CIS*3750 - System Analysis and Design in Applications

Luiza Antonie, Fall 2025, University of Guelph

ML production myths

Myth #1: Deploying is hard

Myth #1: Deploying is hard

Deploying is easy. Deploying reliably is hard

Myth #2: You only deploy one or two ML models at a time

Myth #2: You only deploy one or two ML models at a time

Booking.com: 150+ models, Uber: thousands

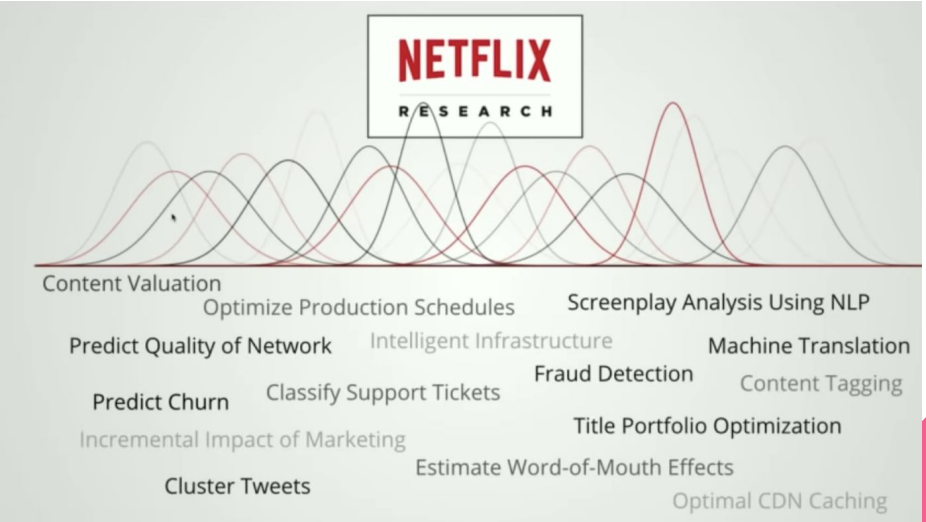


Image from Ville Tuulos (Netflix, Outerbounds)

Slide based on Designing Machine Learning Systems textbook and https://cs329s.stanford.edu course

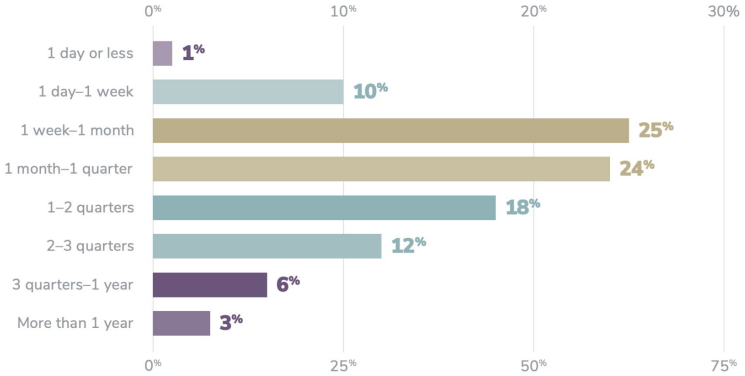
Myth #3: You won't need to update your models as much

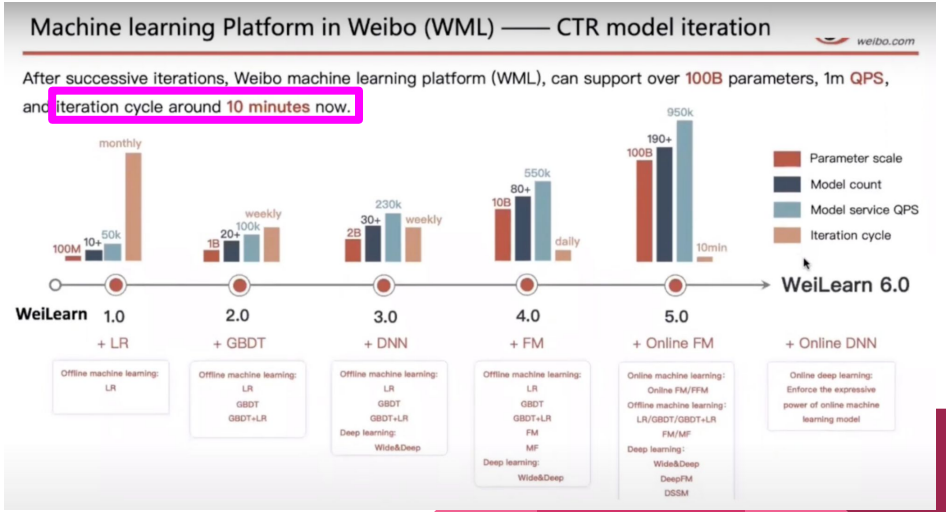
DevOps: Pace of software delivery is accelerating

