Learning Overtime Dynamics through Multiobjective Optimization

Márcio Barros, Luiz Antônio Araújo Jr.

PPGI/UNIRIO, Rio de Janeiro, Brazil

{marcio.barros, luiz.araujojr}@uniriotec.br

UNIRIO

Introduction

- Software developers frequently work overtime, most of the occasions without previous planning

- Unplanned overtime is not good for the health and the lives of the software developers (nor for the product they develop …)

- Working overtime also costs a lot to the companies – in Brazil, the first two hours go up to 120% of the baseline value, and the next two hours cost 150% of the baseline

- Moreover, productivity slows when developers get tired and nasty bugs creep from tired fingers

Introduction

So, companies are paying more for less productivity and are buying something developers don't want to sell -- their five o'clock world!

"Five O'clock World", performed by The Vogues
Good Morning, Vietnam! soundtrack

Objective

To study the effectiveness of different approaches for overtime planning considering the negative effects of overtime on the goods and services produced by the software developers.

Software project model

Simulator

Genetic optimization

The state of the art

- Many papers have applied heuristic optimization to support developer assignment to tasks in software projects
- Some papers have applied similar optimization techniques to study the dynamics of software projects
- Recently, Ferrucci et al. (2013) used a genetic algorithm to find the best allocation of overtime hours on the schedule of software projects
- What is new in our approach? We account for the errors made by tired developers working overtime and the effort required to correct them

Ferrucci et al., “Not going to take this anymore: Multi-objective overtime planning for software engineering projects”, ICSE, 2013

