System Design

with

Computational Thinking

Ben Koo

benkoo@tsinghua.edu.dn

iCenter, Tsinghua University

Motivation

Digitally networked resources are enabling modern makers to perform wonders with little private physical resources. This great amplification of resource utilization comes from the fact that clever makers know where and how to identify ideas, technologies, and potential applications. However, compiling these assortments of opportunities take an overwhelming amount of efforts. To overcome this dilemma, this 16-week course guides novice and veteran makers to “computationally” implement every-layer of a networked computing system with an inter-related hardware and software hands-on projects, so that makers would acquire a consistent, yet sufficiently rich vocabulary and engineering principles to implement and test their ideas. These implementation skills and technical principles would usually take years to learn if taught by traditional courses separately.

Who should take this course?

Anyone who wants to become a full-stack digital designer is welcome. This course will present the vocabulary and compositional rules that enables you to perform system design tasks using ideas starting from digital logic to digital network applications. We will guide a novice maker, from knowing almost absolutely nothing, to have a basic operational skill in creating a functional digital product and speak the global language of maker/hackers.

Who qualify to take this course?

Like any subject matter, to learn something well requires focus and willingness to adopt new habits. Although we have verified that most people with little prior knowledge can learn this material, we will still prefer our main student body can dedicate concentrated and continuous efforts to finish the relevant homework in a synchronized matter. Students who register for our course will have to take some pre-class tests by reading some suggested reading materials and some videos and websites, so that we can verify that registered students are capable, yet independent learners, so that we can get participants who have the potential to contribute original thoughts to the class. We also welcome auditors who need assistance while learning. Auditors may learn it on their own pace, and they can form study groups that do not interfere with the progress of our main class.

Core Concepts delivered through this course

This course explains both individual and collective thought processes in the language of computation. In the other hand, computation is a tangible and measurable cognitive mechanism. We want to show that our rationality is bounded by our ability to process information into decisions. The ability is also limited by the require amount of computing time, memory space, and physical energy. A series of tools are introduced to students to show how each layer of computational abstraction is managed, and computational experimented. Students will accomplish certain design tasks to realize how computation can be reduced to different representation, but all relying upon the same logical substrate. To better relate the mental model of computation in a wide variety of application contexts, we will also study the history of personalities, institutions, and technology developmental paths of computing, so that students can relate computational thoughts to many more potential areas.

Learning Objectives

Given that our students can believe in the above statement, they should achieve the following objectives:

1. Value Identification and Standards formulation: Understand how to frame the question of value proposition in the historical context of computing industry. Learn about the historical trends, major industry standards, and meet up and discuss the industry trends with critical personalities in person and in their direct descendants. Students are expected to create a Strategy Analysis Report (SAR) at the end of the study.
2. Technological Vocabulary and Compositional Rules: Learn to use MediaWiki, Git, GitHub, Wolfram|Alpha, Docker, Raspberry Pi, and other open-sourced computational services and tools for vocabulary management. Understand the combinatorial possibilities of the known spectrum of computational models, software tools, hardware technologies, and service providers of computation. How different symbolic systems can be put together to analyze, predict or control certain systems. At the same time, we will learn about the basic notion of the Correctness of System Design, Design by Contract, Algebra of Computer Programs, the Composition of Distributed and Centralized Computing, and present computation results using Human-Machine Interface Technologies. Students are expected to produce a Computational Resource Analysis Report (CAR) at the end of study.
3. Application Context: Given the personal interests and group decisions, students will identify up-to-date applications of computational thinking to their selected subjects. These applications will follow the reasoning framework of computational thinking and documented using computable languages. Students are expected to product an Industry Analysis Report (IAR) at the end of study.

The learning outcomes should be integrated using a digital publishing process. Students will be collecting their ideas using a MediaWiki-like workflow, to capture their ideas, and publish their thought processes after each learning session. Then, the three main threads of this study will produce three respective reports (SAR, CAR, and IAR). Each project is considered to have an incremental contribution, after a complete report is being edited, refined, and authorized to publish. SAR should contain testable statements or test cases to help verify and validate computational models or industry-specific applications. CAR is a report that articulate how to perform the computation using existing tools and services. IAR is the specific report explaining how an industry would utilize computational thinking and what are the known and expected results. Students’ learning outcomes will be judged based on their contribution and the quality of the three reports.

