Backward Design
An Integrated Approach to a Systems Curriculum

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons
Backward Design
The Long and Rewarding Path to an ACM 2013 Systems Fundamentals Examplar

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons
Backward Design
The Long and Rewarding Path to an ACM 2013 Systems Fundamentals Examplar

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons
Context - how we got here

Not listed:
- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases
Context - how we got here

Not listed:
- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases
Context - how we got here

Not listed:
- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Cross-listed with ISAT
Backward Design: An Integrated Approach to a Systems Curriculum
SIGCSE 2015 • Kirkpatrick et al

Context - how we got here

Not listed:
- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Switch to kernel emphasis

Cross-listed with ISAT
Context - how we got here

- Originally exclusively HW focused
- Switch to kernel emphasis
- Cross-listed with ISAT

Not listed:
- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Backward Design: An Integrated Approach to a Systems Curriculum
SIGCSE 2015 • Kirkpatrick et. al
Backward Design
Fixing the Problems that Certain New Faculty Introduced when Hired

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons
Backward Design
Fixing the Problems that Certain New Faculty Introduced when Hired

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons
Context - how we got here

Other problems:
- CS 350 bottleneck
- Insufficient scaffolding
- Redundancies between siloes
- Training vs. education
- Aging curriculum
Context - how we got here

Other problems:
- CS 350 bottleneck
- Insufficient scaffolding
- Redundancies between siloes
- Training vs. education
- Aging curriculum
Backward Design

Identify situational factors
Backward Curricular Design

Identify situational factors
Identify situational factors

Starting points:
- What are our goals for OUR students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?
Backward Curricular Design

Starting points:
- What are our goals for OUR students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?
Backward Curricular Design

Starting points:
- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?
Backward Curricular Design

Starting points:
- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?
Situational factors
Situational factors

Show-stopper constraints
- Three-section, two-prep teaching structure
- Virginia Community College System (VCCS) relationship
- Faculty interests and staffing
Situational factors

Show-stopper constraints
• Three-section, two-prep teaching structure
• Virginia Community College System (VCCS) relationship
• Faculty interests and staffing

Other considerations
• Prolonged CS1 sequence (assumes no background)
• Infosec/Cyber-defense emphasis and strength
• Student career aspirations and proximity to DC
• Courses have reputations
• Liberal arts university - credit hour questions
Defining objectives

- Identify situational factors
- Define objectives
- Summative assessments and metrics
- Formative assessments/instructional activities
Defining objectives

Starting points for objectives:
- What vision do we have for our students?
- How do we make our vision explicit and measurable?
- Where are our students starting from?
- What should they learn beyond just CS?
Taxonomies galore

Bloom’s

- Creating
- Evaluating
- Analyzing
- Applying
- Understanding
- Remembering
Taxonomies galore

Bloom’s

SOLO (Biggs)

UbD (Wiggins & McTighe)

Fink
Taxonomies galore
Taxonomies galore
Defining learning outcomes

<table>
<thead>
<tr>
<th>Foundational</th>
<th>Human dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What key information should our students understand and remember in the future?</td>
<td>7. What should students learn about themselves as software developers, users of technology, or as students?</td>
</tr>
<tr>
<td>2. What key ideas or themes are important for students to comprehend and describe?</td>
<td>8. What should students learn about understanding others and how they interact with others (including through developed software)?</td>
</tr>
<tr>
<td>Application</td>
<td>Caring</td>
</tr>
<tr>
<td>3. What kinds of thinking (critical, creative, practical) are important for our students to learn?</td>
<td>9. What changes and values should students adopt in regard to interests or ideas?</td>
</tr>
<tr>
<td>4. What software development skills do students need to gain?</td>
<td>Learning how to learn</td>
</tr>
<tr>
<td>5. What skills do our students need to learn about managing complex projects?</td>
<td>10. What should students learn about how to work with computer systems?</td>
</tr>
<tr>
<td>Integration</td>
<td>11. What should students learn about becoming a self-directed learner?</td>
</tr>
<tr>
<td>6. What connections should students recognize and make within the curriculum or to professional life?</td>
<td></td>
</tr>
</tbody>
</table>
## Defining learning outcomes

**Foundational**
1. What key information should our students understand and remember in the future?
2. What key ideas or themes are important for students to comprehend and describe?

