Modeling Training Efficiency and Return on Investment in GIFT: Part 2, GIFT Integration

James Niehaus[1], Gregory A. Goodwin[2], & Kenny Lu[1]
Charles River Analytics[1], U.S. Army Research Laboratory[2]

GIFTSym 6, Orlando Fl
May 10, 2018

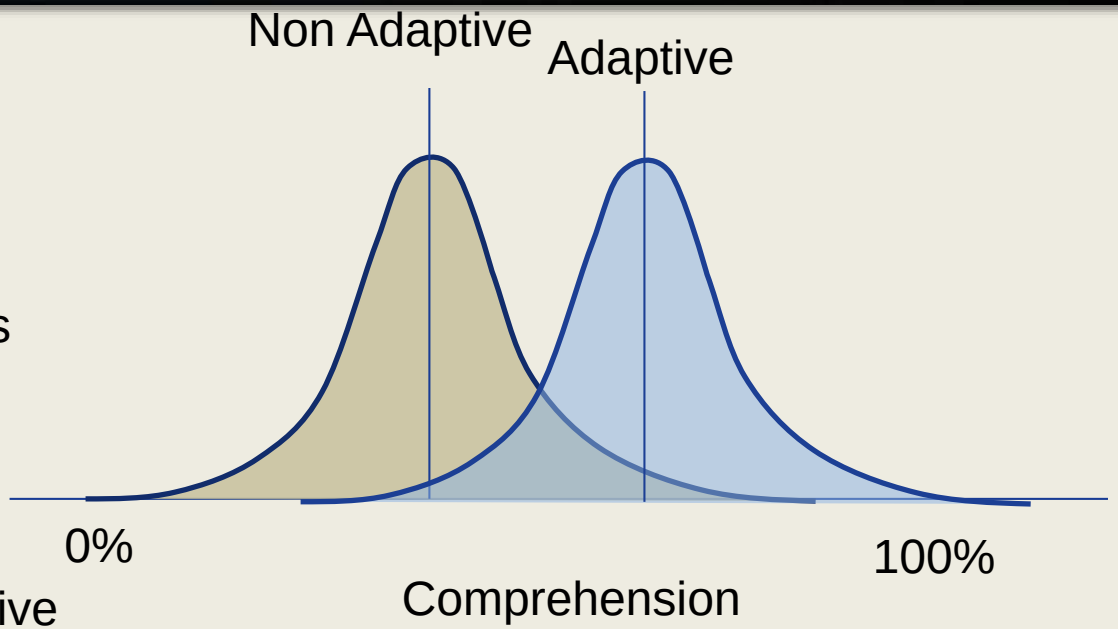
Overview

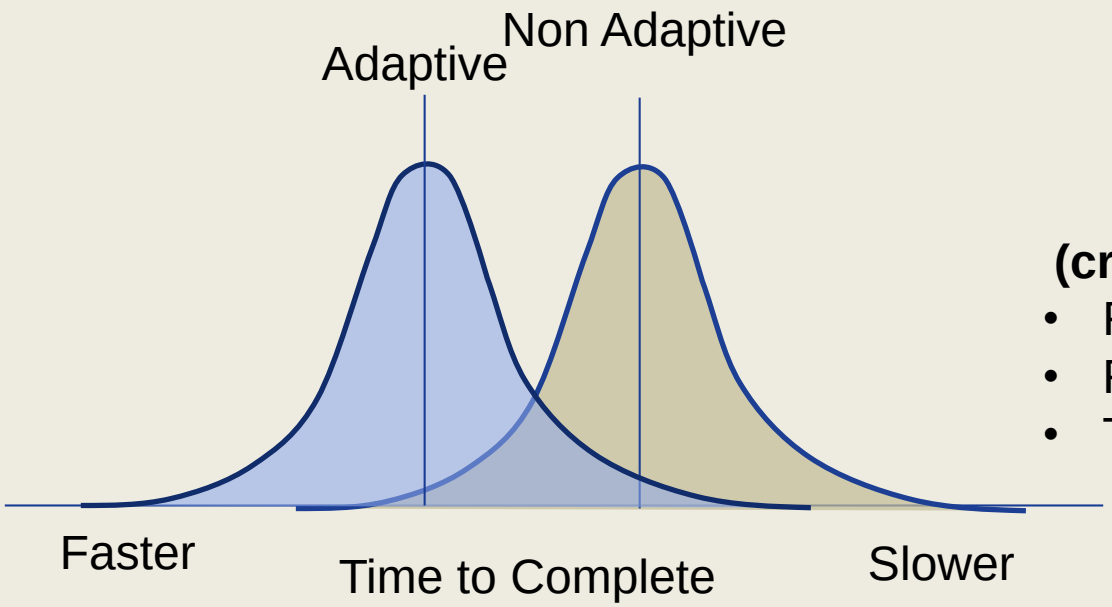
- Probabilistic Programming for Anticipated Simulation Training (PAST) Time
- Benefits of adaptive training
- Modeling time to train in GIFT
- Integrating with GIFT
- Current implementations
- Conclusions

Benefits of Adaptive Training

Training Effectiveness (time held constant)

- Accurate diagnosis of errors
- Targeted remediation
- Tailored training methods