- Elite performers deploy **973x** more frequently with **6570x** faster lead time to deploy (Google DevOps Report, 2021)
- DevOps standard (2015)
  - Etsy deployed 50 times/day
  - Netflix 1000s times/day
  - AWS every 11.7 seconds

DevOps to MLOps: Slow vs. Fast

Only 11% of organizations can put a model into production within a week, and 64% take a month or longer





Left image from Algorithmia | Right image: Machine learning with Flink in Weibo (Qian Yu, QCon 2019)

Slide based on Designing Machine Learning Systems textbook and https://cs329s.stanford.edu course

Accelerating ML Delivery

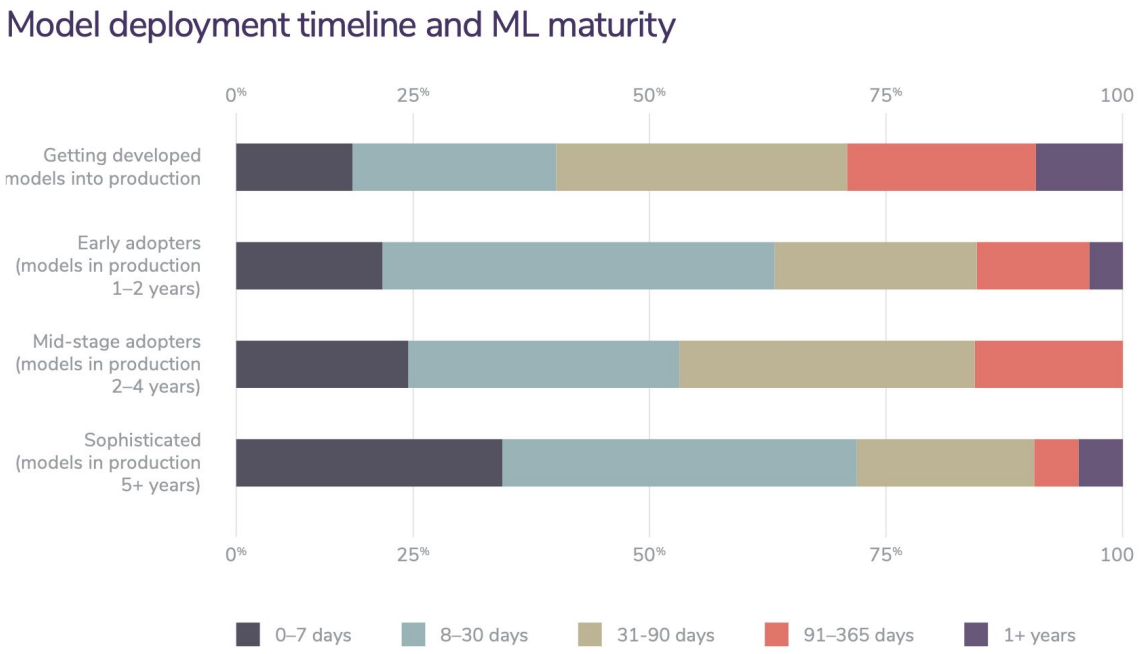
ML + DevOps = 

Myth #4: ML can magically transform your business overnight

Myth #4: ML can magically transform your business overnight

Magically: possible
Overnight: no

Efficiency improves with maturity



2020 state of enterprise machine learning

Slide based on Designing Machine Learning Systems textbook and https://cs329s.stanford.edu course