**Application**
3. What kinds of thinking (critical, creative, practical) are important for our students to learn?
4. What software development skills do students need to gain?
5. What skills do our students need to learn about managing complex projects?

**Integration**
6. What connections should students recognize and make within the curriculum or to professional life?

### Human dimension
7. What should students learn about themselves as software developers, users of technology, or as students?
8. What should students learn about understanding others and how they interact with others (including through developed software)?

### Caring
9. What changes and values should students adopt in regard to interests or ideas?

### Learning how to learn
10. What should students learn about how to work with computer systems?
11. What should students learn about becoming a self-directed learner?
Systems core themes

To reflect trends within the CS field, the JMU CS systems core courses should strive to produce the following characteristics of CS graduates:

- technical understanding of computer and network systems
- appreciation of the interplay between theory and practice
- system-level perspective (thinking in levels of abstraction)
- understanding of how to identify and evaluate trade-offs in design and implementation
- ability to identify common problem patterns and apply appropriate solutions
- demonstrable experience with large software projects
- commitment to individual skill development, such as learning new languages
- commitment to professional responsibility
- understand the differences between systems and application programming
- understand the hardware characteristics that dictate the requirements/features of the controlling software (e.g., sampling frequency, signal propagation delays, arithmetic precision)
To reflect trends within the CS field, the JMU CS systems core courses should strive to produce the following **characteristics of CS graduates**:

- technical understanding of computer and network systems
- appreciation of the interplay between theory and practice
- system-level perspective (thinking in levels of abstraction)
- understanding of how to identify and evaluate trade-offs in design and implementation
- ability to identify common problem patterns and apply appropriate solutions
- demonstrable experience with large software projects
- commitment to individual skill development, such as learning new languages
- commitment to professional responsibility
- understand the differences between systems and application programming
- understand the hardware characteristics that dictate the requirements/features of the controlling software (e.g., sampling frequency, signal propagation delays, arithmetic precision)
Systems core themes

In support of this goal, the following principles should guide the systems curriculum:

• open-ended programming assignments that require analysis, design, and testing
• opportunities to apply systems concepts to realistic problems
• a combination of individual- and group-based projects
• experience both reading and writing code
• exposure to standard industry tools and techniques where appropriate
• exploration of the design considerations underlying existing systems software
• emphasis on developing skills for independent learning
• flexible curriculum that serves the needs of students with varying technical talents
• appropriate coverage of fundamental computer and networking concepts
Systems core themes

In support of this goal, the following principles should guide the systems curriculum:

- open-ended programming assignments that require analysis, design, and testing
- opportunities to apply systems concepts to realistic problems
- a combination of individual- and group-based projects
- experience both reading and writing code
- exposure to standard industry tools and techniques where appropriate
- exploration of the design considerations underlying existing systems software
- emphasis on developing skills for independent learning
- flexible curriculum that serves the needs of students with varying technical talents
- appropriate coverage of fundamental computer and networking concepts
Every JMU CS student who satisfactorily completes all required systems courses should be able to:

- Summarize the technical foundations of how software executes on hardware, how parallel and distributed software communicate, what patterns and structures are used to construct systems and concurrent programs, and what layered abstractions support modern computing systems.
- Explain how bits can represent information, how instructions and communication can be seen as a sequence of state transitions, how mathematical models can describe system behavior, and how reactive programming practices used in systems programming differs from other approaches.
- Read and understand existing protocols, critically evaluate existing protocols, and select appropriate protocols.
- Describe the ways in which information can be exchanged between different parts of an existing system, select appropriate ways to exchange information in proposed systems, and implement systems that exchange information.
- Describe the architectural style/high-level design of a system using appropriate terminology, and evaluate the architectural style/high-level design of existing and proposed systems.