Content Outline

This course is an integrated web of knowledge that presents a bird’s-eye view of the maker technology and cultural landscape. The course contain the following knowledge components:

1. Classic literature and electronic media that best present the essence of field and industry of computation.
2. Learn to use Collective Knowledge Management tools (Mediawiki, Git, Stack Overflow)
3. From Basic Digital Circuits to Software Applications (Nand2Tetris.org)
4. Key Personalities, Technologies, and Historical Institutions of the Maker/Hacker Community

We created the content outline of this course after a group of local makers took the famous online course: “Elements of Computing Systems”, (a.k.a. <http://nand2tetris.org>). We realized that to engage novice maker to better realize their objectives, just learning technical content or just knowing certain anecdotal stories of famous hackers is not enough. Effective makers must first learn to work together or utilize existing network tools to leverage technologies and resources developed by others, yet keep a good habit in using these tools. Then, they also need to have a good knowledge in mathematical/logical reasoning, and use or at least understand the various kinds of tools for different layers of abstraction. Finally, while they are learning about these ideas, makers need to know who and where these technologies or innovative devices came from. With these inter-related contextual knowledge, decision on what to make, and how to make, can be must better supported by a consistent intellectual framework. In other words, this is a digital technology literacy course designed for people who want to “make” in a networked society.

Infrastructure Preparation

This course first introduces popular Internet co-creation tools, such as Mediawiki and GitLab/GitHub services as a common protocol for sharing and managing collective knowledge. Novice makers would not just write code to simulate digital logic circuits, computer architectures, operating systems, programming languages, compilers, etc. they must also frequently store their incremental learning results using version control systems such as Git, and write appropriate learning reports in MediaWiki. This requirement is to build a habit of writing for novice makers. Clearly, to build this program, relevant software and hardware infrastructures must be prepared for all participating students. This include a website for Wiki, and a GitLab/GitHub compatible local file synchronization system, and other related project management software, and public social media services, such as WeChat and Youtube-like content sharing platforms. In this course, students are organized in teams to create documentation and digitized documentary of their own learning process using these popular digital knowledge management systems, and they will soon realize that these networked infrastructures are their venues to engage with the global marketplace.

Therefore, before coming to this class, students are encouraged to play with the basic tools and know about what will be taught and learned in this course. We will prepare material relevant to this class, and students should start experiencing these tools and concepts before starting this class.

|  |  |  |
| --- | --- | --- |
| Week | Content | Detail |
| -2 | Sign up to course explanation website | Read the syllabus carefully, and sign up to GitHub, GitLab, Wikipedia.com, and local MediaWiki website, submit an application to take this course. Watch a few videos on Computational Thinking, and Cognitive Sciences. |
| -1 | Git and Wiki Worshop | Demonstrate the best practices of Git and Wiki. Give a short test to see if students know how to use Git, Read Social Physics and Wikinomics |
| 0 | Models of Computation | An introductory lecture (maybe online) that explains the course, and outlines various models of computation and their applications. This includes Turing Computation, Symbolic Computation, Machine Learning Computation, and Networked Computation. Students needs to know about these terms, so that they can see why this is relevant to their daily lives in this digitized society. |

Technical and Mathematical Content

Makers are the modern alchemists. Modern alchemists; especially digital alchemists have their secret language. The language is a way to declare the testable perimeter of a designed artifact, also known as Design by Contract. To solidify the mindset of system design, Design by Contract (DbC) and automated testing tools are introduced in the very beginning of this course and will become the essential principle to control the quality of learning outcome assessment.

|  |  |  |
| --- | --- | --- |
| Week | Content | Detail |
| H1 | Design by Contract | Truth Table, Boolean Algebra, Digital Circuit Notation, Hardware Description Language |
| H2 | Numerical and Logic Operations, the Definition of Computable Functions | Number encoding system, Half/Full Adder Circuit, Combinatorial Logic, The Workhorse of CPU: Arithmetic Logic Unit, What is a Function? |
| H3 | Memory, the basic unit of Space/ Time | Sequential Logic, Memory Circuitry, Program Counter, Flip-Flop and Clock Diagrams, Basic Instruments in Digital Electronics |
| H4 | Assembler : Symbolic Machine Language: | How to instruct a computer at the lowest level? What are the most primitive computer programs? How machine code is stored and run? |
| H5 | Computer System | Put together a General Purpose Turing Machine: Central Processing Unit, Human-Machine Interface, |
| H6 | Secure System | CPU with Write-Only Memory |
| H7 | Networked Systems | Networked Devices and Network Data Interface |

The course is intentionally divided into two parts. Hardware and software parts are taught in an interleaving manner. Conceptually, students can also learn both tracks separately, however, it would be ideal to learn both tracks in parallel.