ML engineering is more engineering than ML

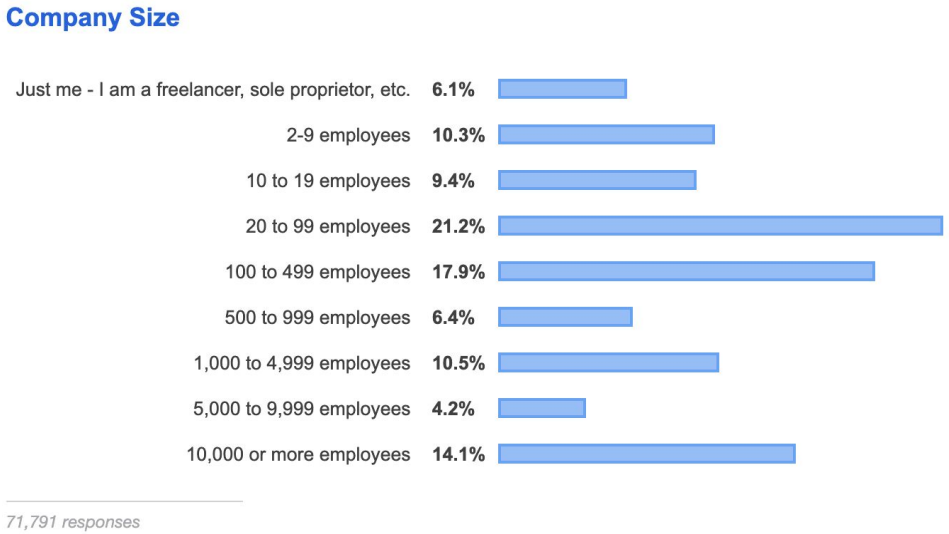
MLEs might spend most of their time:

- wrangling data
- understanding data
- setting up infrastructure
- deploying models

instead of training ML models

Myth #5: Most ML engineers don’t need to worry about scale

Myth #5: Most ML engineers don't need to worry about scale

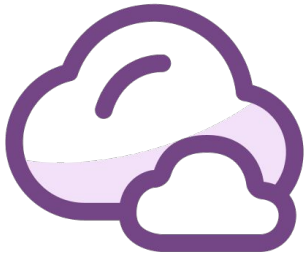


StackOverflow Developer Survey 2019

Slide based on Designing Machine Learning Systems textbook and https://cs329s.stanford.edu course

Activity (10 - 15 min)

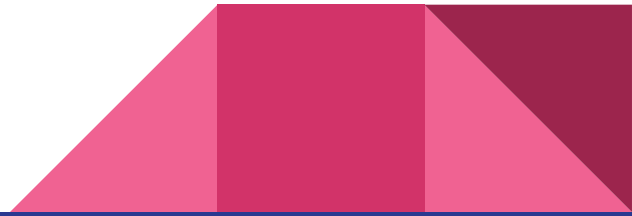
Form small groups (4-6 people) and discuss what skills you think are needed to produce good software products



What skills do you think are needed to produce good software products?

How do we build good software?

- Good communication and research skills to produce good requirements
- Good planning and problem solving skills to come up with a good design
- Good coding skills to implement your solution



How do we build good software?

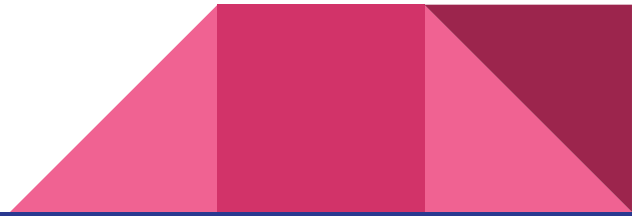
C**ommunication** is essential:

- within the team
- between the teams
- with the client

NEVER ASSUME YOU KNOW ANYTHING!

Teamwork

- Personal conduct and social contract with your teammates
- Building ‘Organizational’ culture around making decisions, resolving disputes and more
- Taking responsibility for work and being aware of the progress of others