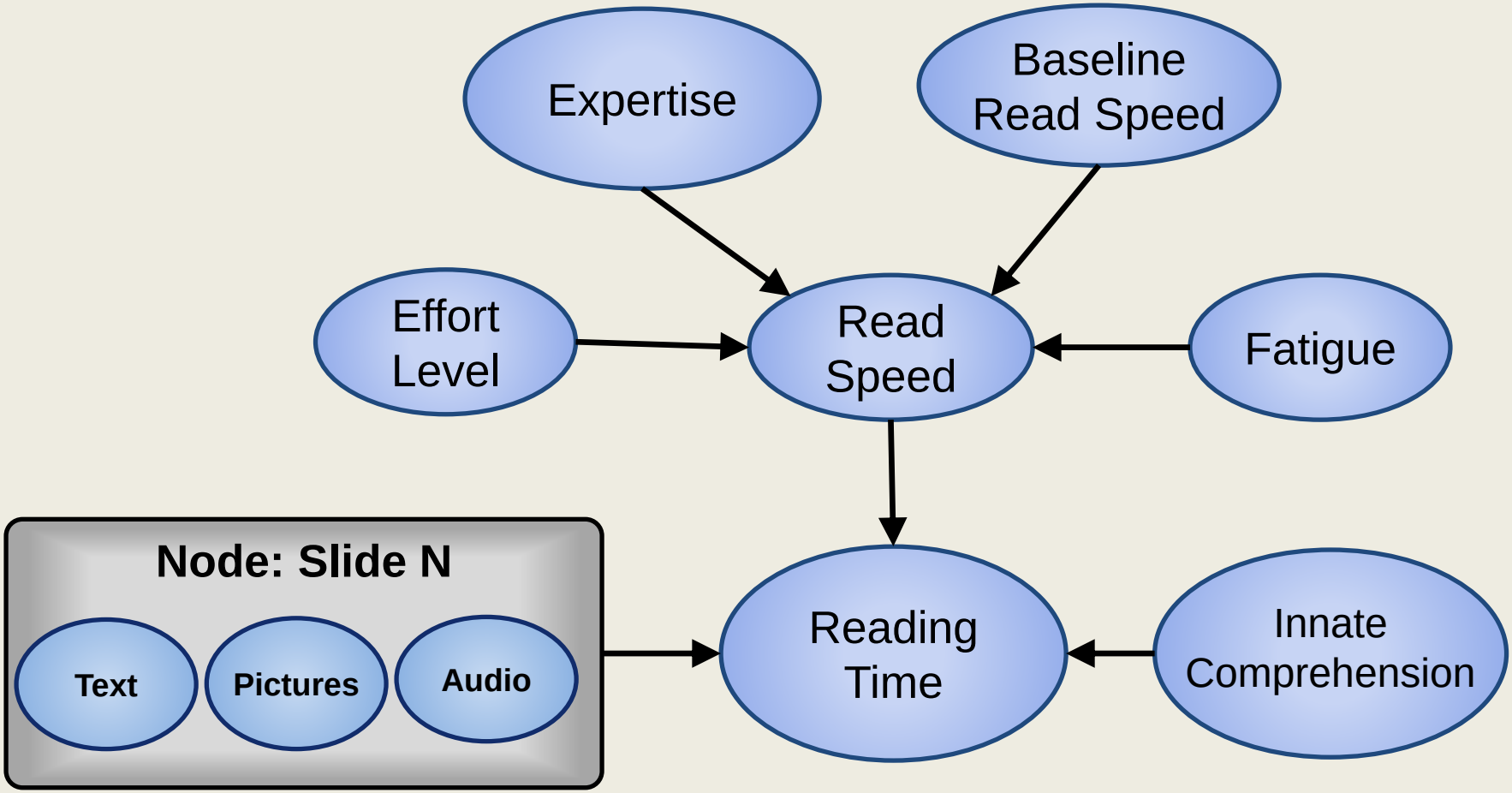
Training Efficiency (criterion held constant)

- Rapid diagnosis of errors
- Rapid remediation
- Tailored training content

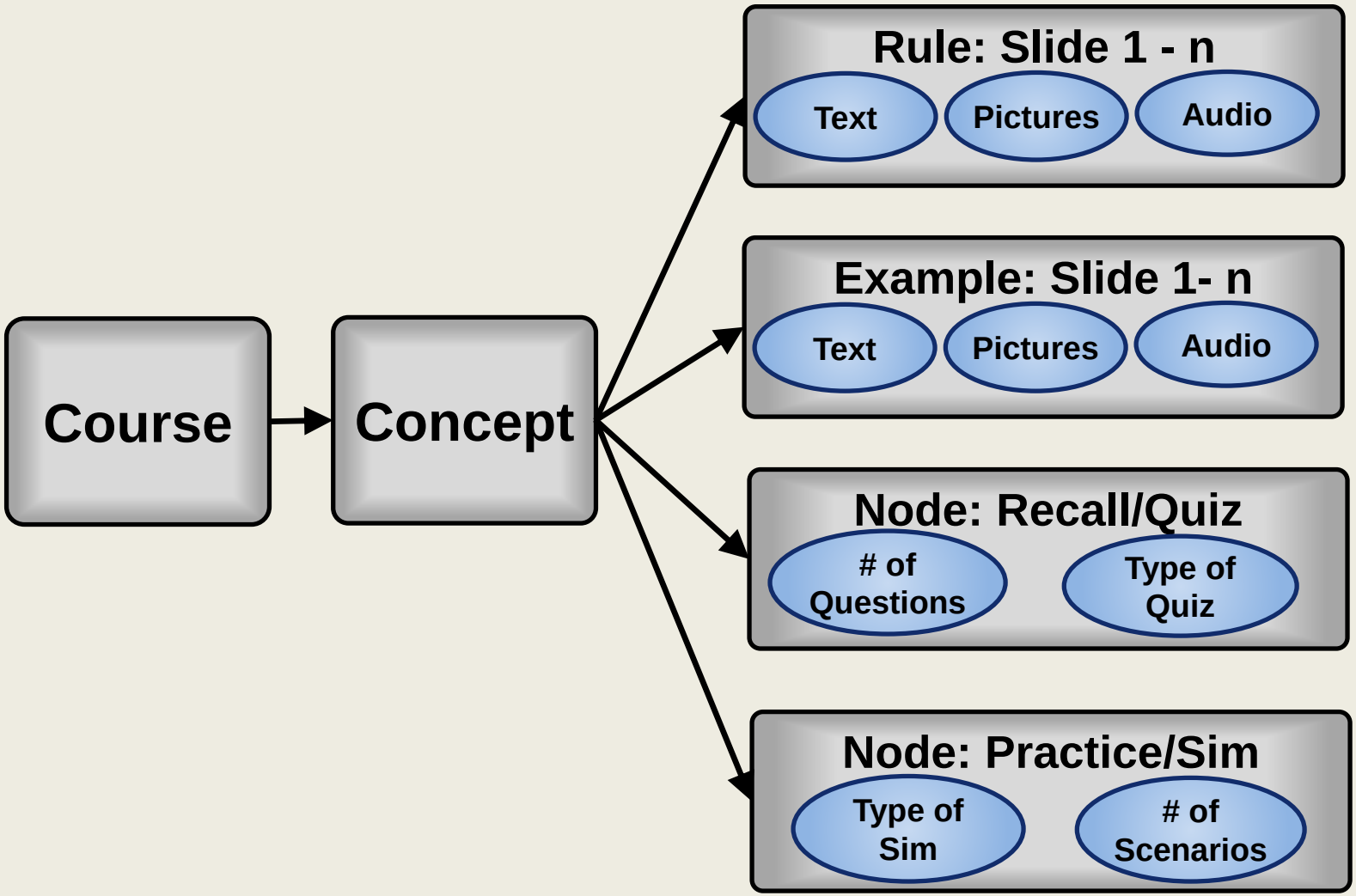
Factors that Impact Training Time

- **Learner Factors**: Aptitude, reading speed, reading comprehension level, prior knowledge & experience
- **Content Factors**: Number of words, number of images, content difficulty, test characteristics, etc.
- **Instructional Factors**: training methods and techniques