...
Systems core themes

Every JMU CS student who satisfactorily completes all required systems courses should be able to:

- Summarize the technical foundations of how software executes on hardware, how parallel and distributed software communicate, what patterns and structures are used to construct systems and concurrent programs, and how layered abstractions support modern computing systems.
- Explain how bits can represent information, how instructions and communication can be seen as a sequence of state transitions, how mathematical models can describe system behavior, and how reactive programming practices used in systems programming differs from other approaches.
- Read and understand existing protocols, critically evaluate existing protocols, and select appropriate protocols.
- Describe the ways in which information can be exchanged between different parts of a existing system, select appropriate ways to exchange information in proposed systems, and implement systems that exchange information.
- Describe the architectural style/high-level design of a system using appropriate terminology, and evaluate the architectural style/high-level design of existing and proposed systems.
- ...
Systems core themes

Enduring Understandings
• Information = Bits + Context
• The Semiotics of Systems
• System Design Involves Tradeoffs
• Systems Programs are a Foundation
• Digital/Discrete Models of an Analog/Continuous World
• The Computer as Other
• Appearances can be Deceiving
• From Abstract Specifications to Complete Implementations
• Communication has Multiple Dimensions
• Resources must be Shared
• Reliability can be Elusive
• Use the Right Tool for the Job
Systems core themes

Enduring Understandings
• Information = Bits + Context
• The Semiotics of Systems
• System Design Involves Tradeoffs
• Systems Programs are a Foundation
• Digital/Discrete Models of an Analog/Continuous World
• The Computer as Other
• Appearances can be Deceiving
• From Abstract Specifications to Complete Implementations
• Communication has Multiple Dimensions
• Resources must be Shared
• Reliability can be Elusive
• Use the Right Tool for the Job
Assessment strategies

- Identify situational factors
- Define objectives
- Summative assessments and metrics
- Formative assessments/instructional activities
Assessment strategies

Summative assessments

- Question bank for exams
  - Good for program assessment, too
- Common projects - 3-phase semester-long system building
  - Shared rubric encourages consistent expectations
Assessment strategies

Identify situational factors

Define objectives

Summative assessments and metrics

Formative assessments/instructional activities

Summative assessments
• Question bank for exams
  • Good for program assessment, too
• Common projects - 3-phase semester-long system building
  • Shared rubric encourages consistent expectations

Formative assessments and activities
• Primarily at instructor discretion
• Challenge: flipped vs. non-flipped classroom structures
New courses

Diagram showing the relationships between courses:
- CS 159 Adv. Prog.
- CS 227 Disc. Math
- CS 240 Data Str.
- CS 350 Comp. Org.
- CS 450 OS
- CS 457 Infosec
- CS 458 Cyber Def.
- CS 460 Networks
- CS 361 Systems II
- CS 450 OS
- CS 456 Arch.
- CS 470 Parallel/DC
- CS 458 Cyber Def.

Backward Design: An Integrated Approach to a Systems Curriculum
SIGCSE 2015 • Kirkpatrick et. al
New courses

Core Tier 1 vs. Core Tier 2

Backward Design: An Integrated Approach to a Systems Curriculum
SIGCSE 2015 • Kirkpatrick et. al
New courses