Teamwork

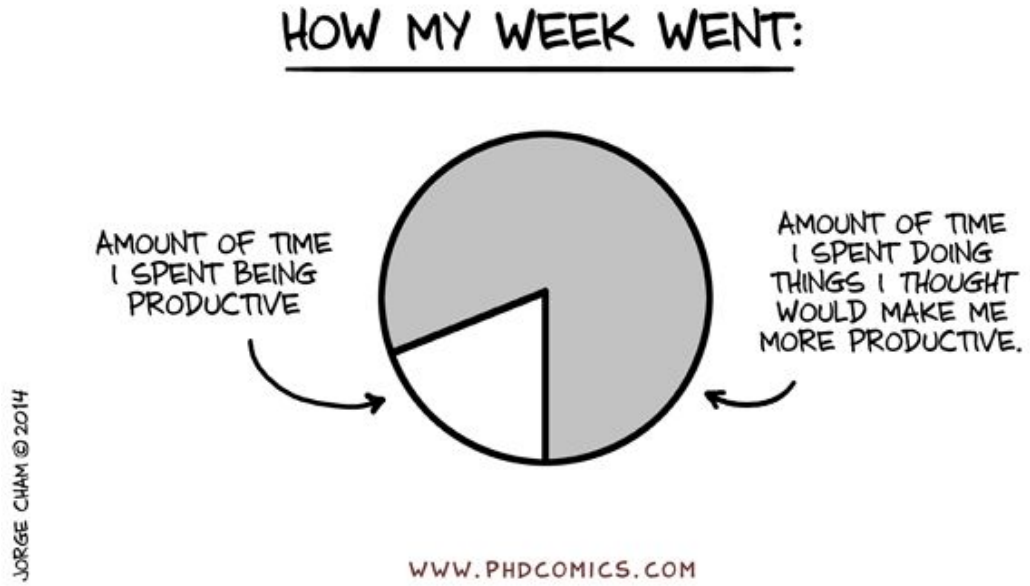
- Elevate the work of those around you
- When you finish a task, ask your teammates if they need help
- Look for opportunities to delegate tasks when you are *TOO* busy.

Teamwork

- Try to take on tasks or help out when you can
- Look hard for the strengths of others and how they might best contribute
- Be respectful when raising perceived weaknesses

Working Effectively

Not all hours are equal, make the most of the time you dedicate to your project!



Working Effectively

- Complex problems require your full attention, otherwise errors creep into your work

- “Finding and fixing a software problem after delivery can be upwards of 100 times more expensive than finding it and fixing it during the requirements and early design phases.” Error Cost Escalation Through the Project Life Cycle ntrs.nasa.gov, 2004

Teams and Communication

- The goal isn’t to talk to each other, discussions must be purposeful
- Your group should have a well defined reason for meeting
- Good meetings have an agenda, a facilitator and a note taker.

Teams and Communication

- The role of the facilitator is to manage the meeting agenda, not to be in charge
- The importance of clearly recording decisions so they can be referenced later is crucial
- What were the options, why was the decision made?

Teamwork

- How did you manage group dynamics?

- What decisions influenced your designs, and why?

