Relevance to workforce/career readiness	5
Applicability to Auri's Journey	4
Novelty for thesis	4

15. Notes for Thesis Synthesis

IDC (2016) highlights that while specific technical skills change, **cross-functional skills**—particularly problem-solving, communication, and digital literacy—are the primary drivers of success in high-growth occupations. To ensure student success, educational tools should shift focus from narrow job training to fostering these broadly applicable "job readiness" competencies.

1. Citation

Author(s): Mahsa Mohaghegh and Michael McCauley. **Year:** 2016. **Title:** Computational Thinking: The Skill Set of the 21st Century. **Journal/Conference:** International Journal of Computer Science and Information Technologies. **Volume/Pages:** Vol. 7 (3), 1524-1530.

2. Study Aim & Context

• **Main Goal:** To introduce **computational thinking (CT)**, examine its characteristics and benefits, and explore its importance in education, specifically for **years 1–13**.

• **Focus:**

  ◦ **Computational thinking as a transferable skill**.

  ◦ **Programming literacy & workforce readiness**.

  ◦ **Problem-solving and abstraction**.

  ◦ **Early programming education**.

• **Field:** Computer Science Education, STEM Education.

3. Core Research Problem & Purpose

The study addresses the **misunderstanding of computational thinking**, which is often incorrectly reduced to "thinking like a computer" or mere programming. There is a recognized need to bring CT to the **core of primary and secondary education** to prepare the next generation for a world filled with technological advances, yet there is complexity in how to effectively integrate these principles into non-CS curricula.

4. Methodology

Aspect	Details
Research Type	Qualitative Review / Literature Synthesis.
Target Learners	**Primary and secondary school students (Years 1–13)**.
Educational Tool / Program	Visual tools (Scratch, App Inventor, Kodu, Alice) and **CS Unplugged**.
Data Collected	Analysis of existing CT frameworks (Wing, Grover, Jaokar) and implementation cases (Kaupapa Māori).
Evaluation Metrics	Problem-solving efficiency, logical reasoning, and integration of technology in cultural/heritage contexts.
Study Setting	Educational systems (notably New Zealand), primary/secondary classrooms.

5. Technology & Programming Features

• **Emphasized Elements:**

◦ **Logical thinking:** Deducing information and forming realistic conclusions (e.g., Sudoku puzzles).

◦ **Algorithmic thinking:** Step-by-step processing and strategic thinking.

◦ **Abstraction and Decomposition:** Breaking down complex tasks into manageable sub-problems.

◦ **Efficiency:** Minimizing time and memory resources required to solve problems.

• **Implementation:** Moving beyond "learning the tools" to understanding the **concepts behind the algorithm**. It suggests using analogies, like comparing **binary strings and parity bits** to simple error-checking tasks.

6. Outcomes & Findings

• **Transferable Skills:** CT allows non-computer scientists to benefit from computational approaches to problem-solving in fields like **biology, geology, and economics**.

• **Innovative Thinking:** CT trains the mind to question existing things and think "outside the box," leading to innovation.

• **Cultural Engagement:** In the **Kaupapa Māori** context, digital literacy programs (like the DNA project) improved students' connection to their language and heritage while developing leadership and problem-solving skills.

• **Workforce Readiness:** CT is identified as one of the **top ten key skills** for the future workforce, specifically for its ability to translate vast data into abstract concepts.

7. Fields & Contexts Impacted

CT is becoming woven into the "very fabric of science" and impacts:

• **STEM:** Biology (molecular functions), Physics (supernovas), Engineering (structural modeling).

• **Business & Finance:** Online auctioning, banking, and data-based reasoning.

• **Social Sciences & Humanities:** Cultural heritage and leadership in indigenous communities.

8. Pedagogical / Educational Implications

• **Early Introduction:** CT should be treated as a **vital skill** alongside reading, writing, and arithmetic from an early age.

