해, 팀에 기여, 기여 인정, 공통점을 찾기, 도움을 주기, 도움 받기 등을 증진시키기 위해 높은 수준의 질문하기
- 모니터링하고, 팀워크에 반영하기 (즉, 메타인지와 자기 조절 (meta-cognition and self-regulation))

과제의 특징

- 컴퓨팅 과업은 팀에서 충돌이 일어나고 다수의 입력이 통합되기에 충분히 복잡하다.

가변적 특성

- 팀 멤버들 : 동료, 전문가, 사용자
- 팀 멤버들의 지역

디자인 패턴의 사용

영역 모델링이 완료되면, 평가 논의는 작업 및 평가, 평가 절차, 측정 모델에 대한 상세 설명으로 추가 설명된다(개념적인 평가 프레임워크 레이어 참조). 종종, 요구되는 중심역량을 이끌어내는 관찰 가능한 수행 상황에 학생들을 배치하는 일을 개발하는 것에 중점을 둔다. 작업 수행 관찰에는 다양한 형태가 있다: 지필평가; 산출물 분석(Brennan & Resnick, 2012; Werner, Denner, & Campe, 2014); 산출물 기반 인터뷰(Barron et al., 2002; Grover, Pea, & Cooper, 2015); 게임 플레이에서의 전략, 전술 및 움직임(Kerr & Chung, 2012; Shute, 2011); 시뮬레이션 기반 및 개인 학습 환경에서의 학생들의 움직임(Gobert et al., 2013); 학생들의 사고를 가시화할 수 있는 환경(Linn, Clark, & Slotta, 2003); 학생들의 학습을 돕거나 방해하는 인지적 • 비인지적 요소를 드러내는 분석 기반 환경의 학습에 관한 최근 연구 등(Berland, Martin, Benton, Smith, & Davis, 2013). 증거는 수동적인 관찰을 통해 얻는 것이 아니라 의도적으로 학생들을 필요한 증거를 이끌어내는 상황, 도전, 또는 과제에 둠으로써 얻는다(Grover & Bienkowski, forthcoming).

개념적 평가 프레임워크 단계에서 중심역량은 학생 모델을 공식화하기 위해 선택(그리고 결합이 가능하게)된다. 평가 설계자는 커리큘럼에서 특정한 학습 목표에 부합하도록 하기위해 다루어야 할 중심역량과 수정 사항(있는 경우)을 결정한다. 동반 모델(companion)과 증거 모델에서, 설계자는 작업 채점 방법, 점수가 적용될 측정 모델, 점수의 의미를 규정한다. 그런 다음 두 모델의 결합은 평가를 기반으로 한 학생 수행에 대해 어떤 것이 추론될지 정확하게 규정할 수 있다.

또한 설계자는 개념적 평가 프레임워크 단계에서 작업 모델을 정의한다. 작업 모델은 포함될 작업의 수와 유형을 명시하고, 항목의 형태(예, 지필)와 학생이 완료해야 하는 시간과 같은 평가의 구체적 요구 사항을 설명한다. 학생, 증거, 작업 모델은 엄격한 선형 방식으로 만들어지는 것이 아니다; 오히려 평가 설계자는 새로운 평가를 위해 모델들을 결합하거나 반복적으로 적용하여 개선시키고 논리적으로 일관성 있는 기반을 마련한다. 그러므로 작업 모델이 생성되면, 학생과 증거모델의 초기 작업에 영향을 미칠 수 있다.

마지막으로, 평가 실행 및 전달(delivery) 단계에서는, 항목 및 평가 양식이 타당성의 기준에 따라 개발, 검토, 검증된다(AERA/APA/NCME, 2014). 예를 들어, 초기 파일럿 작업에서는 학생들이 과제 완료 시 사고구술(think-alouds), 전문가 리뷰, 대상 집단에 가까운 학생들의 샘플을 시험하여 항목 난이도를 보정하고 학생들의 하위 그룹간에 차이가 있는지를 결정하기 위한 파일럿 테스팅을 포함할 수 있다(e.g, differential item functioning analyses, Osterlind & Everson, 2009). 이후의 현장 실험은 더 광범위하고 다양한 학생 표본을 포함할 수 있으며, 그 시점에서 심리 측정 작업이 진행될 것이다.

요약 및 향후 연구

우리의 경험 및 다른 사람들의 연구(Haertel, et al., in press-a)에서도 알 수 있듯이 평가 설계에 대한 엄격하고 원칙적인 접근은 잘 정의된 지식과 기술에 대한 유효한 평가 뿐만 아니라 새로운 맥락에서 동일한 지식이나 기술을 측정하는 데 사용할 수 있는 새로운 세트에 대한 템플릿을 산출한다. 본 보고서에서 명시한 디자인 패턴은 학습내용을 전달하는 여러 (수업)방법과 다양한 학습 환경에서 컴퓨팅 사고의 수행(practices)과 협업, 의사소통의 내용을 평가할 수 있는 템플릿을 만들기 위한 첫걸음이다. 본 디자인 패턴을 평가개발에 사용하기 위해서는 (우선) 교육과정에 진술된 학습목표와 맞춰 보아야 한다.