Simple Model for Time to Read



Representation of Course Content and Factors that Influence Completion Time

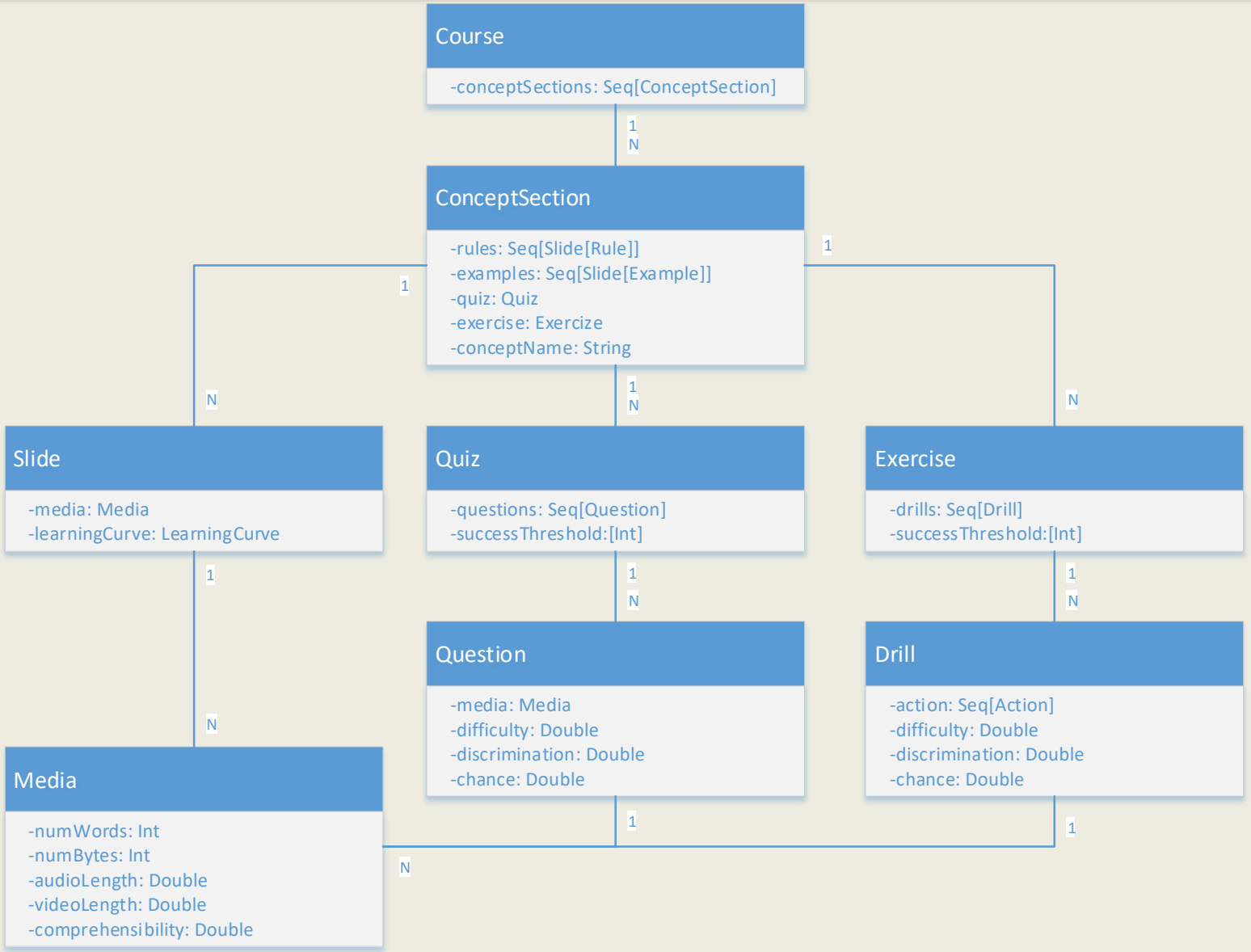


Initial Student Model

Student

-fatigue: Double
-readSpeed: Double
-expertise: Map[Concept, Double]
-effortLevel: Double
-innateComprehension: Double
-repetitions: Map[Drill, Int]
-numFailures: Int

Merrill GIFT Course Model



Figaro Slide Reading Time

```
def readSlide[T <: TrainingElement](slide: Slide[T], concept: Concept, student:
Student): (Student, Element[Double]) =
{
  Estimate reading time   mediaIngestionTime(slide.media, student)
  Update Fatigue          Math.min(1, Math.max(1.005*student.fatigue, 0.00000001))
  Update Expertise        student.expertise |+| Map(concept ->
        expertiseIncrease[T](slide, concept, student))
        (student.copy(fatigue = newFatigue,
                      expertise = newExpertise),
        readingTime)
}
```

Figaro Quiz Taking Time Model

```
def takeQuiz(quiz: Quiz, concept: Concept, student: Student): (Student,
Element[Double], Element[Boolean]) = {

                lement[Boolean]] = quiz.questions.map(q => probOfSuccess(q,
                student))

                Container(probs: _*)

                e((x: Double, y: Double) => x+y)(quiz.questions.map{q =>
                mediaIngestionTime(q.media, student)}: _*) // thinking time

                Math.min(1, Math.max(1.05*student.fatigue, 0.00001))

                tigue = newFatigue), readingTime, questions.count(x => x).map(_ >=
                quiz.successThreshold))
}
```