• **Conceptual Focus:** Educators must prioritize teaching **how to construct programs and why certain solutions work** over simply teaching coding syntax.

• **Teacher Training:** Teachers need to find areas in their current curricula where they can demonstrate CT through **analogy and terminology**.

9. Practical Takeaways for Auri’s Journey

• **Focus on Logic Over Syntax:** Design Auri’s Journey to emphasize the **steps of problem-solving (algorithms)** rather than just memorizing code.

• **Lower Entry Barriers:** Use **visual programming environments** (similar to Scratch) to allow players to construct programs quickly and focus on logic.

• **Real-World Analogies:** Use familiar, **real-world analogies** within the game to explain abstract computer science concepts like binary or error-checking.

10. Strengths & Novel Contributions

• **Broad Scope:** Examines CT from early childhood through to university and professional levels.

• **Cultural Inclusivity:** Provides specific evidence of how CT can be integrated into **indigenous education (Māori)** to enhance cultural connection.

• **Interdisciplinary Evidence:** Clearly links CT to advances in diverse scientific fields like physics and biology.

11. Limitations & Research Gaps

• **Infrastructure Barriers:** Significant challenges exist in providing the necessary **infrastructure and resources** to all schools.

• **Technological Fear:** Parents and teachers often suffer from a **fear of technology**, which can hinder student engagement.

• **Definition Ambiguity:** There is still no universally agreed-upon definition of exactly what "computational thinking" entails, leading to implementation confusion.

12. Key Quotes / Definitions

• "Computational thinking is the **skill of the 21st Century**".

• "Solving a computational problem involves **logical and algorithmic thinking** approaches".

• "The key skill is in **logically breaking down a problem** and systematically devising an algorithm suitable for solving it".

13. Tags / Keywords

["computational_thinking", "problem_solving", "algorithmic_thinking", "21st_century_skills", "early_education", "Kaupapa_Maori", "STEM_integration"]

14. Relevance Score & Applicability

Category	Rating (1–5)
Relevance to programming skills	⭐ ⭐ ⭐ ⭐ ⭐
Relevance to workforce/career readiness	⭐ ⭐ ⭐ ⭐ ⭐
Applicability to Auri’s Journey	⭐ ⭐ ⭐ ⭐

Novelty for thesis ⭐⭐⭐⭐

15. Notes for Thesis Synthesis

Mohaghegh and McCauley (2016) argue that **computational thinking** is a foundational 21st-century literacy that extends far beyond computer science, enabling problem-solving in diverse fields from biology to indigenous studies. They emphasize that for early education (Years 1–13), the focus must remain on **conceptual understanding and logical decomposition** rather than the rote learning of programming syntax.

1. Citation

Author(s): Martyna K. Fojcik & Marcin Fojcik. **Year:** 2021. **Title:** Teachers Experience with Introducing Programming in Different Courses for Non-Computer Science Students. **Journal/Conference:** *Education and New Developments 2021*, pp. 491-495.

2. Study Aim & Context

• **Main Goal:** To present the requirements and challenges of introducing programming to non-computer scientists from a teacher's perspective, specifically adapting general programming knowledge to the needs of different subjects.

• **Focus:**

◦ **Technology integration across disciplines:** Analyzing how programming fits into non-CS study programs.

◦ **Digital Literacy:** Supporting student development of digital competencies through programming.

◦ **Programming literacy & workforce readiness:** Addressing the changing demands of the labor market.

• **Field:** Higher Education, Teacher Education, and Engineering Education.

3. Core Research Problem & Purpose

The research addresses the **lack of a clear definition** for digital literacy and programming requirements in non-computer science professions. It highlights a gap where university subjects often focus on **theoretical knowledge** or computer science "creator" perspectives rather than the specific **practical skills** needed by different professions (e.g., math teachers vs. engineers). Additionally, it notes the discrepancy between graduate skills and employer expectations for practical technology experience.