평가를 개발할 때 고려해야할 추가적인 사항들이 있다. 그 중 한 가지는 본 보고서에서 명시한 디자인 패턴은 현재 컴퓨팅 사고에 속한 (하위) 지식을 명시하지 않았다. 왜냐하면 필요한 지식은 어떤 기술이 가르쳐지는가라는 맥락에 따라 어느 정도 달라질 수 있기 때문이다. 그러므로 특정내용에 관한 평가를 개발할 때에는 실습활동과 같은 맥락의 지식을 명확히 하는 것이 중요하다. 이것은 학생들이 실습에 어려움을 겪는지, 지식으로 어려움을 겪는지를 판단하는 항목을 개발하는 데 도움이 될 수 있다. 활동평가의 제작에 있어 특정 지식내용은 핵심 지식이나 기술이 아니라 추가적으로 활용가능한 것으로 사용될 수 있으며, 이러한 내용들은 교사에게 진단정보를 제공하여줄 것이다.

이런 점에서 우리의 작업은 고등학교 과정의 컴퓨터 과학 개론의 구성요소인 네 가지 핵심활동(core practices)과 두 가지 교차활동(cross cutting practices)을 적용하기 위한 AP computer science Principles 과 ECS Curricula 에서 제작한 결론에 입각하여 진행되었다. 이 작업들은 컴퓨터과학영역의 전반적 지식, 기술을 대표하지는 않으나 이전의 폭은 넓으나 깊이가 낮은(milewide, inch-deep) K-12 과학표준에 반대하는 의미로 핵심활동을 대변한다.

또한 과학 및 수학의 최신 기준과 같이 (이 작업들은) 컴퓨팅 사고에 관한 문제들을 해결하기 위한 방법 설계 및 확인 기술의 적용을 평가한다. 적절(필요)한 경우 각 활동의 중심역량(FKSAs)은 협업과 의사소통의 맥락에서 평가될 수 있으며, 경우에 따라 독립적으로 평가될 수도 있다.

증거기반 디자인(Evidence-centered design)은 과제가 학습(과정)을 측정하기 위해 사용될 때의 증거로 사용할 때 우리의 논의에 대해 확신을 주었다. (ECD의) 디자인 패턴은 (평가의) 측정을 위해 어떤 것이 중요한지를 설명하여 주며 중요내용을 측정하기 위해 어떤 방식으로 접근해야 하는지에 관한 안내(가이드라인)를 제공하여 준다.

전에 언급했듯이 설계방식을 만드는 것은 필연적으로 (여러)학문의 내용들이 얽히고 반복된 과정이다. 이 보고서에서 제시한 디자인 패턴은 컴퓨팅 사고 활동영역의 평가개발을 목적으로 원칙적으로 검토되고 있지만 초기 단계이다. 때문에 평가 설계(개발) 과정은 진행 중이며, 더 많은 컴퓨터과학 교육 및 평가 동호회(커뮤니티)의 참여를 기대한다. 개선된 최신 내용들은 우리 웹사이트 pact.sri.com에서 항상 확인가능하다.

Glossary

Advanced Placement(AP) = 대학교과선수이수제

agent-based modeling = 에이전트 기반 모델링

algorithmic notion = 알고리즘개념

artifacts = 산출물

assessment framework = 평가체제

causal reasoning = 인과적추론

Common Core (State) Standards = 미국 중핵표준교육과정(미국 42개 주 교육감협의회에서 주도하는)

computational notions = 컴퓨팅개념

domain = 영역

computational work (such as a program, website, or problem solution) = 컴퓨팅 작업

the computational thinking practices patterns = 컴퓨팅 사고력 실행 패턴

the Computer Science teachers Association(CSTA) = 미국정보과학교사협회

construct centered approach = 구조중심 접근 (Kagan 협동학습 원리)

construct definition = 구조정의

data analysis = 데이터 분석

design patterns = 디자인 패턴(Design patterns are meant to be generative of multiple tasks and to guide assessment specialists in developing tasks to assess both knowledge and skills in the context of specific learning experiences.)

discipline = 학문분야

domain analysis = 영역 분석

domain modeling = 영역 모델링

evidence centered design = 근거기반 디자인

Exploring Computer Science(ECS) = 고등학교 기초 컴퓨터과학 교과

Focal knowledge skills and other attributes(FKSAs) = 중심역량(지식,기술,속성)

frameworks = 체계

information technology fluency standards = IT 유창성 표준

the International Socienty for Technology in Education(ISTE)