Modeling a software project

- A software project is modeled as a set of working packages

Manage students

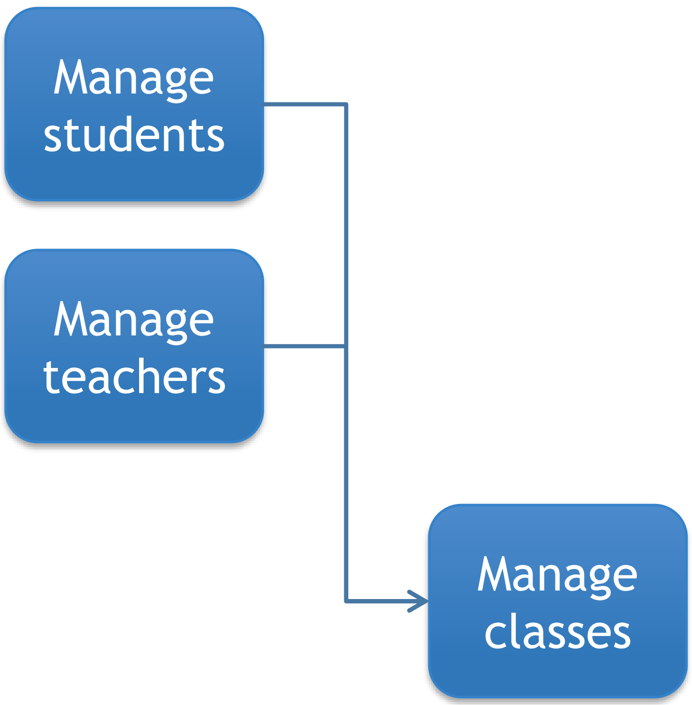
Manage teachers

Manage classes

Taking part of the ACAD instance as an example …

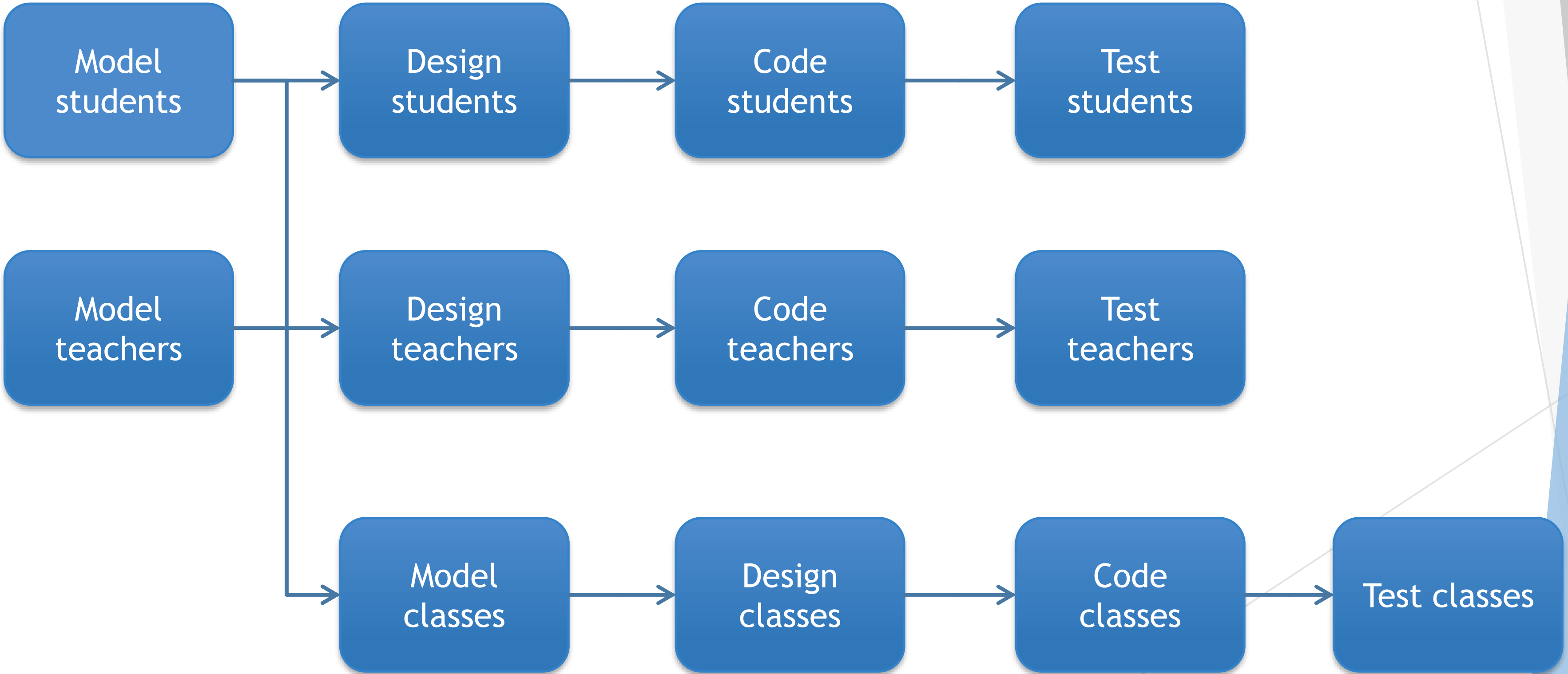
Modeling a software project

- A working package may depend on other working packages



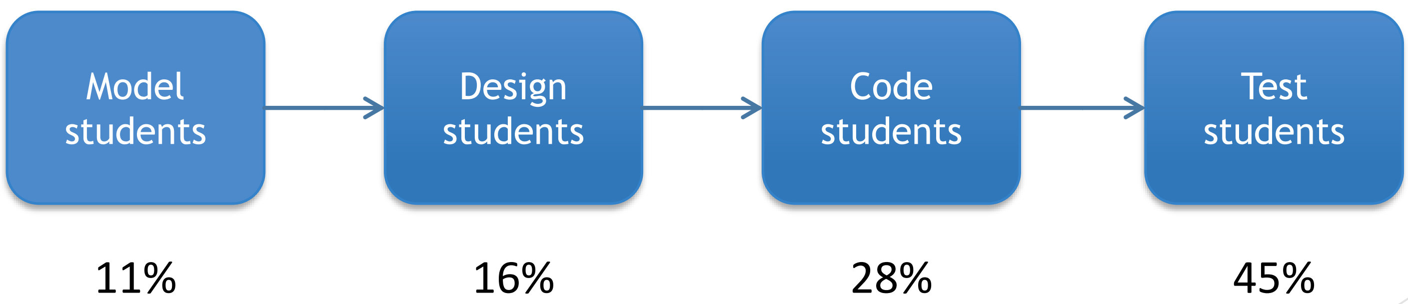
Modeling a software project

- Working packages must be modeled, designed, coded and tested



Modeling a software project

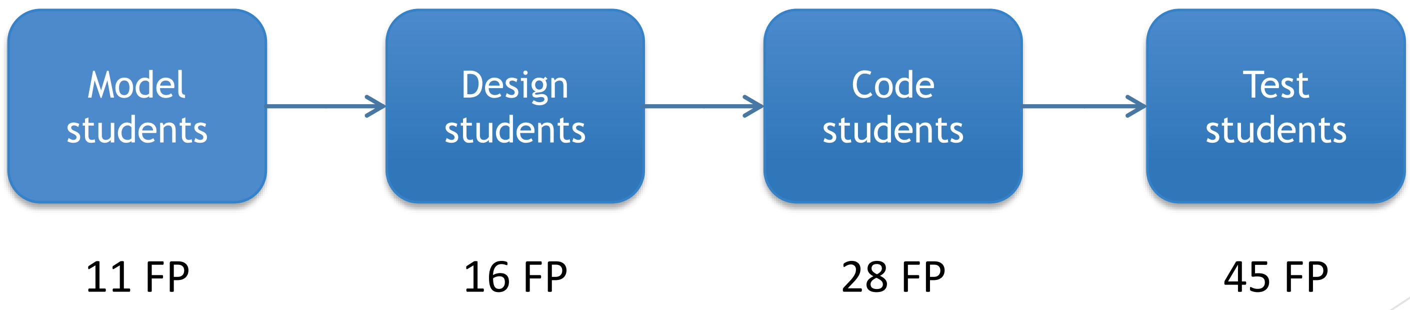
- Each activity downstream requires an amount of effort …