4. Methodology

Aspect	Details
Research Type	Qualitative analysis and content comparison.
Target Learners	University students in non-CS programs (Automation Engineering and Teacher Education).
Educational Tool	Two introductory programming courses (HVL and HVO) analyzed via the **SOLO-taxonomy**.
Data Collected	Teacher experiences, national/EU requirements, and course content analysis.
Evaluation Metrics	**SOLO-taxonomy levels** (Pre-structural to Extended Abstract) to measure observed learning outcomes.
Study Setting	Higher education institutions in Norway (HVL and Volda University College).

5. Technology & Programming Features

- **Emphasized Elements:**
  - **Fundamental coding concepts:** Variables, simple loops, and if-sentences.
  - **Algorithmic thinking:** Writing simple algorithms and explaining efficiency/accuracy.
  - **Computational Thinking:** Supporting computational structures and thinking in students/pupils.
  - **Debugging & Logical Reasoning:** Reading others' code and debugging working programs.
- **Implementation:** Features were customized to the profession; for teachers, the focus was on **pedagogical/didactical knowledge**, while for engineers, the focus was on **modeling real-world automation** and software interfaces.

6. Outcomes & Findings

• **Practical vs. Theoretical Gap:** Analysis showed that students often understand programming theory at a higher SOLO-level than they can perform in practical exercises.

• **Course Customization:** Programming for non-computer scientists must be customized to the **pedagogical and didactical requirements** of their specific profession.

• **Competency Area:** Digital competence is successfully categorized into areas like **information literacy, content creation, and problem-solving**, which programming supports.

• **Employment Impact:** Students who engage in practical projects or work-integrated learning double their chances of employment.

7. Fields & Contexts Impacted

The study identifies that **ICT skills** are required for two-thirds of new jobs since 2010. Specifically, the fields of **Automation Engineering** and **Teacher Education** are highlighted, though the findings extend to all non-CS professions where digital tools must be used "responsibly and safely."

8. Pedagogical / Educational Implications

• **Move Beyond Theory:** Higher education must adapt from explaining theoretical tools to helping students develop **practical skills** through teamwork and cooperation.

• **SOLO-taxonomy Application:** Using the SOLO-taxonomy helps teachers identify where students are failing to bridge the gap between "barely anything" (pre-structural) and "extended abstract" (relational understanding).

• **Professional Relevance:** Curriculum design should avoid "overloading" students with unnecessary CS elements (like sorting binary trees for math teachers) and focus on relevant professional application.

9. Practical Takeaways for Auri’s Journey

• **Prioritize Practical Skill over Theory:** Design challenges where the player must **perform** a task (e.g., writing a simple loop) rather than just identifying a theoretical concept.

• **Profession-Specific Mini-Projects:** Include scenarios relevant to non-CS fields, such as **modeling a real-world action** or creating an interface for a user.

• **Support Computational Thinking:** Focus on teaching the player how to **give instructions** to a system and understand "computational structures" rather than just learning syntax.

• **Emphasize "Soft" Skills:** Incorporate elements that require **time management or communication**, as these are the competencies employers actually seek alongside technical knowledge.

10. Strengths & Novel Contributions

• **SOLO-taxonomy Model:** Provides a robust, multi-level framework (Pre-structural to Extended Abstract) specifically tailored to categorize programming outcomes for non-CS students.

• **Teacher-Centric View:** Offers unique insights into the specific challenges instructors face when trying to balance general digital literacy with professional requirements.

11. Limitations & Research Gaps

• **Curriculum Structure:** The current structure of higher education is noted as being poorly prepared for the **cooperative and practical** forms of teaching students now expect.

• **Lack of Coherent Plans:** There is no universal plan for teaching digital competencies, and digital skills are often conflated with programming skills unnecessarily.

12. Key Quotes / Definitions

• **Digital Competence:** "self-confidence, critical thinking, and responsibility for using digital technologies."