Pass Quiz Likelihood

Estimate reading time

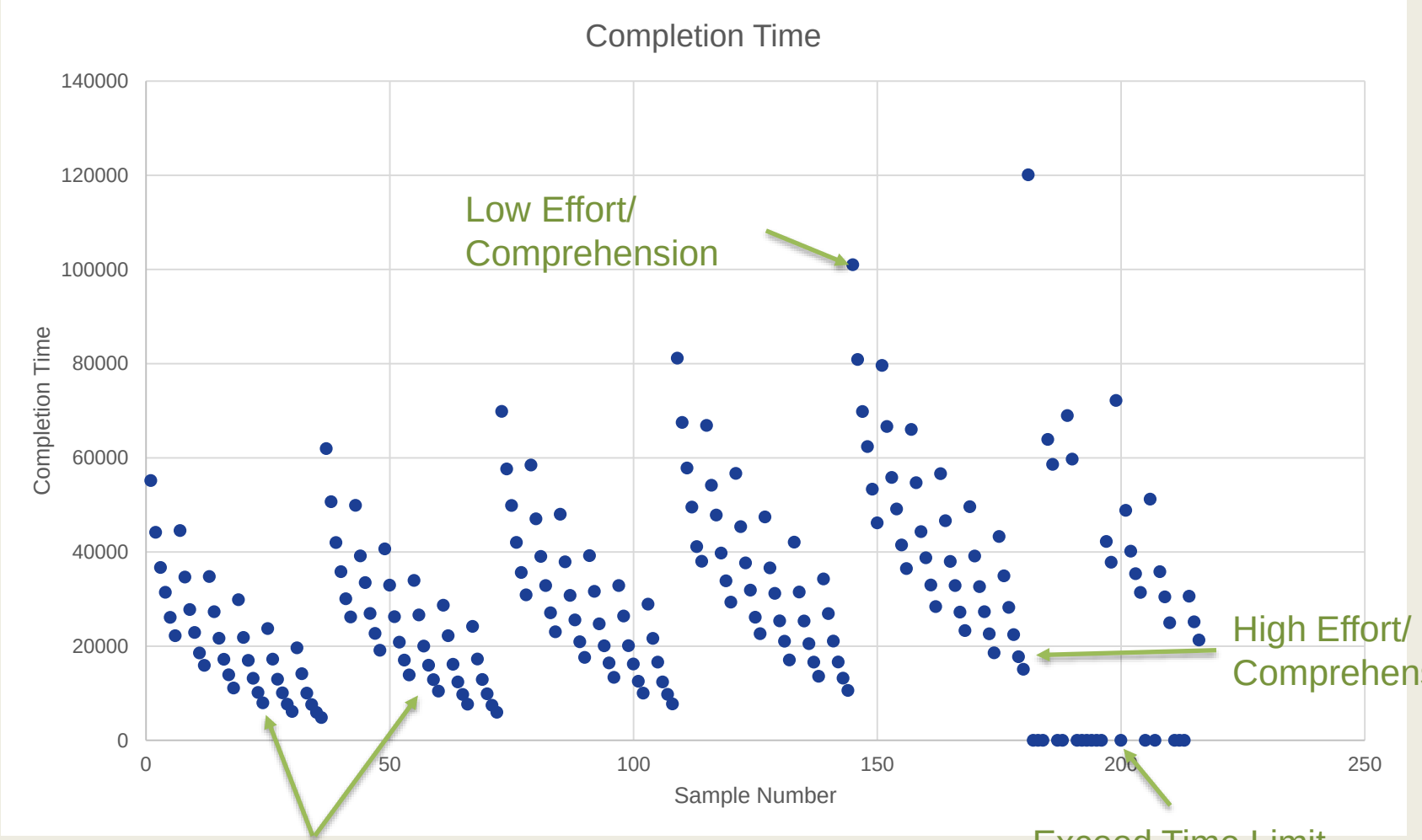
Update Fatigue

Update Student

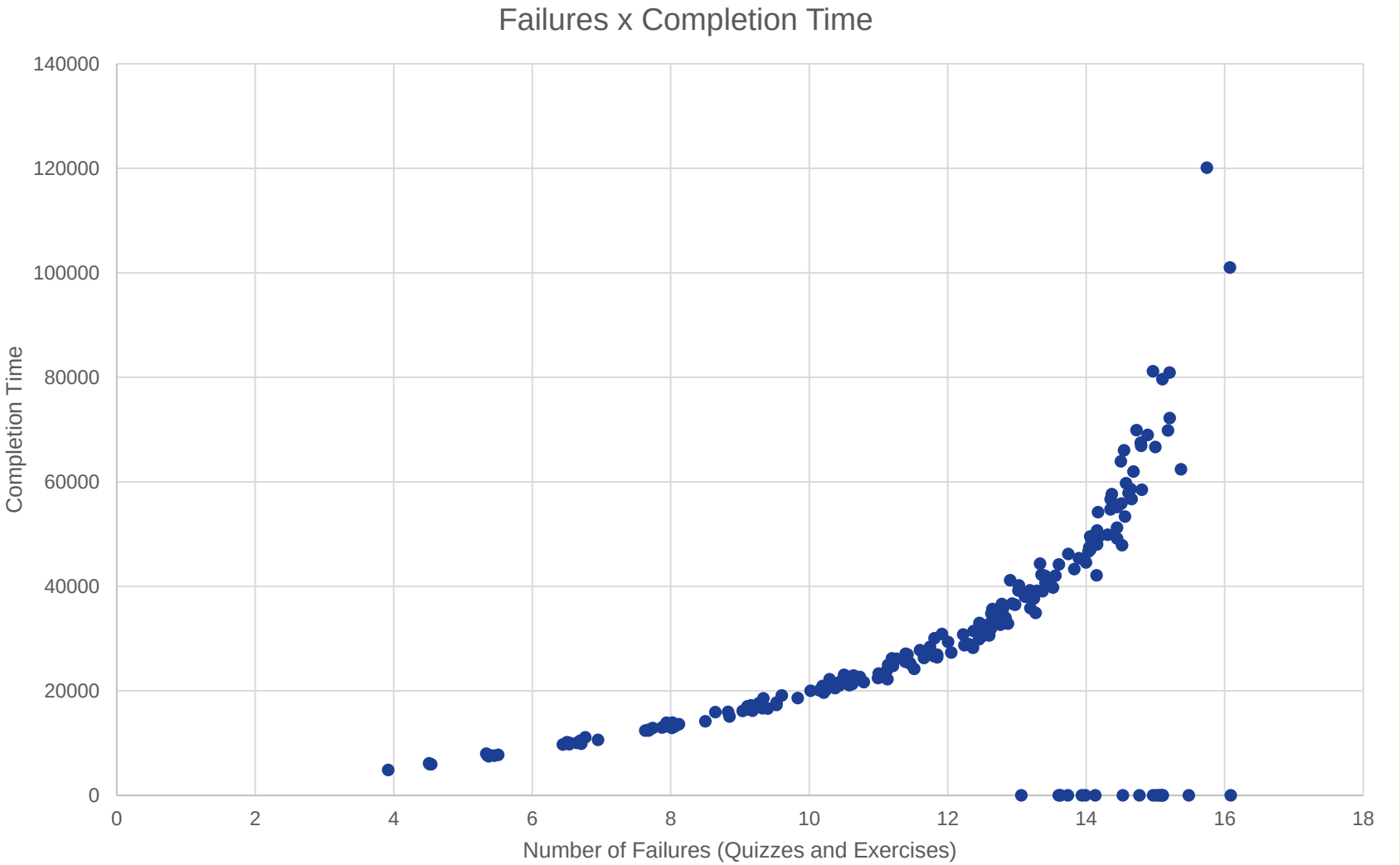
Sample Students

```
object StudentGenerator{
  def generateStudents: Seq[Student] = {
    for{


      fatigue <- 0.0 to 0.5 by 0.1
      innateComp <- 0.5 to 1 by 0.1
      effortLvl <- 0.5 to 1 by 0.1
      readSpeed = 300


    }yield{
      Student(fatigue,
                Constant(readSpeed),
                Map(),
                effortLvl,
                innateComp,
                Map())
    }}}
```