|  |  |  |
| --- | --- | --- |
| Week | Content | Detail |
| S1 | Design by Contract | Computer as a data flow system. Use other metaphors, such as material flow to represent computation. Contracts before computation. Software installation, includes development tools, multi-media game API library, and manual installation |
| S2 | Program Control | How to write test cases, Unit Tests, System Tests, Stress Tests, use Video games as examples. |
| S3 | Virtual Machine I: Stack Arithmetic | Abstract Data Structures, Stack Arithmetic, What is Computation? Abstract Machine as a System Design Standardization technique |
| S4 | Virtual Machine II : Function Module | Functions as Subroutines, parameter passing, recursion, memory/resource allocation |
| S5.0 | High Level Programming Language | Language as a machine, Language Specification, Code examples, Software simulates physical applications |
| S5.1 | Language-specific Software Libraries | This is dedicated to the programming language of choice. We will primarily show how Python and its software libraries can be used in this course. Other tools and languages can be used here, too. |
| S6 | Compiler | Syntax Analysis: Parser Design, Grammar Specifications, Parse Tree, |
| S7 | Symbolic Computation | Symbolic Computation, Code Generation, Language-Oriented Engineering |
| S8 | Operating Systems | Data Representation, Resource Allocation, Input/Output Management, Device Drivers |

To further illustrate the power of digital contract, the universal expressive power of Turing Machine is presented in multiple layers of software/hardware systems, including the design and implementation of Assemblers, Compilers, Virtual Machines, Operating Systems, Arithmetic Logic Unit (ALU), and Central Processing Units (CPU).

All of these above systems have test cases, and relevant testing tools, to perform unit tests, system tests, and stress test. How and when these test cases are satisfied, can be an objective indicator for how well learning proceeds. These system design assignments are shown as recurring themes to demonstrate that complex engineered products can be decomposed into modularized, yet computationally testable design contracts.

After taking both hardware and software courses, students should have identified their project team members, and create a collaborative project as teams. The remaining weeks in a 16-week semester is divided as follow:

|  |  |  |
| --- | --- | --- |
| Week | Content | Detail |
| SD9 | System Design by Contract: build a networked system with modern Human-Computer Interfaces | Student team should propose a project that utilizes the above-mentioned computational tools and sensor technologies to build a human-computer interfacing system. This week is a project proposal week, every team should present a 5~10 minute project, and present their “System Design Contracts”. |
| SD10 | Project Control | Review and present their system design test cases, Unit Tests, System Tests, Stress Tests. In certain cases, the team could reverse engineer an existing Video game, and show its test cases. They should also show their program report and each teams’ code commit history. |
| SD11 | Functional Decomposition | Present the Software “Object Model” or “Function Model” of the desired system. Show a data/information flow |
| SD12 | Implementation Data Types and Function Calls | For the previously known system, identify Application Programing Interfaces (API), and tools that performs the specified tasks in the Object/Function Model |
| SD13 | Project Description Manual (Product Demo Material) | Present the Data Dictionary, Product Features, Resource Allocation, Main technical merits, and other product description content as a draft copy as website, digital content asset package. |
| SD14 | Knowledge Refinement | Each team should present their key findings in their respective projects, what can be done differently, what are their best practices. |
| SD15 | Knowledge Compilation: based on Git and Wiki data. | All teams should present their project finding as computational system, using a common set of vocabulary, and review other teams’ knowledge base on wiki and Git. Then, they should identify Design Patterns between different teams |
| SD16 | Final Presentation | Present and demonstrate the product with its digital documentation. |

Historical and Societal Contexts

To prolong students’ interest in absorbing these abstract and technically challenging concept, the entire course is gamified as a system of puzzles that leads participants into a stage-gated digital civilization. The history of computing science, the main personalities, key technologies, and influential institutions are presented to students during relevant course work. Students are required to know about these key personalities and institutions, so that they can see how breakthrough technologies are developed in certain social contexts.

In terms of personalities, we will introduce Alan Turing, Noam Chomsky, Claude Shannon, Richard Stallman, Linus Torvalds, the infamous Satoshi Nakamoto, and Jimmy Wales. These persons are directly responsible for the modern maker culture and how information/knowledge distribution can be democratized. As the course proceed, other personalities will also be included by instructors and students.