• **Teaching Challenge:** "The challenge for today's teachers is to find out what existing and new approaches are needed in teaching and how best to meet students' needs."

13. Tags / Keywords

["digital_literacy", "programming_for_non-computer_scientists", "SOLO-taxonomy", "workforce_readiness", "teacher_education", "higher_education"]

14. Relevance Score & Applicability

Category	Rating (1–5)
Relevance to programming skills	5
Relevance to workforce/career readiness	5
Applicability to Auri’s Journey	4
Novelty for thesis	5

15. Notes for Thesis Synthesis

Fojcik & Fojcik (2021) emphasize that introducing programming to non-computer scientists requires a shift from pure computer science theory to **profession-specific practical application**. Through the lens of the **SOLO-taxonomy**, the authors illustrate that while students may grasp theoretical concepts, there remains a significant gap in their ability to perform practical, digital-literacy-based tasks, highlighting the need for more **project-based, customized learning** in higher education.

1. Citation

Author(s): Amber Settle, Debra S. Goldberg, & Valerie Barr. **Year:** 2013. **Title:** Beyond computer science: computational thinking across disciplines. **Journal/Conference:** ITiCSE '13: Proceedings of the 18th ACM conference on Innovation and technology in computer science education, pp. 289–290.

2. Study Aim & Context

• **Main Goal:** To explore and discuss diverse projects that **integrate computational thinking (CT)** into secondary and undergraduate curricula across **non-computing disciplines**.

• **Focus:**

◦ **Technology integration across disciplines:** Bringing computational content into traditional K-12 and undergraduate courses.

◦ **Computational thinking as a transferable skill:** Establishing CT as a basic skill for every child's education regardless of their primary field.

• **Field:** Computer Science Education and curriculum development.

3. Core Research Problem & Purpose

The study addresses the **low enrollment** and **homogeneous population** typically found in computer science classes. It highlights barriers in the U.S. education system, such as computer science not being a **core high school topic** and a **shortage of qualified teachers** to implement national requirements. The purpose is to provide students with **meaningful exposure** to CT in courses they are already taking (like history or art) without sacrificing the existing learning goals of those subjects.

4. Methodology

Aspect	Details
Research Type	**Mixed/Qualitative** (Panel discussion summarizing multiple NSF-funded projects).
Target Learners	**K-12 students** (middle and high school) and **undergraduates**.
Educational Tool / Program	Integration of CT into existing courses (e.g., Latin, Russian history, biology, art) and **NSF-funded initiatives** like ECSITE.
Data Collected	Student self-reports, faculty feedback from 17 departments, and educational software development.
Evaluation Metrics	**Learning outcomes** (disciplinary material mastery), teacher resource development, and sustainability/dissemination.
Study Setting	**Classroom-based** across multiple institutions including DePaul University, Union College, and 16 K-12 schools.

5. Technology & Programming Features

• **Elements Emphasized:**

◦ **Computational thinking in context:** Teaching CT through disciplinary applications like **cryptography in history** or algorithmic structures in **screenwriting**.

◦ **Computation tracks:** Creating specific tracks within non-computing disciplines.

◦ **Bidirectional learning:** Using computing to both solve disciplinary problems and reveal relationships between problems and solutions.

• **Implementation:** Faculty modified existing general education courses and developed **field-specific tools**, such as educational software and user manuals tailored for non-CS students.

6. Outcomes & Findings

• **Academic Impact:** A majority of students reported that the **computational component helped them learn** the disciplinary material in their respective non-CS courses.

• **Deliverables:** The projects resulted in **publishable research results**, publically available educational software for high schools/colleges, and new tools for researchers.

• **Engagement:** Modified courses at DePaul University successfully spanned **19 different courses** across five liberal studies areas, reaching a diverse student body.