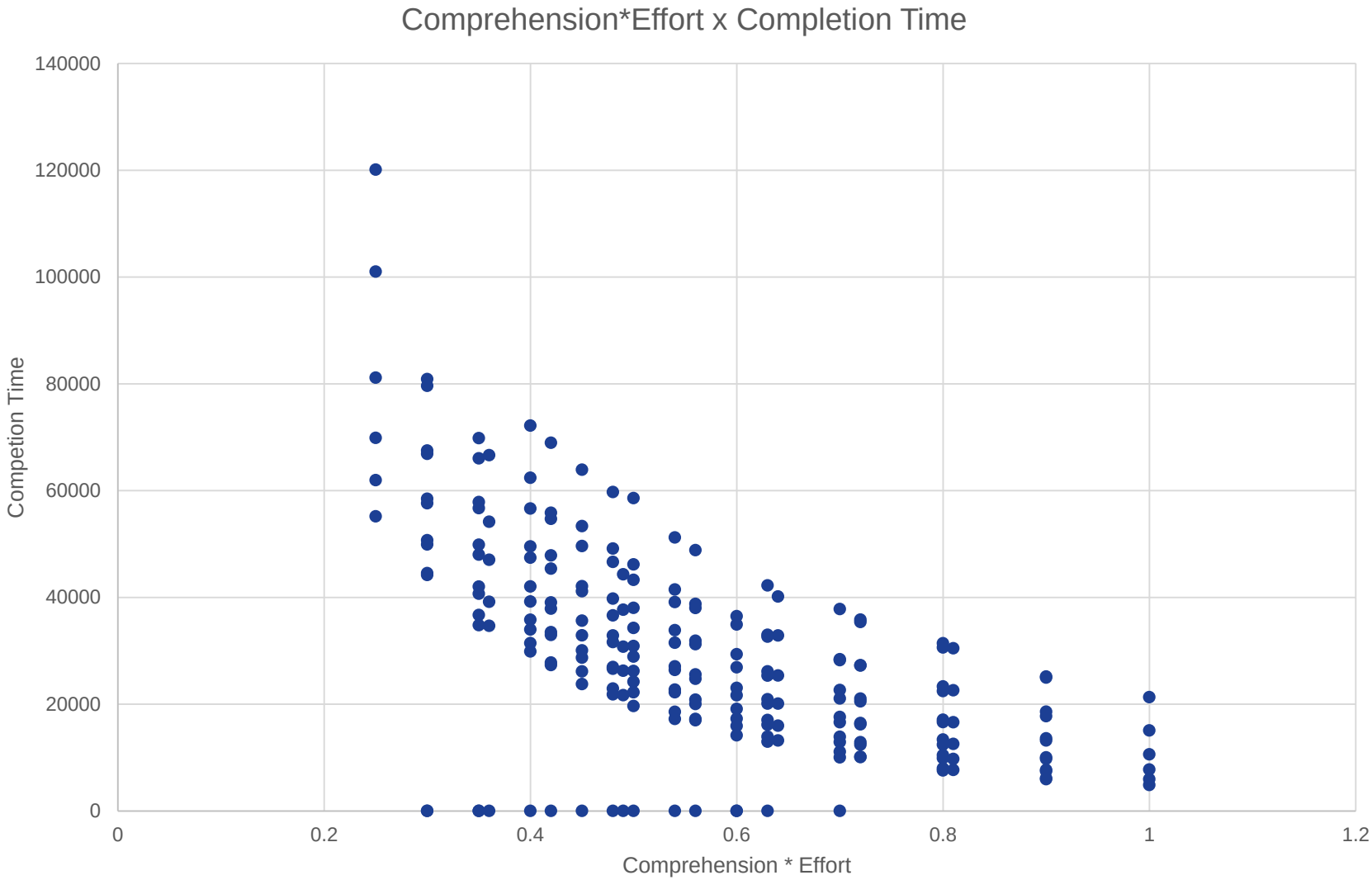
Simulation Results



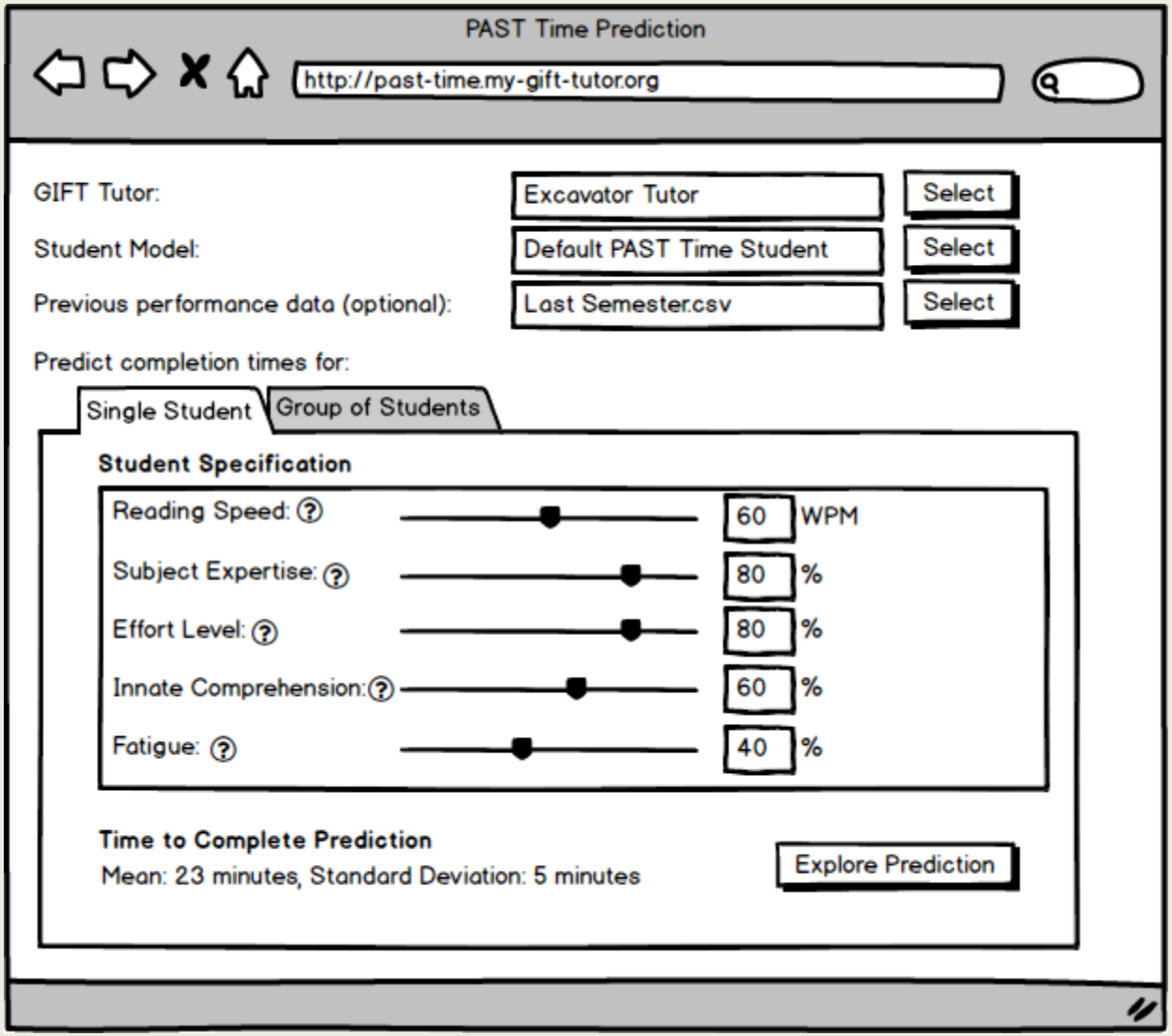
Simulation Results (2)