K-12 = 유초중등교육

literature(in CS education) = CS 관련 문헌자료

modeling = 모델링

National Governors Association = 전미교육감협의회

National Research Council(NRC) = 미 국가연구회의(그냥 NRC로 표기)

New Generation Science Standards(NGSS) = 미국 차세대 표준과학교육과정

NSF(National Science Foundaton) = 미국 국가과학재단

pilot-test = 파일럿 실험

principled approach of computational thinking(PACT) = PACT

problem solving = 문제 해결

simulation = 시뮬레이션

SRI = SRI연구재단

STEM = 과학기술교육(우리나라는 융합교육 또는 스팀교육)

Structured problem decomposition (modularizing) = 구조적문제분해 (모듈화)

reasoning = 추론

supporting construct = 지원 구조

systems thinking = 시스템사고 (체계적 사고)

terminology = 전문용어

unit assessment = 단원 평가

cumulative assessment = 총괄 평가

(across units 1-4)

visualizing data = 데이터 시각화

References and Bibliography

2020 Computing [Special issue]. (2006). *Nature, 440*(7083). doi:10.1038/440413a

Adams, J. B. (2008). Computational science as a twenty-first century discipline in the liberal arts. *Journal of Computing Sciences in Colleges, 23*(5), 15–23.

Aho, A. V. (2011). Ubiquity Symposium: Computation and Computational Thinking. *Ubiquity, 2011*(January). doi:10.1145/1922681.1922682

Alvarado, C., Dodds, Z., & Libeskind-Hadas, R. (2012). Increasing women's participation in computing at Harvey Mudd College. *ACM Inroads, 3*(4), 55–64.

American Educational Research Association (AERA), American Psychological Association (APA) & National Council on Measurement in Education (NCME). (2014). *The Standards for Educational and Psychological Testing*. Washington, DC: AERA. Retrieved December 5, 2015 from http://www.apa.org/science/programs/testing/standards.aspx

Armoni, M. (2009). Reduction in computer science: A (mostly) quantitative analysis of reductive solutions to algorithmic problems. *JERIC, 8*(4), 1–30.

Arpaci-Dusseau, A., Astrachan, O., Barnett, D., Bauer, M., Carrell, M., Dovi, R., Franke, B., Gardner, C., Gray, J., Griffin, J., Kick, R., Kuemmel, A., Morelli, R., Muralidhar, D., Osborne, R. B., & Uche, C. (2013). *Computer science principles: Analysis of a proposed advanced placement course*. Paper presented at the 44th ACM Technical Symposium on Computer Science Education, Denver, CO. doi:10.1145/2445196.2445273

Astrachan, O., Hambrusch, S., Peckham, J., & Settle, A. (2009). T*he present and future of computational thinking*. Paper presented at the Proceedings of the 40th ACM Technical Symposium on Computer Science Education, Chatanooga, TN. doi: 10.1145/1508865.1509053

Astrachan, O., & Briggs, A. (2012). The CS Principles Project. *ACM Inroads, 3*(2), 38–42. doi:10.1145/2189835.2189849

Barr, V., & Stephenson, C. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48-54. doi:10.1145/1929887.1929905.

Barr, D., Harrison, J. & Conery. L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning & Leading with Technology, 38*(6) 20-23. Retrieved from http://csta.acm.org/Curriculum/sub/CurrFiles/LLCTArticle.pdf

Barron, B., Martin, C., Roberts, E., Osipovich, A., & Ross, M. (2002). Assisting and assessing the development of technological fluencies: Insights from a project-based approach to teaching computer science. *Proceedings of the Conference on Computer Support for Collaborative Learning,* Boulder, CO. 668–669.

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). *Recognizing computational thinking patterns*. Paper presented at the 42nd ACM Technical Symposium on Computer Science Education. doi:10.1145/1953163.1953241.

Basawapatna, A. R., Repenning, A., & Lewis, C. H. (2013). *The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking*. Paper presented at the 44th ACM Technical Symposium on Computer Science Education (501-506). doi:10.1145/2445196.2445346

Basu, S., Dickes, A., Kinnebrew, J.S., Sengupta, P., & Biswas, G. (2013). *CTSiM: A Computational Thinking Environment for Learning Science through Simulation and Modeling*. Paper presented at the 5th International Conference on Computer Supported Education, Aachen, Germany. Retrieved December 1, 2015 from http://www.teachableagents.org/papers/2013/Basu_ICCE.pdf

Bennedssen, J., & Caspersen, M. E. (2008, September). Abstraction ability as an indicator of success for learning computing science? In *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 15-26). ACM.