7. Fields & Contexts Impacted

The sources list a wide range of impacted fields: **Art, astronomy, biology, classics, economics, English, experimental humanities, geology, history, music, political science, psychology, health education, mathematics, and social studies.**

8. Pedagogical / Educational Implications

• **21st Century Skills:** CT is viewed as a natural entry point for modern curriculum requirements.

• **Meaningful Exposure:** CT can be integrated without compromising the **traditions or learning goals** of non-computing disciplines.

• **Faculty Collaboration:** Success requires working closely with classroom teachers to ensure CT content is **responsive to the differing needs** of specific content areas.

9. Practical Takeaways for Auri's Journey / Future Tools

• **Teach in Context:** Design beginner-friendly interventions that teach CT through **subjects students already care about** (like art or storytelling) rather than as a standalone technical topic.

• **Support Non-CS Instructors:** Develop **comprehensive teacher resources** to overcome the lack of specialized computer science educators.

• **Identify Disciplinary Barriers:** Focus on **standards and requirements** of the host discipline to ensure the tool is perceived as helpful rather than an additional burden.

10. Strengths & Novel Contributions

• **Scale and Diversity:** Demonstrated successful CT integration across **eight different departments** and multiple school districts.

• **Multi-Institutional:** Included collaborations across several colleges and K-12 districts, providing evidence of **scalability**.

11. Limitations & Research Gaps

• **Systemic Barriers:** The lack of computer science as a **core topic** in K-12 remains a significant hurdle for national-scale implementation.

• **Standards Tension:** Bringing CT into traditional courses is difficult because those courses must already address **numerous pre-existing standards**.

12. Key Quotes / Definitions

• "Computational thinking is an emerging **basic skill** that should become an integral part of every child's education."

• "Computing is **transforming society** and impacting many areas of study."

13. Tags / Keywords

["computational_thinking", "cross_disciplinary", "K-12", "undergraduate", "curriculum_integration", "STEM_education"]

14. Relevance Score & Applicability

Category	Rating
Relevance to programming skills	4
Relevance to workforce/career readiness	4
Applicability to Auri's Journey	5
Novelty for thesis	3

15. Notes for Thesis Synthesis

Settle, Goldberg, and Barr (2013) demonstrate that integrating **computational thinking** into non-computing disciplines like the humanities and sciences provides students with meaningful exposure to computing without requiring them to enroll in traditional CS courses. Their findings suggest that students often find **computational components helpful** in mastering their primary subject matter, supporting the push for CT as a universal basic skill.

1. Citation

Author(s): Andréa Cartile **Year:** 2020 **Title:** Barriers to Digital Literacy: Learning to Program **Journal/Conference:** Proceedings 2020 Canadian Engineering Education Association (CEEA-ACEG20) Conference, Paper 170.

2. Study Aim & Context

• **Main Goal:** To discuss **barriers to acquiring the digital literacy** needed to learn **end-user programming** (programming as a tool for non-computer science domains).

• **Focus:**

  ◦ **Technology integration across disciplines** (Engineering).

  ◦ **Programming literacy & workforce readiness**.

  ◦ **End-user programming proficiency**.

• **Field:** Engineering Education, Digital Literacy.

3. Core Research Problem & Purpose

• **Gap Addressed:** The study identifies a **significant gap in secondary education curriculum** where computer science is not a prerequisite or a formal part of high school education, leading to a "digital literacy deficit" for university freshmen.

• **Problem:** The **inaccessibility and lack of consensus** in online resources, which often assume a high level of prior digital competence, contributing to high failure rates in introductory programming.

4. Methodology

Aspect	Details
Research Type	**Qualitative / Review / Discussion Paper**.
Target Learners	**First-year engineering students** in non-computer-based fields.
Educational Tool	Analysis of existing curricula and **online programming resources** (e.g., W3Schools, Khan Academy, Codecademy).
Data Collected	Evaluation of entrance requirements, secondary curricula, and online learning platforms.

Evaluation Metrics	**Martin's Levels of Digital Literacy** (Competence, Usage, Transformation) and transition scales (Non-programmer to Professional).
Study Setting	**Canadian Accredited Engineering Universities** and online environments.