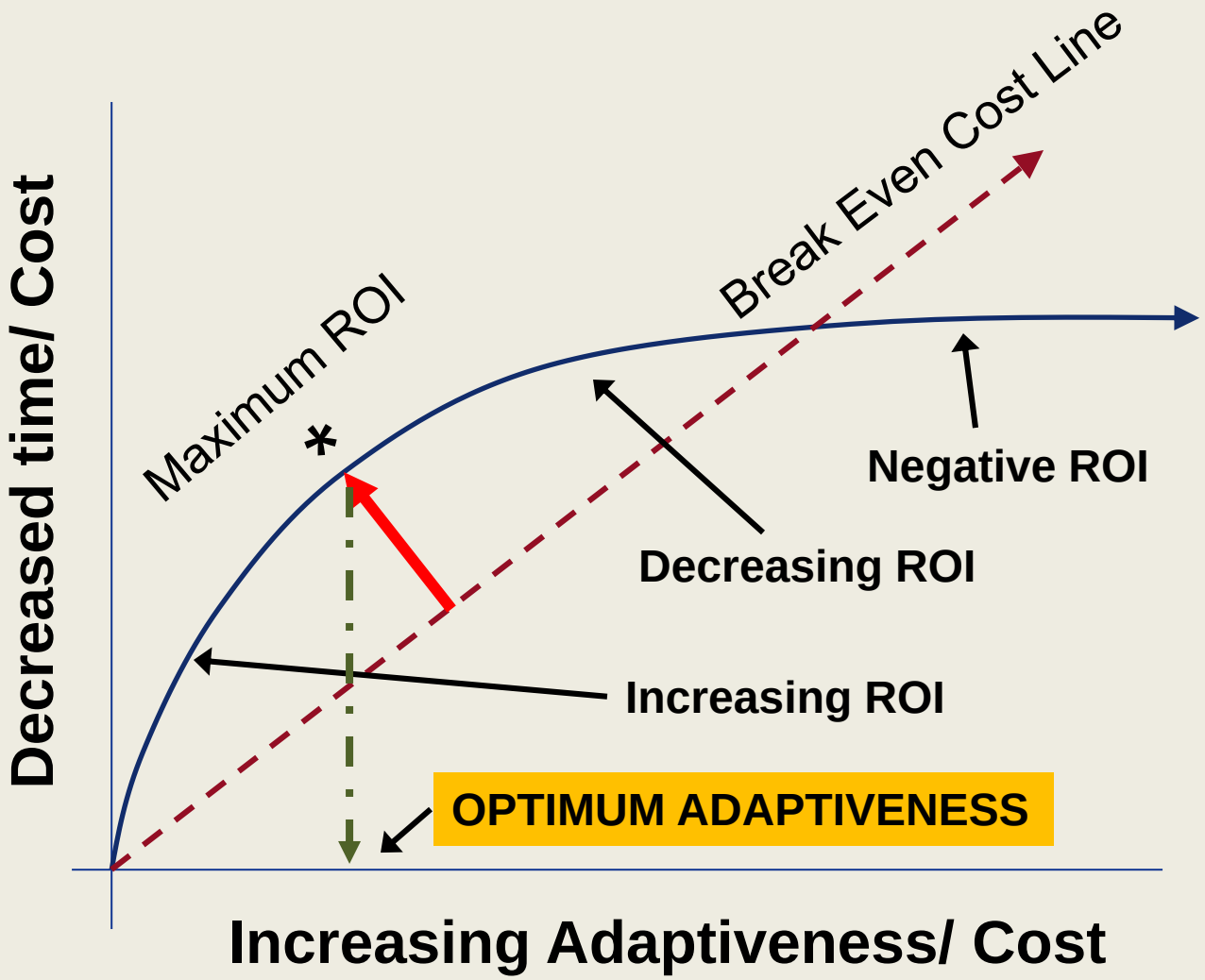
Simulation Results (3)