Bennett, V. E., Koh, K., & Repenning, A. (2013). *Computing Creativity: Divergence in Computational Thinking*. Paper presented at the 44th ACM Technical Symposium on Computer Science Education. Denver, CO. doi:10.1145/2445196.2445302

Berdik, C. (2015, November 24). *What a School District Designed for Computational Thinking Looks Like*. KQED Mind/Shift. Retrieved from http://ww2.kqed.org/mindshift/2015/11/24/what-a-school-district-designed-for-computational-thinking-looks-like/

Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. *Journal of the Learning Sciences, 22*(4), 1–36. doi:10.1080/10508406.2013.836655

Bienkowski, M. & Snow, E. (2014). *Building evidence and building practice in Computer Science education*: White paper presented at the 2014 Future Directions in Computer Science Education Summit. Orlando, FL. Retrieved from https://stacks.stanford.edu/file/druid:mn485tg1952/BienkowskiMarieSRI.pdf

Bienkowski, M., Rutstein, D., & Snow, E. (2015). *Computer Science Concepts in the Next Generation Science Standards*. Presented at the 2015 annual meeting of the American Educational Research Association (AERA), Chicago, IL.

Bienkowski, M. (2015). Making Computer Science a First-Class Object in the K-12 Next Generation Science Standards (Abstract Only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, p. 513.

Bootstrap. (2015). Bootstrap materials: curriculum and software. Retrieved December 5, 2015 from http://www.bootstrapworld.org/materials/fall2015/index.shtml

Bouvier, D., Chen, T.-Y., Lewandowski, G., McCartney, R., Sanders, K., & VanDeGrift, T. (2012). User Interface Evaluation by Novices. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 327–332). New York, NY, USA: ACM. doi:10.1145/2325296.2325372

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada.

Buffum, P. S., Lobene, E. V., Frankosky, M. H., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2015). A Practical Guide to Developing and Validating Computer Science Knowledge Assessments with Application to Middle School. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 622–627). New York, NY, USA: ACM. doi:10.1145/2676723.2677295

Camp, T. (2012). "Computing, we have a problem…." *ACM Inroads, 3*(4), 34–40. doi:10.1145/2381083.2381097

Cápay, M., & Magdin, M. (2013). Alternative methods of teaching algorithms. *Procedia-Social and Behavioral Sciences, 83,* 431–436.

Chazelle, B. (2012). Natural Algorithms and Influence Systems. *Communications of the ACM, 55*(12), 101–110. doi:10.1145/2380656.2380679

Cheng, B. H., Ructtinger, L., & Fujii, R. (2010). *Assessing Systems Thinking and Complexity in Science* (*Technical Report No. 7)*. Menlo Park, CA: SRI International. Retrieved from https://www.sri.com/work/publications/assessing-systems-thinking-and-complexity-science

Colburn, T., & Shute, G. (2007). Abstraction in computer science. *Minds and Machines, 17*(2), 169–184.

Cortina, T. (2007). An Introduction to Computer Science for Non-Majors Using Principles of Computation. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 218-222). doi:10.1145/1227310.1227387

Danielsiek, H., Paul, W., & Vahrenhold, J. (2012). Detecting and understanding students' misconceptions related to algorithms and data structures. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 21-26). doi: 10.1145/2157136.2157148

Davier, A. A., & Halpin, P. F. (2013). *Collaborative problem solving and the assessment of cognitive skills: Psychometric considerations*. ETS Research Report Series. Retrieved November 15, 2015 from https://www.ets.org/Media/Research/pdf/RR-13-41.pdf

DeBarger, A. H., & Snow, E. (2010). *Design Pattern On Model Use In Interdependence Among Living Systems (Large-Scale Assessment Technical Report 13)*. Menlo Park, CA: SRI International. Retrieved from https://www.sri.com/work/publications/design-pattern-model-use-interdependence-among-living-systems-large-scale-assessme

della Cava, M. (2015). Should students learn coding? Students, schools disagree, poll finds. USA Today. Retrieved from http://www.usatoday.com/story/tech/2015/08/20/google-gallup-poll-finds-parents-want-computer-science-education-but-administrators-arent-sure/31991889/

Denning, P. J. (2009). The Profession of IT beyond computational thinking. *Communications of the ACM, 52*(8), 28-30.

Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying Elementary Students' Pre-Instructional Ability to Develop Algorithms and Step-By-Step Instructions. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 511-516). doi: 10.1145/2538862.2538905

Fan, L. (2012, July). Learning of algorithms: A theoretical model with focus on cognitive development. *The 12th International Congress on Mathematics Education,* Seoul, Korea. Retrieved from http://eprints.soton.ac.uk/358300/

Forišek, M., & Steinová, M. (2012). Metaphors and Analogies for Teaching Algorithms. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 15–20). doi:10.1145/2157136.2157147

Friend, M., & Cutler, R. (2013, August). *Efficient egg drop contests: How middle school girls think about algorithmic efficiency*. Paper presented at the International Computing Education Research Conference, San Diego, CA.

Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. In R. T. Mittermeir (Ed.), *Informatics education–The bridge between using and understanding computers*. (pp. 159–168). Berlin, Germany: Springer-Verlag

Gal-Ezer, J., & Zur, E. (2002). The concept of "algorithm efficiency" in the high school CS curriculum. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*. Boston, MA.