Project: Building an intelligent tutoring system

- Personalized learning with AI and ML

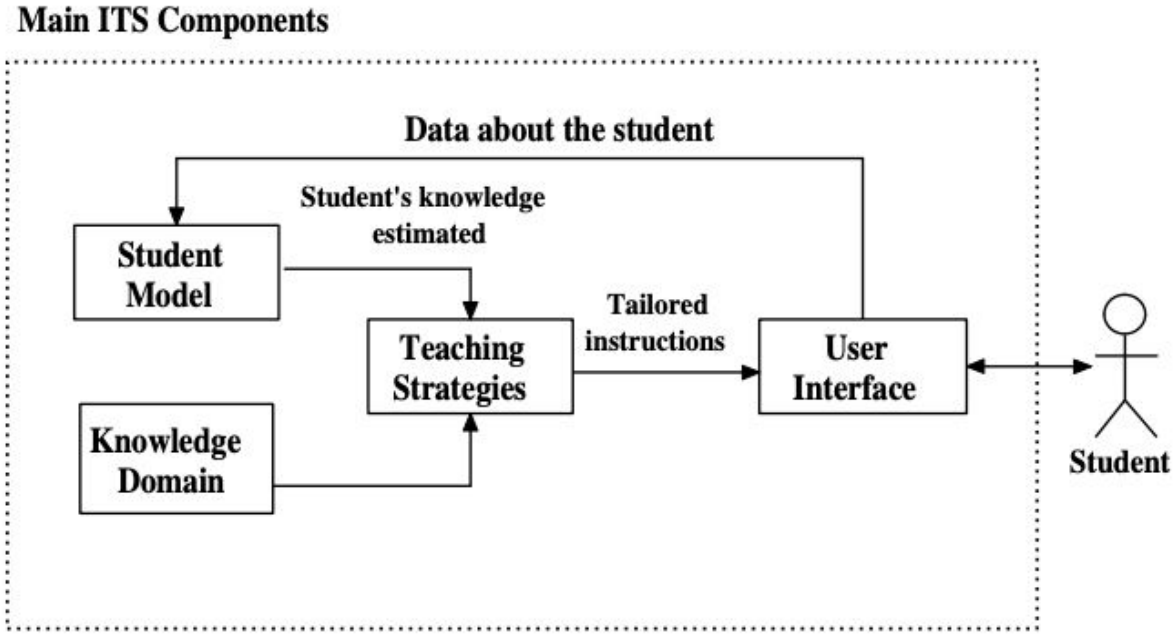


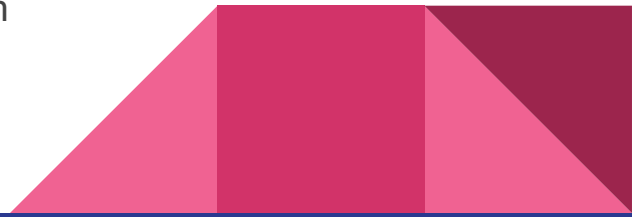
Image source: https://www2.cs.uregina.ca/~butz/publications/wais06.pdf

What is an Intelligent Tutoring System (ITS)?

- Intelligent tutoring systems are educational software that teach concepts, give feedback and adapt to each learner’s needs using data-driven methods, often powered by machine learning.

What is an Intelligent Tutoring System (ITS)?

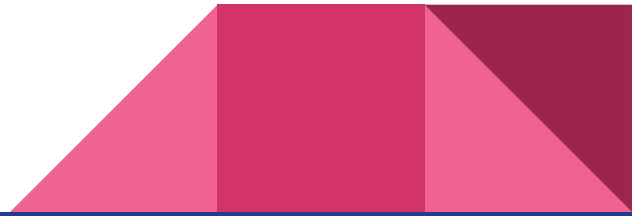
- The domain model
  - knowledge and skills to be taught
  - Instructional content
- The student model
  - tracks the learner's progress
- The tutoring model
  - determines the most effective teaching strategies based on data from the domain and student models.
- The user interface model
  - facilitates the interaction between the student and the system



The Domain Model

- Instructional content (e.g., algebra, Python programming)

- Knowledge representations
  - Vector representations
  - Graphs
  - Rules
  - Hierarchies/ ontologies



The Domain Model

- Concepts:
  - addition/subtraction, multiplication/division, solving linear equations
- Dependencies:
  - Addition → Multiplication → Solving linear equations
- Mapping exercises/problems:
  - “Solve 3x + 2 = 11” → tagged with “linear equations” (depends on prior arithmetic skills)
- Solutions

ML in the Domain Model

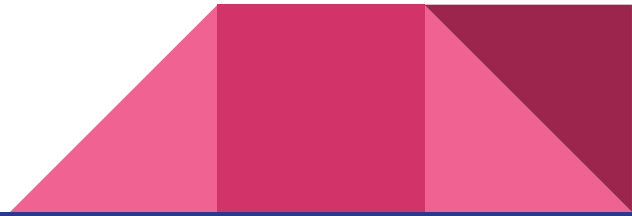
- Automatic skill discovery
  - group questions by difficulty/skill
- Difficulty estimation
  - predicting difficulty of a concept/problem
- Knowledge tracing
  - estimate mastery of concepts

For Your Project

- Pick a domain (e.g., algebra, Python programming, statistics, English).
- Define skills/concepts.
- Tag exercises/problems with these concepts.

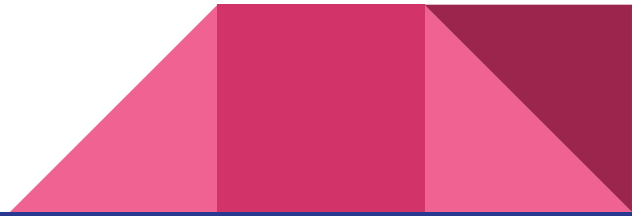
The Student Model

- A representation of the learner
- Tracks what a student knows or struggles with
- A profile that evolves over time as the student interacts with the ITS.



The Student Model

- Information gathered and stored
  - Knowledge state: mastery of specific concepts/skills.
  - Performance history: attempts, correctness, speed, hints requested.
  - Affective states (sometimes): engagement, frustration, confidence.
  - Learning style/preferences: if the system supports multiple modalities.