5. Technology & Programming Features

• **Emphasized Elements:**

◦ **Martin's Three Levels of Digital Literacy:** Level I (Competence/skills), Level II (Usage/application), and Level III (Transformation/innovation).

◦ **End-User Programming:** Developing digital tools to support everyday professional tasks rather than professional software development.

◦ **Programming in Context:** Using code as a tool for domain-specific applications (e.g., finite element modeling).

• **Implementation:** Proposed through the **integration of programming assignments and labs** within existing non-CS courses throughout the degree.

6. Outcomes & Findings

• **The Literacy Gap:** Most engineering students arrive with varying degrees of **self-taught digital competencies**, facing a steep learning curve in formal university courses.

• **Online Resource Barriers:** Many popular tutorials (e.g., Java on W3Schools) claim to require no experience but immediately assume familiarity with complex interfaces like the **Windows Command Prompt**.

• **Lack of Consensus:** Online resources differ wildly in their starting points—ranging from motivation and computational thinking to formal syntax—making it difficult for beginners to navigate.

7. Fields & Contexts Impacted

• **Engineering:** All disciplines (Civil, Mechanical, etc.) increasingly require digital skills for domain-specific software development.

• **General Workforce: 82% of online job postings** require digital skills, and 90% of IT jobs are now found **outside the IT sector**.

8. Pedagogical / Educational Implications

• **Contextualized Learning:** Programming should not be taught as a **"siloed subject"**; it is more effective when used as a tool to support domain-specific applications.

• **Scaffolded Entry:** Educators must recognize that opening a command-line interface and returning "Hello World" requires significant **background information** that beginners may lack.

9. Practical Takeaways for Auri's Journey / Future Tools

1. **Lower the Initial "Logic" Barrier:** Avoid assuming students understand backend interfaces; ensure the tool explains **"what" and "why"** before requiring command-line interactions.

2. **Integrate Programming in Context:** Design mini-projects that solve **real-world engineering or professional problems** to foster "Level II: Digital Usage".

3. **Provide Curated Learning Paths:** Since online resources lack consensus, educational tools should provide a **linear, consensus-based curriculum** to prevent student "resource amassing" and confusion.

10. Strengths & Novel Contributions

• **Adaptation of Literacy Models:** Effectively adapts Martin's (2009) digital literacy levels specifically for the **non-computer-based engineering context**.

• **Identification of "End-User" Needs:** Clearly distinguishes between the needs of professional developers and **end-user programmers**, emphasizing "digital empowerment".

11. Limitations & Research Gaps

• **Systemic Absence:** The lack of computer science in **high-school curricula** (especially in Québec) remains a primary barrier that universities cannot easily fix in a single semester.

• **Constraint of Time:** The standard **13-week university course structure** is often insufficient to bridge the literacy gap and teach coding simultaneously.

12. Key Quotes / Definitions

• **End-User Programming:** "People who write programs, but not as their primary job function".

• **Digital Empowerment:** "Enabling people to grow as competent subjects who have control over their lives and surroundings".

• **The Device Paradox:** "The experience of using a digital device is not necessarily equivalent to digital knowledge".

13. Tags / Keywords

["digital_literacy", "end-user_programming", "engineering_education", "curriculum_gap", "workforce_readiness", "online_resources"].

14. Relevance Score & Applicability

Category	Rating (1-5)
Relevance to programming skills	5
Relevance to workforce/career readiness	5
Applicability to Auri's Journey	4
Novelty for thesis	4

15. Notes for Thesis Synthesis

Cartile (2020) highlights that while digital skills are required for 82% of jobs, a significant gap exists between student digital device usage and true **digital literacy**. The study advocates for **moving away from siloed programming courses** toward an integrated model where programming is a tool for professional innovation, thereby bridging the curriculum gap found in secondary education.