Jones, “Software Assessments, Benchmarks, and Best Practices”, Addison-Wesley, 2000

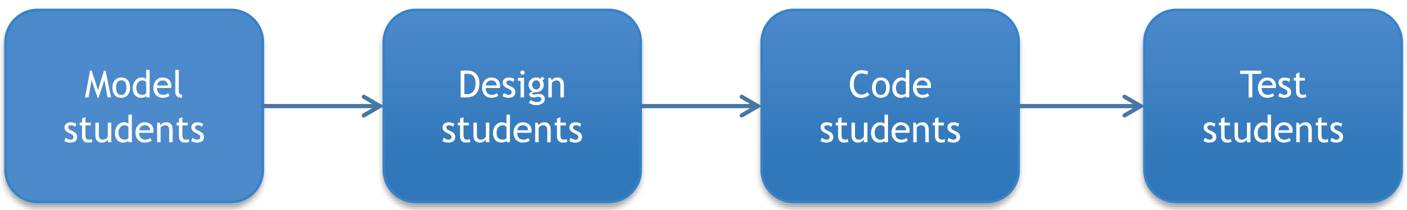
Modeling a software project

- The effort is represented in function points, an abstract measure of the complexity to develop a set of software requirements



Modeling a software project

- A regular developer builds about 28 FP/month …

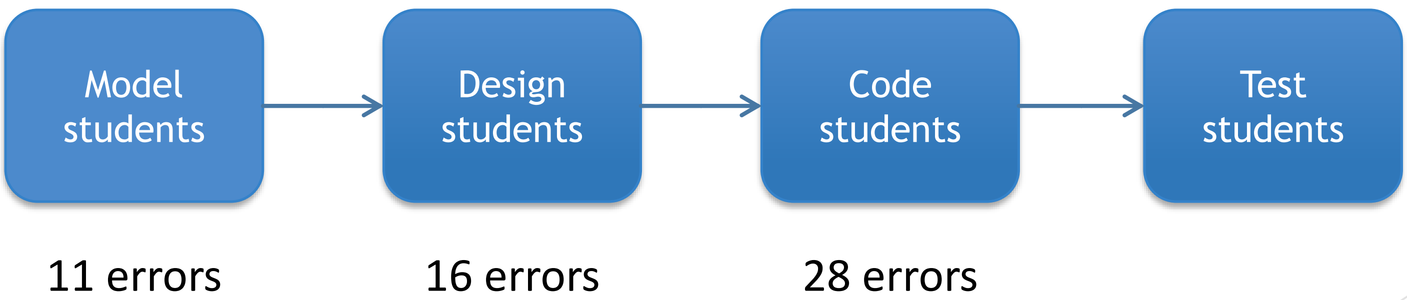


68 hours | 102 hours | 170 hours | 281 hours

Jones, "Software Assessments, Benchmarks, and Best Practices", Addison-Wesley, 2000

Modeling a software project

- … and introduces on average one error/FP.