Gal-Ezer, J., & Zur, E. (2003). The efficiency of algorithms—Misconceptions. *Computers & Education, 42*(3), 215–226. doi:10.1016/j.compedu.2003.07.004.

Gal-Ezer, J., & Stephenson, C. (2010). Computer science teacher preparation is critical. *ACM Inroads, 1(*1), 61–66.

Gallup. (2015). *Searching for Computer Science: Access and Barriers in U.S. K-12 Education*. Gallup. Retrieved from http://csedu.gallup.com/183725/landscape-computer-science-education.aspx

Garcia, D. D., Harvey, B., & Segars, L. (2012). CS principles pilot at University of California, Berkeley. *ACM Inroads, 3*(2), 58-60. doi:10.1145/2189835.2189853

Garner, S., Haden, P., & Robins, A. (2005). My program is correct but it doesn't run: A preliminary investigation of novice programmers' problems. In A. Young & D. Tolhurst (Eds.), *Proceedings of the 7th Australasian Conference on Computing Education*, Vol. 42 Australian Computer Society, Inc.. Darlinghurst, Australia. (pp. 173–180).

Gibson, P. J. (2012, July). T*eaching graph algorithms to children of all ages*. Paper presented at the Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel.

Ginat, D., & Menashe, E. (2015). SOLO Taxonomy for Assessing Novices' Algorithmic Design. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 452–457). doi:10.1145/2676723.2677311

Gobert, J., Sao Pedro, M., Raziuddin, J., and Baker, R. S., (2013). From log files to assessment metrics for science inquiry using educational data mining. *Journal of the Learning Sciences. 22*(4), 521-563.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin, 40*(1), 256-260.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The Exploring Computer Science program. *ACM Inroads, 3*(2). doi:10.1145/2189835.2189851

Goode, J., & Margolis, J. (2011). Exploring computer science: A case study of school reform. *ACM Transactions on Computing Education, 11*(2), 1–16. doi:10.1145/1993069.1993076

Google for Education. (n.d.). *Exploring Computational Thinking: CT Overview.* Retrieved December 5, 2015, from www.google.com/edu/resources/programs/exploring-computational-thinking/

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38-43. doi:10.3102/0013189X12463051

Grover, S., & Pea, R. (2012). Using a discourse-intensive pedagogy and Android's App Inventor for introducing computational concepts to middle school students. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 723-728). doi: 10.1145/2445196.2445404

Grover, S. (2013, May 28). Learning to Code Isn't Enough. Retrieved from https://www.edsurge.com/news/2013-05-28-opinion-learning-to-code-isn-t-enough

Grover, S., Pea, R., & Cooper, S. (2015). *"Systems of Assessments" for Deeper Learning of Computational Thinking in K-12.* Paper presented at the Annual Meeting of the American Educational Research Association, Chicago, IL.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), 199–237. doi:10.1080/08993408.2015.1033142

Grover, S., & Bienkowski, M. (forthcoming). *Process over product for studying computational thinking practices in introductory programming.* Accepted for presentation at the Annual Meeting of the American Educational Research Association, Washington DC.

Guzdial, M. (2008). Paving the way for computational thinking. Communications of the *ACM 51*(8), 25-27.

Haertel, G. D, Vendlinki, T. P., Rutstein, D., DeBarger, A., Cheng, B. H., Snow, Eric B., D'Angelo, C., Harris, C., Yarnall, L., & Ructtinger, L. (in press-a). General introduction to evidence-centered design. In H. Braun (Ed.), *Meeting the challenges to measurement in an era of accountability* (pp. 107-148). London, UK: Routledge

Haertel, G. D., Vendlinski, T. P., Rutstein, D., DeBarger, A., Cheng, B. H., Ziker, C., Harris, C. J., D'Angelo, C., Snow, E. B., Bienkowski, M., & Ructtinger, L. (in press-b). Assessing the life sciences: Using evidence-centered design for accountability purposes. In H. Braun (Ed.), *Meeting the challenges to measurement in an era of accountability.* London, UK: Routledge.

Hasni, T. F., & Lodhi, F. (2011). Teaching Problem Solving Effectively. *ACM Inroads, 2*(3), 58–62. doi:10.1145/2003616.2003636

Hazzan, O., Lapidot, T., & Ragonis, N. (2011). *Guide to teaching computer science: An activity-based approach.* New York, NY: Springer.