When we present technical terms, certain technologies, such as mechanical clock, mechanical calculators, integrated circuits, the Internet, HTTP/HTML, Bitcoin/Blockchain, Virtual Reality and Augmented Reality will be mentioned, and used in various homework assignments when appropriate. The goal is to make students aware that technologies are not just tools, they became popular because they all answer certain social needs. They are both the evidence and the enabler of social evolution. Finally, we will also introduce certain institutions, namely DARPA, Bell Lab, and the Free Software Foundation. These institutions have their social-economic agenda, and they all influenced the world in ways that went beyond the original expectation. Students are expected to incrementally and digitally present their essays on their interpretations of these historical/technical references. They are also welcome to identify other personalities, institutions, and technologies in their essays.

Time Allocation

We expect that learning this course would require 10 to 15 hours per week focusing on the given homework assignments. Weekly content presentation would range from 3 to 5 lecture hours. Some of the lectures can be delivered using MOOC platforms. To reduce the amount of trial and error time expended by novice makers, weekly even daily meet-up with teaching fellows and other students could be very helpful. Usually one hour tutorial or in-person technical per week can resolve a lot of problems. On the other hand, this course can be taught as a college level course with 3 to 5 credits in about 16 weeks.

Learner Feedback

At the point of this writing, the course development team has successfully delivered the hardware aspect of this course to a wide range of students, including 8-year-old boys and 40+ year-old housewives, mostly with little or no background in computation. The goal is to expand this computational approach to refine and deliver best learning practices for other foundational literacy skills, such as math, science and language learning. In the long run, creative making projects based on computational thinking and system design could be an integrative protocol to reveal the quality and quantity of collective intelligence in this increasingly digitized world.

Performance Assessment

The goal of this course is not to grade students, but to help them learn the secrets of great makers. Therefore, grading is not a focus. However, we will still provide feedback to students by observing whether they have frequently submitted their contributions to Wiki and Git. Moreover, as long as they committed their solutions to the testable assignments in specified time. They can receive full credit for each of the assignments. And the overall accumulated percentage of accomplished test cases is the final score. For team performance assessment, we plan to give certain prizes to various teams’ creative products. The quality of teamwork and collaborative effort can also be graded using Wiki and Git track records. However, at the time of this writing, a specific set of rules have not been implemented. The ideal way of doing this is to allow each class of students to write their own Design Contracts as their version of “constitution”.

Content Detail

1. Learn the foundational principles of computation. Knowing the basic building blocks of a Turing-Complete computational system.
2. Learn to classify different models of computation, and realize that given infinite space and time, all these models can emulate the others. The popular models include: imperative model, declarative model, continuous variable simulation, Bayesian inference, Neural network, PAC Machine Learning, functional programming, Idea Flow, and agent-based modeling, …
3. Realize that any functional social organization can be thought of as a computable system. Its constitution, its cultural norms, and its agent-agency interactions can be modeled as computing systems.
4. Learn to classify computational systems using a repeatable analytical framework. Define computational ingredients as: Context, Vocabulary ,Rules
5. Learn about Idea Flow by Alex Pentland, and use the language of Social Physics to create a computational system by establishing a vocabulary, and continuously manage this vocabulary as a name space using modern name space management tools.
6. Know about Virtualization as a popular approach to manage and standardize computation. Including virtual machines, cloud-computing, and Docker-like container approaches.
7. Students will also learn about Computer Architectures, natural language processing and Machine Learning Libraries, Symbolic Computation, Data-Intensive Computation, Emotional Computing, and Social computation
8. Learn to manage an evolvable vocabulary of Data Structures and Algorithms using Wiki-like knowledge management tools.
9. Participate in developing and using the Personal Data Stores (OpenPDS), and understand the mechanisms of how personalized data can be protected and traded. Students will also learn to categorize application-specific data to be stored or manipulated in different kinds of abstract data types.
10. Describe the world as a composition of our proposed abstract data types. After taking this course, students should be able to re-organize information content of any kind in terms of well-defined data structures. Information content includes social, economical, and technological data. They should be learn to measure and compare the performance characteristics once they made choices of data types in describing their world. For example, knowing that file systems are often organized in tree-based data structures, Hypertext documents are linked as a dynamic network, Search engines often present results as lists. Relate these classifications to daily objects and organizations, say social, economical, and technological designs.
11. Pragmatic tools, services, and institutions that are providing computing technologies and services. The “who is who” in history and in modern day computing science/engineering arena. The key technological components and technological metrics/barriers in the field of computing.
12. Get access to latest and greatest computing services and construct and test these computing infrastructures in person, using commercially available virtualization tools, including VR and AR
13. Make decisions computable, and know what are in-computable.
14. Work with real people, real projects and real data
15. Construct a set of real computational model and document it,
16. Use collaborative/networked digital publishing tools to create an Industry Analysis Report for your own product