User Interface to Modeling



Maximizing ROI

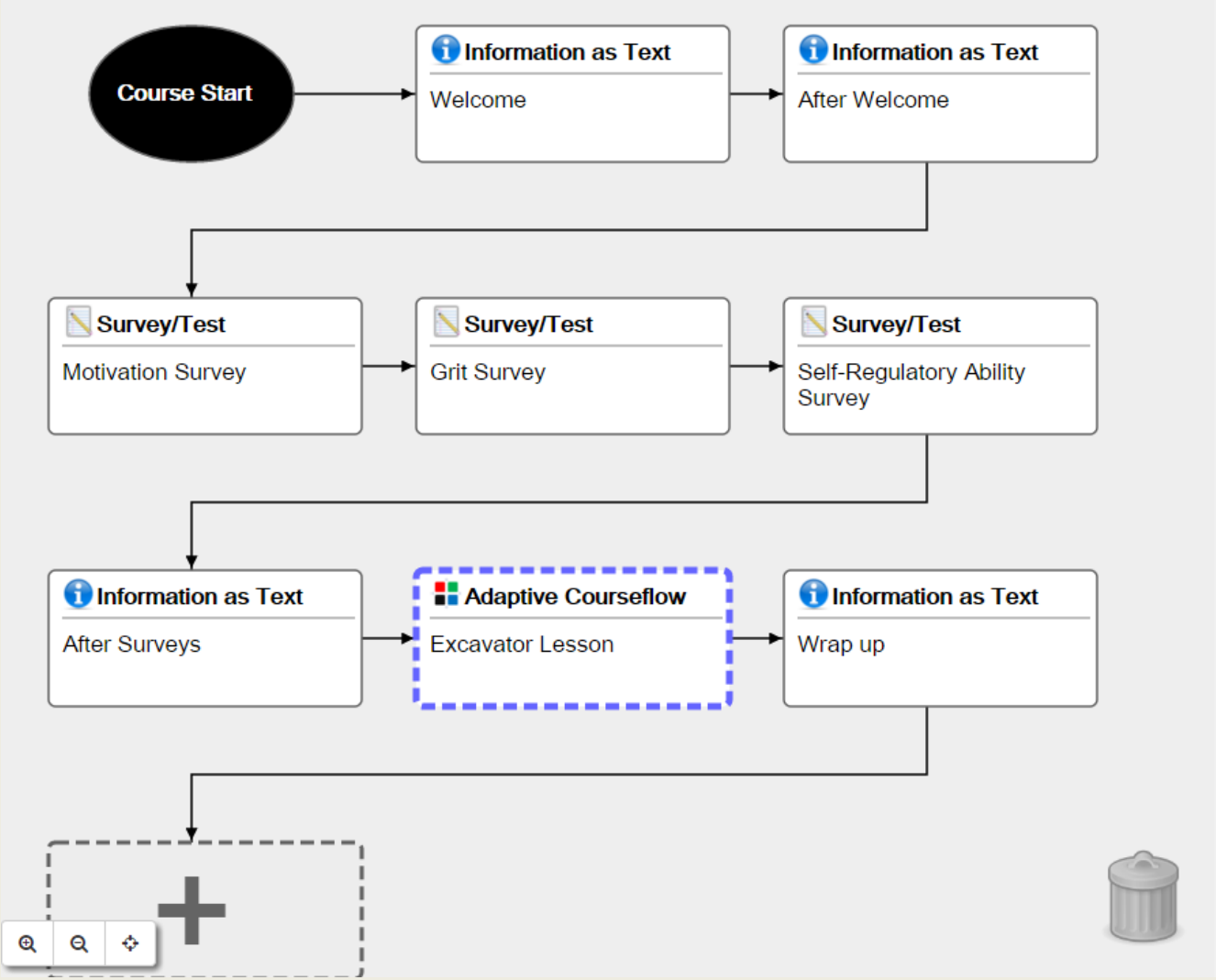


Conclusions

- Calculating ROI is difficult for training – examining time saved is a simple but easy metric to consider focused on the cost of training delivery.
- Future work will validate the predictive model with learner data.
- Other applications of the model include :
  - Run time monitoring: identify anomalous students in need of intervention.

Back Up Slides

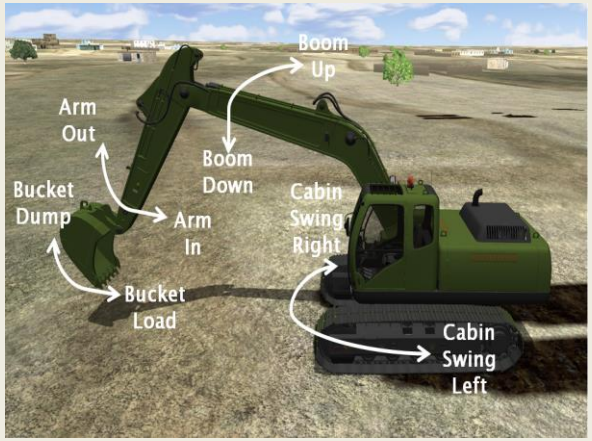
Excavator Trainer Course Map



Adaptive Courseflow Object

Rules

Here are the excavator components and their movements



Examples

This is how you move the bucket.

Recall

Which control is labeled “D” on the Excavator?