CS 261 Computer Systems I (required)
Introduction to operation of modern interrupt-driven computer systems. Explores the representation of software and information in binary memory, the primary components of a CPU, multithreaded programming, and basic interactions with an Operating System. Prerequisites: Grade of “C-” or better in CS 159.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C basics</td>
<td>6</td>
<td>Memory model, pointers</td>
</tr>
<tr>
<td>Compiler and debugger use</td>
<td>3</td>
<td>GCC/clang, Makefiles, GDB</td>
</tr>
<tr>
<td>CPU/memory organization</td>
<td>3</td>
<td>Registers and cache, locality</td>
</tr>
<tr>
<td>Binary representation</td>
<td>4.5</td>
<td>Two’s complement, IEEE 754, arithmetic encoding</td>
</tr>
<tr>
<td>von Neumann cycle</td>
<td>3</td>
<td>Role of CPU and memory, load/store instructions</td>
</tr>
<tr>
<td>Basic circuits</td>
<td>6</td>
<td>Logic gates, adders, ALUs, control signals</td>
</tr>
<tr>
<td>Threads vs. processes</td>
<td>6</td>
<td>Fork vs. thread creation, unique memory space</td>
</tr>
<tr>
<td>Interrupts and OS principles</td>
<td>4.5</td>
<td>Interrupts, system calls, user vs. kernel mode</td>
</tr>
<tr>
<td>System software design, evaluation</td>
<td>4.5</td>
<td>Benchmarks, complexity, static/dynamic analysis</td>
</tr>
</tbody>
</table>
New courses

**CS 361 Computer Systems II (required)**
Intermediate exploration of modern interrupt-driven computer systems. Explores models of computation and complex systems, techniques for communication and synchronization of parallel and concurrent software, and the protocols that make up the Internet. Prerequisites: Grades of “C-” or better in CS 240 and CS 261.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture analysis, evaluation, design</td>
<td>3</td>
<td>P2P vs. client-server, layered architecture</td>
</tr>
<tr>
<td>State models</td>
<td>4.5</td>
<td>Notion of state, UML, FSM</td>
</tr>
<tr>
<td>Mathematical modeling</td>
<td>3</td>
<td>Basic systems theory</td>
</tr>
<tr>
<td>Information exchange</td>
<td>6</td>
<td>Communication basics (blocking vs. non-blocking, IPC vs. sockets)</td>
</tr>
<tr>
<td>Synchronization primitives and problems</td>
<td>6</td>
<td>Locks vs. semaphores, producer-consumer, readers-writers, dining philosophers, deadlock</td>
</tr>
<tr>
<td>Parallel decomposition</td>
<td>3</td>
<td>Data vs. task parallelism, Amdahl’s law, fork-join pattern, libraries</td>
</tr>
<tr>
<td>Protocol analysis, evaluation, design</td>
<td>9</td>
<td>Protocols and services, timing and statechart diagrams, connections, push/pull, flow control, reliability, handshaking, metrics</td>
</tr>
<tr>
<td>The Internet model</td>
<td>4.5</td>
<td>HTTP, DNS, DHCP, TCP, UDP, IP, 802.3, 802.11, ARP</td>
</tr>
</tbody>
</table>
# New courses

## CS 450 Operating Systems

Introduction to the design and implementation of modern operating systems. Explores fundamental concepts of operating systems, memory management, virtualization, resource allocation, file systems, and system protection mechanisms. Course work includes a significant programming component. Prerequisites: Grade of “C-” or better in CS 361.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread and process management</td>
<td>4.5</td>
<td>Review of multithreading, OS structures for representing threads and processes, context switches</td>
</tr>
<tr>
<td>OS interface and IPC</td>
<td>4.5</td>
<td>Review of system calls vs. interrupts, pipes, shared memory, other forms of IPC</td>
</tr>
<tr>
<td>Synchronization implementation</td>
<td>6</td>
<td>Hardware support for implementation of semaphores, locks, spinlocks</td>
</tr>
<tr>
<td>Memory management</td>
<td>6</td>
<td>Paging vs. segmentation, virtual memory, demand paging, implementation of shared memory</td>
</tr>
<tr>
<td>Virtualization</td>
<td>6</td>
<td>Virtualization vs. emulation, trap-and-emulate, binary translation, hardware support for virtualization</td>
</tr>
<tr>
<td>Scheduling</td>
<td>3</td>
<td>Scheduling policies and evaluation</td>
</tr>
<tr>
<td>I/O and file systems</td>
<td>6</td>
<td>Interrupt-driven I/O, DMA, RAID, file system implementation and metadata</td>
</tr>
<tr>
<td>Security and protection</td>
<td>6</td>
<td>CIA model, access control mechanisms, malware defense mechanisms</td>
</tr>
</tbody>
</table>
New courses