Hazzan, O. (2003). How Students Attempt to Reduce Abstraction in the Learning of Mathematics and in the Learning of Computer Science. *Computer Science Education, 13*(2), 95–122. doi:10.1076/csed.13.2.95.14202

Hazzan, O. (2008). Reflections on Teaching Abstraction and Other Soft Ideas. *ACM SIGCSE Bulletin, 40*(2), 40–43. doi:10.1145/1383602.1383631

Hazzan, O., & Kramer, J. (2007). Abstraction in Computer Science & Software Engineering: A Pedagogical Perspective. *Frontier Journal, 4*(1), 6–14. Retrieved from https://www.hwswworld.com/pdfs/frontier35.pdf

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads, 1*(2), 4-7.

Hershkowitz, R., Schwarz, B. B., & Dreyfus, T. (2001). Abstraction in Context: Epistemic Actions. *Journal for Research in Mathematics Education, 32*(2), 195–222. doi:10.2307/749673

Hill, J. H. (2007). Applying abstraction to master complexity: The comparison of abstraction ability in computer science majors with students in other disciplines. *Proceedings of the 2nd International Workshop on the Role of Abstraction in Software Engineering* (pp. 15-21). doi: 10.1145/1370164.1370169

Hu, C. (2011). Computational Thinking: What It Might Mean and What We Might Do About It. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 223–227). doi:10.1145/1999747.1999811

Jona, K., Wilensky, U., Trouille, L., Horn, M., Orton, K., Weintrop, D., & Beheshti, E. (2014). *Embedding Computational Thinking in Science, Technology, Engineering, and Math (CT-STEM).* Presented at the Future Directions in Computer Science Education Summit, Orlando, FL. Retrieved from http://ccl.northwestern.edu/papers/2014/OrtonKaiNorthwestern-1.pdf

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. (2010). Identifying student misconceptions of programming. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 107–111). doi: 10.1145/1734263.1734299

Kafai, Y. B., Peppler, K. A., & Chapman, R. N. (Eds.). (2009). *The Computer Clubhouse: Constructionism and creativity in youth communities*. New York, NY: Teachers College Press.

Kerr, D., & Chung, K. W. K. (2012). Identifying key features of student performance in educational video games and simulations through cluster analysis. *Journal of Educational Data Mining, 4*(1), 144–182.

Koh, K.H., Basawapatna, A.R., Bennett, V.E., & Repenning, A. (2010) Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. *Proc. of VL/HCC '10*, IEEE Computer, Madrid, Spain, 59-66

Kramer, J. (2007). Is Abstraction the Key to Computing? *Communications of the ACM, 50*(4), 36–42. doi:10.1145/1232743.1232745

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., et al. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32-37. doi:10.1145/1929887.1929902

Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 57-62). doi: 10.1145/2157136.2157156

Linn, M. C., Clark, D., & Slotta, J. D. (2003). WISE design for knowledge integration. *Science Education, 87*(4), 517-538.

Lister, R. (2012). The CC2013 Strawman and Bloom's Taxonomy. *ACM Inroads, 3*(2), 12–13. doi:10.1145/2189835.2189840

Liu, M., & Haertel, G. (2011). *Design Patterns: A Tool to Support Assessment Task Authoring (Large-Scale Assessment Technical Report 11).* Menlo Park, CA: SRI International. Retrieved from http://ecd.sri.com/downloads/ECD_TR11_DP_Supporting_Task_Authoring.pdf

Loman, N., & Watson. (2013). So you want to be a computational biologist? *Nature Biotechnology, 31*, 996-998. doi:10.1038/nbt.2740

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 101–112). doi:10.1145/1404520.1404531

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ., 10*(4), 16:1–16:15. doi:10.1145/1868358.1868363

Malyn-Smith, J., & Lee, I. (2012). Application of the Occupational Analysis of Computational Thinking-Enabled STEM Professionals as a Program Assessment Tool. *Journal of Computational Science Education, 3*(1), 2–10. Retrieved from http://www.shodor.org/media/content/jocse/volume3/issue1/jocse_volume3_issue1#page=8

Margolis, J., Ryoo, J. J., Sandoval, C. D. M., Lee, C., Goode, J., & Chapman, G. (2012). Beyond Access: Broadening Participation in High School Computer Science. *ACM Inroads, 3*(4), 72–78. doi:10.1145/2381083.2381102

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., B-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*. ACM SIGCSE Bulletin. 33(4). 125-180. doi: 10.1145/572139.572181

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of Programming in Scratch. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 168–172). doi:10.1145/1999747.1999796

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (Moti). (2013). Learning computer science concepts with Scratch. *Computer Science Education, 23*(3), 239–264. doi:10.1080/08993408.2013.832022

Messick, S. (1994). The interplay of evidence and consequences in the validation of performance assessments. *Educational Researcher, 23*(2), 13-23.