ML in the Student Model

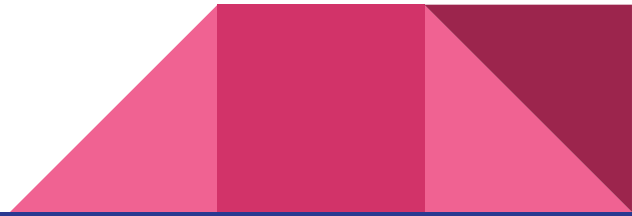
- **Prediction**: Will the student answer the next item correctly?
- **Classification**: Detect misconceptions or error types.
- **Recommendation**: Which skill should the student practice next?
- **Clustering**: Group students by learning behaviors (fast/slow, high/low engagement).

For Your Project

- What data does your ITS collect?
  - answers
  - time spent on problem
  - hints used

- How does the system update the student model after each interaction?

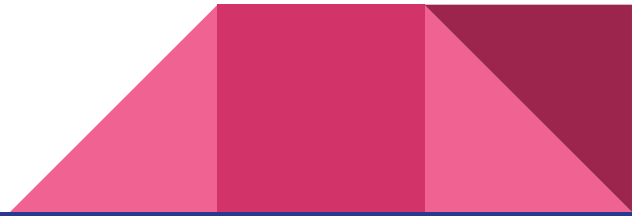
- What ML will you consider?



The Tutoring Model

- The “teacher” inside the ITS
- Makes the system interactive and adaptive, like a real tutor who decides how to teach.

- Uses information from the student model and domain model to decide what to do next
  - Which exercise to suggest next
  - Whether to provide a hint or explanation
  - When to move on to new material



ML in the Tutoring Model

Policy learning: Use reinforcement learning (RL) to decide the “best” next action for learning gains.

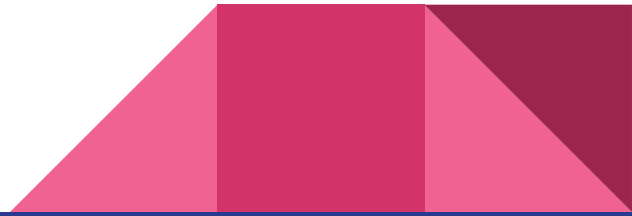
Dialogue systems: NLP-based tutors that hold conversations (e.g., chatbots)

Feedback personalization: Predict which type of explanation works best for a student.

Adaptive hinting: Classify the student’s state and select the most helpful hint level.

For Your Project

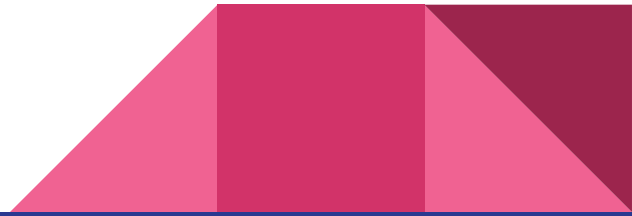
- What kinds of hints or feedback will your system provide?
- When to move a student forward vs. when to review material.
- Whether to use a simple rules-based tutor (if X errors → give hint) or explore ML for adaptation.



The UI model

- The communication layer between the learner and the system.

- How students see content, input responses, receive feedback, and interact with the tutor.



The UI Model

- Presenting content: problems, hints, explanations, visualizations.
- Capturing input: multiple-choice, free text, speech, code, drag-and-drop.
- Feedback display: correctness, hints, worked examples.
- Progress tracking: dashboards, mastery meters, badges, streaks.
- Interaction style: linear lesson flow, chatbot-style dialogue, gamified interface.

ML for UI Model

- Adaptive feedback presentation: choose whether to show a hint as text, a diagram, or a worked example depending on the learner’s history.
- Affective computing: detect frustration/engagement from clicks, pauses, give motivational messages.
- Personalization: UI themes, difficulty level sliders, progress visualizations.

For Your Project

- How will students interact with the ITS (text, multiple choice, chatbot, etc.)?
- How will the system present feedback and hints?
- What progress indicators (dashboards, badges, levels) will be shown?
- How will the design balance simplicity vs. engagement

ITS Examples

- MATHia - Carnegie Learning math tutor
- Duolingo - language education
- ALEKS - math, chemistry, statistics and more