Practice

Complete tasks in virtual environment

Figaro Probabilistic Programming Language

1,000 characters of text

0.2 probability of prior knowledge

Population reading speed: normally distributed @100 characters/min

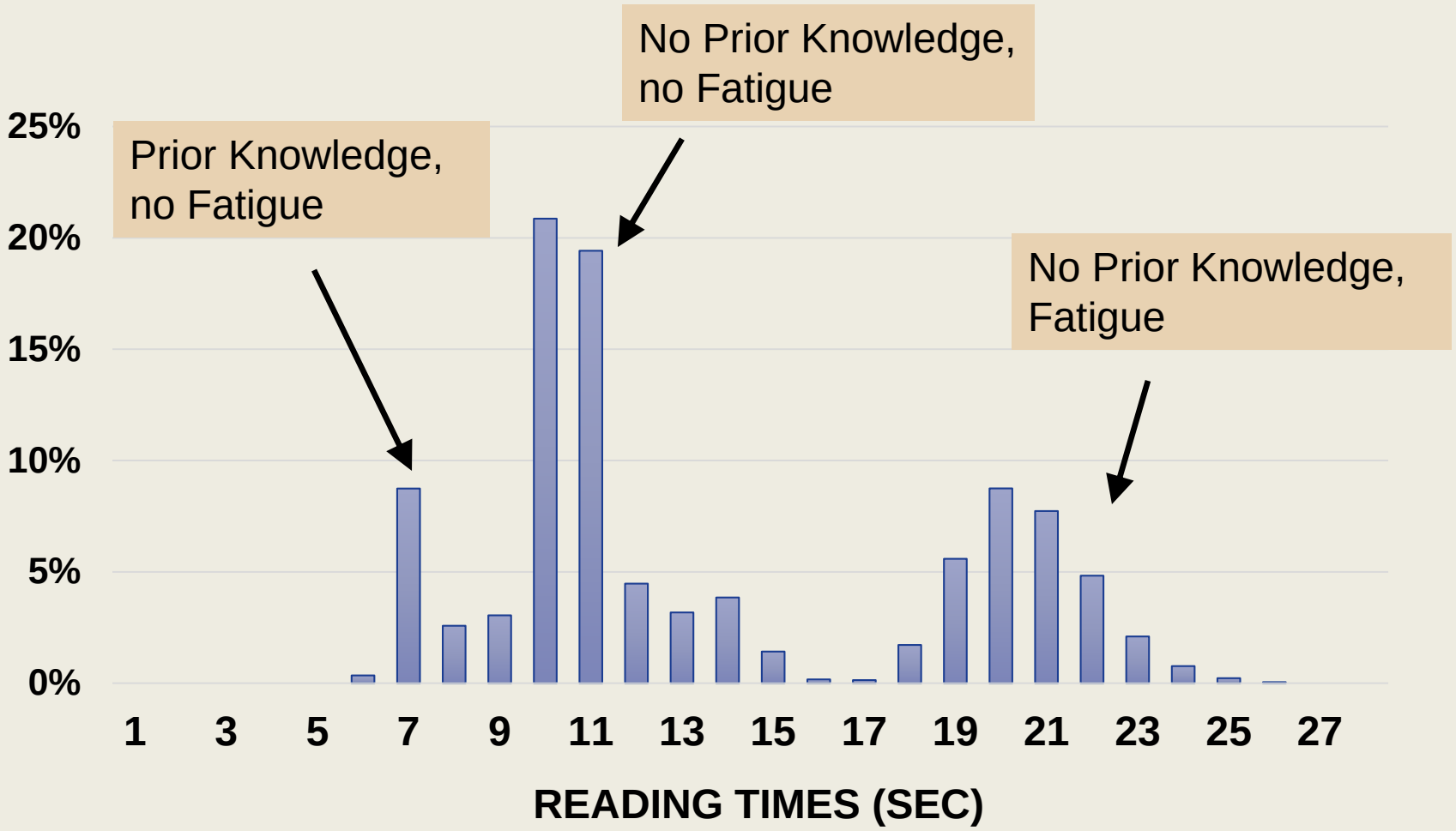
```
val text = Constant(1000.0)
  val priorKnowledge = Flip(0.2)
  val populationReadSpeed = Normal(100.0, 50.0)
  val readSpeed = If(priorKnowledge,
    populationReadSpeed ++ Normal(50.0, 25.0), populationReadSpeed)
  val fatigued = Flip(0.4)
  val readingTime = If(fatigued,
    text / (readSpeed * Constant(0.5)), text / readSpeed)

  val algorithm = Importance(10000, readingTime)
  algorithm.start
  println(algorithm.distribution(readingTime))
```

Prior knowledge increases around 50 char/min

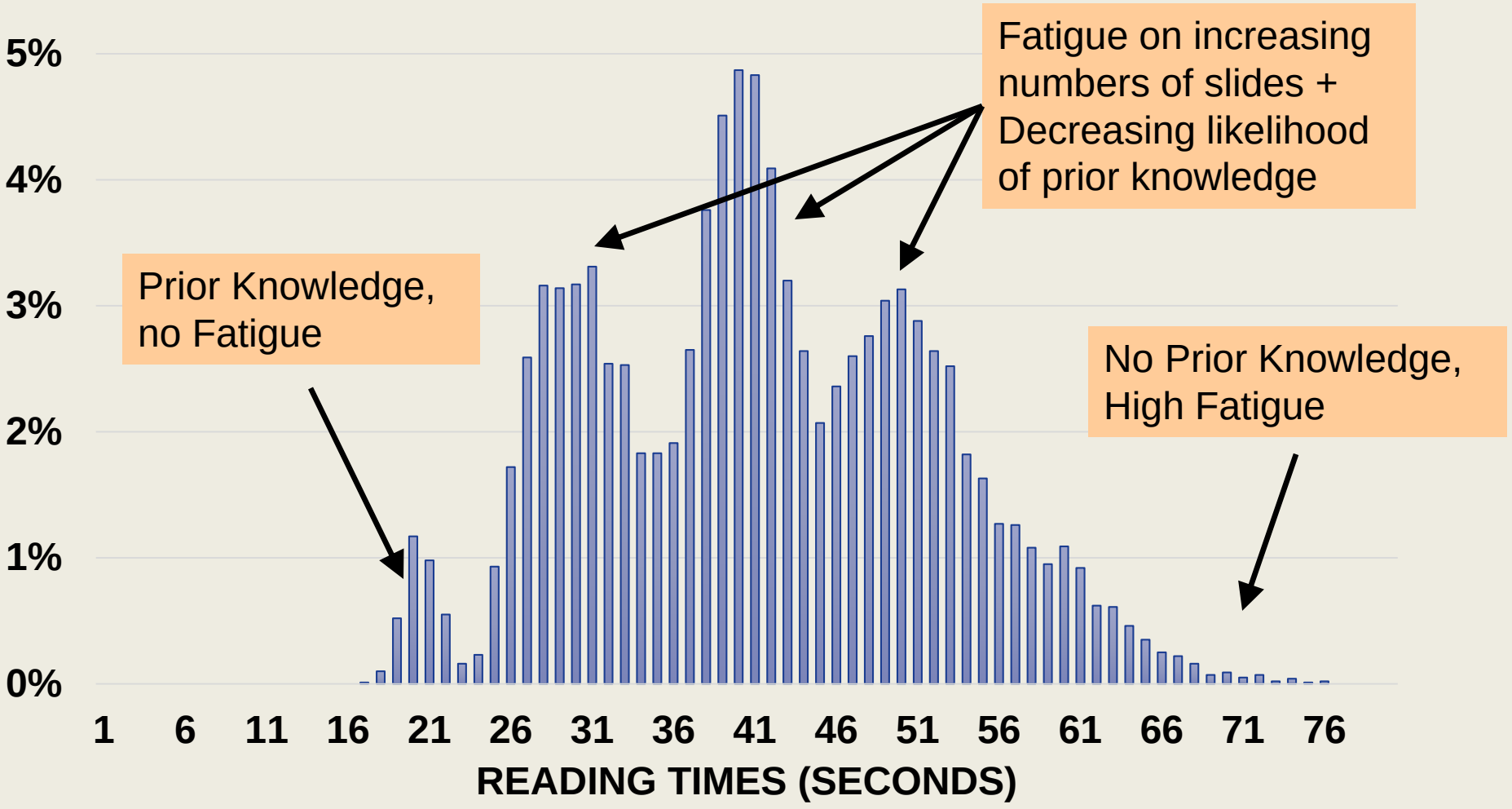
Learner reading speed

Effect of fatigue – decrease speed by 50%

Reading Time 2 Slides



Reading Time: 3 Slides



Probability of fatigue increases 5% with each slide