CS 456 Computer Architecture
Introduction to the design and implementation of modern CPU architectures. Explores hardware-based parallel execution, quantitative performance evaluation, I/O interfacing techniques, and hardware descriptor languages. Course work includes a significant programming component. Prerequisites: Grade of “C-” or better in CS 261.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly language</td>
<td>6</td>
<td>RISC assembly language and decoding</td>
</tr>
<tr>
<td>Building a datapath</td>
<td>6</td>
<td>Logic gates, control unit, ALU construction, register banks, von Neumann implementation</td>
</tr>
<tr>
<td>Hardware descriptor languages</td>
<td>3</td>
<td>Verilog, VHDL, RTL</td>
</tr>
<tr>
<td>Pipelined datapath and hazards</td>
<td>6</td>
<td>Pipelined datapath and control, data hazards (forwarding vs. stalling), control hazards, exceptions</td>
</tr>
<tr>
<td>Memory hierarchy and cache design</td>
<td>6</td>
<td>Quantitative performance measures, cache mapping techniques, cache coherence protocols</td>
</tr>
<tr>
<td>Storage and I/O interfacing</td>
<td>4.5</td>
<td>Storage devices, bus protocols, I/O performance</td>
</tr>
<tr>
<td>Instruction-level parallelism</td>
<td>4.5</td>
<td>Branch prediction, dynamic scheduling</td>
</tr>
<tr>
<td>Data-level parallel architectures</td>
<td>3</td>
<td>Vector, SIMD, GPU architectures</td>
</tr>
<tr>
<td>Thread-level parallel techniques</td>
<td>3</td>
<td>Hyperthreading, shared-memory multiprocessors</td>
</tr>
</tbody>
</table>
New courses

**CS 470 Parallel and Distributed Systems**

Introduction to parallel and distributed systems. Explores shared memory, cluster, grid, peer-to-peer, and cloud computing models along with parallel software patterns, distributed file systems, and performance considerations. Course work includes a significant programming component. Prerequisites: Grade of “C-” or better in CS 361.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel/distributed concepts</td>
<td>3</td>
<td>Amdahl’s law, critical paths, speedup/scalability, data/task decomposition, applications, research challenges</td>
</tr>
<tr>
<td>Parallel patterns</td>
<td>6</td>
<td>Naturally (embarrassingly) parallel, nearest-neighbor, communication, producer-consumer, master-workers, pipelines, map/reduce</td>
</tr>
<tr>
<td>Parallel systems</td>
<td>9</td>
<td>Shared memory, SMP, SIMD, OpenMP, GPUs/co-processors, race conditions, mutual exclusion, deadlock, cache effects, dense/sparse matrices</td>
</tr>
<tr>
<td>Distributed systems</td>
<td>9</td>
<td>MPI, MapReduce, global address spaces, clusters, topologies, synchronization, collectives, clocks, NUMA, hybrid architectures, fault tolerance</td>
</tr>
<tr>
<td>Grid, P2P, and cloud computing systems</td>
<td>6</td>
<td>Heterogeneous systems, decentralized computation, consensus, IaaS, virtualization</td>
</tr>
<tr>
<td>Distributed file systems</td>
<td>6</td>
<td>RPC, data replication, transactions, consistency</td>
</tr>
<tr>
<td>Parallel performance</td>
<td>3</td>
<td>Tools, measurement, scheduling/load balancing, contention, communication overhead, power usage</td>
</tr>
</tbody>
</table>
Possible future courses