Microsoft Corporation. (2014). *Computational Thinking in the Sciences and Beyond*. Retrieved from http://research.microsoft.com/apps/video/dl.aspx?id=232450

Mislevy, R., & Riconscente, M. (2005). *Evidence-centered assessment design: Layers, structures, and terminology (PADI Technical Report 9)*. Menlo Park, CA: SRI International. Retrieved from https://www.sri.com/work/publications/evidence-centered-assessment-design-layers-structures-and-terminology-padi-technic

Mislevy, R. J., & Riconscente, M. M. (2006). Evidence-centered assessment design: Layers, concepts, and terminology. In S. Downing & T. Haladyna (Eds.), *Handbook of Test Development* (pp. 61-90). Mahwah, NJ: Erlbaum

Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2003). *On the Structure of Educational Assessments*. Los Angeles, CA: Center for the Study of Evaluation, National Center for Research on Evaluation, Standards, and Student Testing, Graduate School of Education & Information Studies, University of California, Los Angeles.

Mislevy, R. J., Riconscente, M. M., & Rutstein, D. W. (2009). *Design Patterns for Assessing Model-Based Reasoning (Large-Scale Assessment Technical Report 6)*. Menlo Park, CA: SRI International. Retrieved from http://ecd.sri.com/downloads/ECD_TR6_Model-Based_Reasoning.pdf

Muller, O., Ginat, D., & Haberman, B. (2007). Pattern-oriented Instruction and Its Influence on Problem Decomposition and Solution Construction. In Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (pp. 151–155). New York, NY, USA: ACM. doi:10.1145/1268784.1268830

Muller, O., & Haberman, B. (2009). *A Course Dedicated to Developing Algorithmic Problem Solving Skills - Design and Experiment*. Presented at the PPIG 2009 - 21st Annual Workshop, Limerick, Ireland. Retrieved from http://www.ppig.org/workshops/ppig-2009-21st-annual-workshop

Murphy, L., Fitzgerald, S., Lister, R., & McCauley, R. (2012). Ability to "Explain in Plain English" Linked to Proficiency in Computer-based Programming. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (pp. 111–118). New York, NY, USA: ACM. doi:10.1145/2361276.2361299

National Governors Association (NGA) Center for Best Practices, Council of Chief State School Officers. (2010). *Common Core State Standards*. Washington, DC: National Governors Association Center for Best Practices, Council of Chief State School Officers.

National Research Council. (2012). *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. (Committee on a Conceptual Framework for New K-12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education, Ed.).Washington, DC: The National Academies Press. Retrieved from http://www.nap.edu/catalog/13165/a-framework-for-k-12-science-education-practices-crosscutting-concepts

National Research Council (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: The National Academies Press. Retrieved from http://www.nap.edu/.

National Research Council. (2012). *Report of a workshop on the pedagogical aspects of computational thinking*. Washington, DC: The National Academies Press. Retrieved from http://www.nap.edu/.

NGSS Lead States. (2013). *Next Generation Science Standards: For States, By States*. Washington, DC: National Academy of Sciences.

Nicholson, K., Good, J., & Howland, K. (2009). *Concrete thoughts on abstraction*. Paper presented at the Psychology of Programming Interest Group (PPIG 2009) Limerick, Ireland . Retrieved from http://www.ppig.org/library/paper/concrete-thoughts-abstraction

Northwestern University. (2015). Computational Thinking in STEM: Lesson Plans. Retrieved December 5, 2015, from http://ct-stem.northwestern.edu/lesson-plans/

OECD. (2013, March). PISA 2015: *Draft Collaborative Problem Solving Framework*. Retrieved from http://www.oecd.org/pisa/pisaproducts/Draft%20PISA%202015%20Collaborative%20Problem%20Solving%20Framework%20.pdf

Osterlind, S. J., & Everson, H. T. (2009). *Differential item functioning (2nd ed., Vol. 161)*. Thousand Oaks, CA: SAGE Publications.

Petersen, A., Craig, M., & Zingaro, D. (2011). Reviewing CS1 Exam Question Content. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 631–636). doi:10.1145/1953163.1953340

Phillips, P. (2009). Computational Thinking: a problem-solving tool for every classroom. Retrieved from http://www.csta.acm.org/Resources/sub/ResourceFiles/CompThinking.pdf

Qin, H. (2009). Teaching computational thinking through bioinformatics to biology students. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (pp. 188-191). doi: 10.1145/1508865.1508932

Razzouk, R., & Shute, V. (2012). What Is Design Thinking and Why Is It Important? *Review of Educational Research (82)*3, 330-348. doi:10.3102/0034654312457429

Regev, A., & Shapiro, E. (2002). Cellular abstractions: Cells as computation. *Nature, 419*, 343-419. doi:10.1038/419343a

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 265–269). doi:10.1145/1734263.1734357

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM, 52*(11), 60–67. doi:10.1145/1592761.1592779

Robertson, J., & Howells, C. (2008). Computer game design: Opportunities for successful learning. *Computers & Education, 50*(2), 559-578.

Rutstein, D., Snow, E., & Bienkowski, M. (2014). *Computational thinking practices: Analyzing and modeling a critical domain in computer science education.* Paper presented at the 2014 annual meeting of the American Educational Research Association (AERA), Philadelphia, PA.