**Breadth-first scaffolding as foundation for:**
- Advanced networking
- Database design and implementation
- Virtualization technologies
## ACM 2013 Mapping

### ACM 2013 KA | Current | Proposed
---|---|---
**Algorithms and Complexity**
- Distributed algorithms (T1) | none | 470

**Architecture and Organization**
- Machine-level representation of data (T2) | 350 | 261, 456
- Assembly level machine organization (T2) | 350 | 261, 456
- Interfacing and I/O strategies (interrupts) (T2) | 350 | 456
- Memory architecture (T2) | 350 | 261, 456
- Functional organization (ILP/datapaths) (E) | 350 (some) | 261, 456

**Operating Systems**
- OS overview (T1) | 450 | 261
- OS principles (APIs, processes, interrupts) (T1) | 450 | 261, 456
- Concurrency (T2) | 450 | 361, 450
- Scheduling (T2) | 450 | 450
- Memory management (T2) | 450 | 450
- Security and protection (T2) | 450 | 457/453
- File systems (E) | 450 (some) | 450
- System performance evaluation (E) | none | 261

**Network-centric Computing**
- Introduction (ISPs, circuit vs. packet) (T1) | 460 | 361
- Network applications (HTTP, sockets) (T1) | 460 (most) | 361, 466, 470
- Reliable data delivery (TCP, flow control) (T2) | 460 | 361, 466
- Routing vs. forwarding (T2) | 460 | 361, 466
- LANs (T2) | 460 | 361, 466
- Resource allocation (T2) | 460 | 361, 466
- Mobility (T2) | 460 | 361, 466

### ACM 2013 KA | Current | Proposed
---|---|---
**Parallel and Distributed Computing**
- Parallelism fundamentals (T1) | 450 (most) | 261, 470
- Parallel decomposition (T1,T2) | none | 361, 470
- Communication and coordination (T1,T2) | 450 (some) | 361, 450, 470
- Parallel architectures (T1,T2) | 350 (little) | 261, 470
- Parallel performance (E) | none | 470
- Distributed systems (E) | none | 361, 470

**System Fundamentals**
- Computational paradigms (T1) | 350 (some) | 261, 361, 456, 470
- Cross-layer communications (T1) | 450 (little) | 361, 450, 466, 470
- States, transitions, state machines (T1) | 350 (some) | 361, 456, 466, 470
- System support for parallelism (T1) | 350 (little) | 361, 456, 466, 470
- Performance (T1) | none | 261
- Resource allocation and scheduling (T2) | 450 | 450
- Proximity (T2) | 350 | 450, 456, 470
- Virtualization (T2) | none | 450
- Reliability through redundancy (T2) | 460 (little) | 361, 456, 470
Transitional plan
Transitional plan

**Fall 2015**

- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)
Transitional plan

**Fall 2015**
- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)

**Spring 2016**
- CS 350 replaced by CS 261
- First offering of CS 470 (Parallel and Distributed)
Transitional plan

**Fall 2015**
- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)

**Spring 2016**
- CS 350 replaced by CS 261
- First offering of CS 470 (Parallel and Distributed)

**Fall 2016**
- New curriculum offering fully in place
  - CS 261 and CS 361 now offered every semester
  - First offering of CS 456 (Architecture)
Lessons learned

Too many to name...
- A well-defined vision statement works wonders
  - Revisit frequently, particularly for conflict resolution
- SMART objectives make writing assessments easier
  - Take the time to do this well
- Transition to new curriculum requires improvisation and flexibility
- Build your own taxonomy
  - Fink and Wiggins/McTighe for aspirational structure
  - Hansen for enumerating points of emphasis
  - Bloom and SOLO for instructional objectives
- The process will take longer than you expect
  - Hidden assumptions, connections, requirements
https://github.com/kirkpams/JMU-CS-Systems