Sakhnini, V., & Hazzan, O. (2008). Reducing abstraction in high school computer science education: The case of definition, implementation, and use of abstract data types. *Journal on Educational Resources in Computing (JERIC), 8*(2), 1–13.

Samurçay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 161–178). Mahwah, NJ: Lawrence Erlbaum.

Scalable Game Design: Computational Thinking Patterns. (2014, June 3). Retrieved from http://sgd.cs.colorado.edu/wiki/Category:Computational_Thinking_Patterns

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., ... & Verno, A. (2011). CSTA K--12 Computer Science Standards: Revised 2011.

Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating Computational Thinking with K-12 Science Education Using Agent-based Computation: A Theoretical Framework. *Education and Information Technologies, 18*(2), 351-380. doi:10.1007/s10639-012-9240-x

Shein, E. (2014). Should everybody learn to code? *Communications of the ACM, 57*(2), 16-18. doi:10.1145/2557447

Shute, V. J. (2011). Stealth assessment in computer-based games to support learning. *Computer Games and Instruction, 55*(2), 503–524.

Snow, E., Fulkerson, D., Nichols, P., & Feng, M. (2011). *Design patterns to support storyboards and scenario-based, innovative item types.* Paper presented at the 2011 annual meeting of the American Educational Research Association (AERA), New Orleans, LA.

Sudol-DeLyser, L. A. (2015). Expression of Abstraction: Self Explanation in Code Production. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 272–277). doi:10.1145/2676723.2677222

Sykora, C. (2014, September 11). Computational thinking for all. Retrieved from https://www.iste.org/explore/articleDetail?articleid=152&category=Solutions&article=Computational-thinking-for-all

Szalay, A., & Gray, J. (2006). 2020 Computing: Science in an exponential world. *Nature, 440*, 413-414. . doi:10.1038/440413a

Tadmor, B., & Tidor, B. (2005). Interdisciplinary research and education at the biology-engineering-computer science interface: a perspective. *Drug Discovery Today, 10*(23-24), 1706-1712. doi:10.1016/S1359-6446(05)03702-5

Tew, E. A., & Guzdial, M. (2010). The FCS1: A Language Independent Assessment of CS1 Knowledge. *Proceedings of the The 42nd ACM Technical Symposium on Computer Science Education.* doi:10.1145/1953163.1953200

Tew, A. E., & Guzdial, M. (2010). Developing a Validated Assessment of Fundamental CS1 Concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 97–101). doi:10.1145/1734263.1734297

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's Taxonomy for CS Assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78* (pp. 155–161). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1379249.1379265

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2003). A model curriculum for K–12 computer science: Final report of the ACM K-12 task force curriculum committee. New York, NY: The Association for Computing Machinery.

Vardi, M. Y. (2012). What is an Algorithm? *Communications of the ACM, 55*(3), 5–5. doi:10.1145/2093548.2093549

Walker, H. M. (2012). Developing a Useful Curricular Map. *ACM Inroads, 3*(4), 14–16. doi:10.1145/2381083.2381089

Werner, L., Campe, S., & Denner, J. (2012). Children Learning Computer Science Concepts via Alice Game-programming. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 427–432). doi:10.1145/2157136.2157263

Werner, L., Denner, J., Bliesner, M., & Rex, P. (2009). *Can middle-schoolers use Storytelling Alice to make games? Results of a pilot study.* Paper presented at the 4th International Conference on Foundations of Digital Games. doi:10.1145/1536513.1536552

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 215–220). doi:10.1145/2157136.2157200

Werner, L., Denner, J., & Campe, J. (2014). Children programming games: A strategy for measuring computational learning. *Transactions on Computing Education, 14*(4), 24:21–24:22. doi:10.1145/2677091

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—an embodied modeling approach. *Cognition and Instruction, 24*(2), 171-209. doi:10.1207/s1532690xci2402_1

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). Running on empty: The failure to teach K-12 computer science in the digital age. *Association for Computing Machinery/Computer Science Teachers Association.*

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33–35. doi:10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society a: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725. doi:10.1098/rsta.2008.0118

SRI Education™

SRI Education, a division of SRI International, is tackling the most complex issues in education to identify trends, understand outcomes, and guide policy and practice. We work with federal and state agencies, school districts, foundations, nonprofit organizations, and businesses to provide research-based solutions to challenges posed by rapid social, technological and economic change. SRI International is a nonprofit research institute whose innovations have created new industries, extraordinary marketplace value, and lasting benefits to society.

Silicon Valley
(SRI International headquarters)
333 Ravenswood Avenue
Menlo Park, CA 94025
+1.650.859.2000
education@sri.com

Washington, D.C.
1100 Wilson Boulevard, Suite 2800
Arlington, VA 22209
+1.703.524.2053

www.sri.com/education

Stay Connected