Jones, “Software Assessments, Benchmarks, and Best Practices”, Addison-Wesley, 2000

Modeling a software project

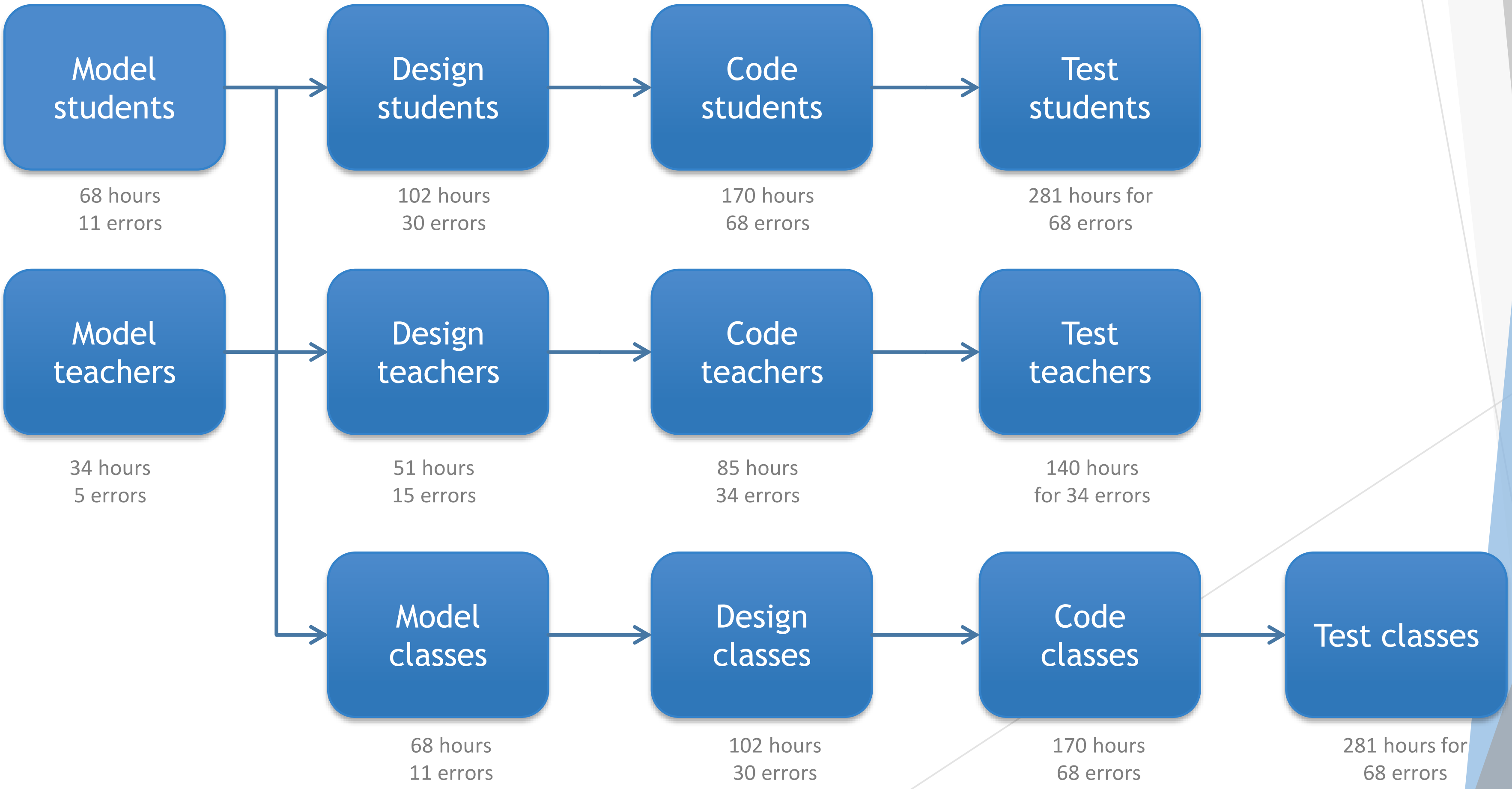
- These errors regenerate themselves as development moves downstream

Model students	→	Design students	→	Code students	→	Test students
11 errors		11 errors received + 25% regenerated + 16 new errors = 30 total errors		30 errors received + 33% regenerated + 28 new errors = 68 total errors		68 errors to be found and corrected

Abdel-Hamid e Madnick, “Software Project Dynamics”, Prentice-Hall, 1991

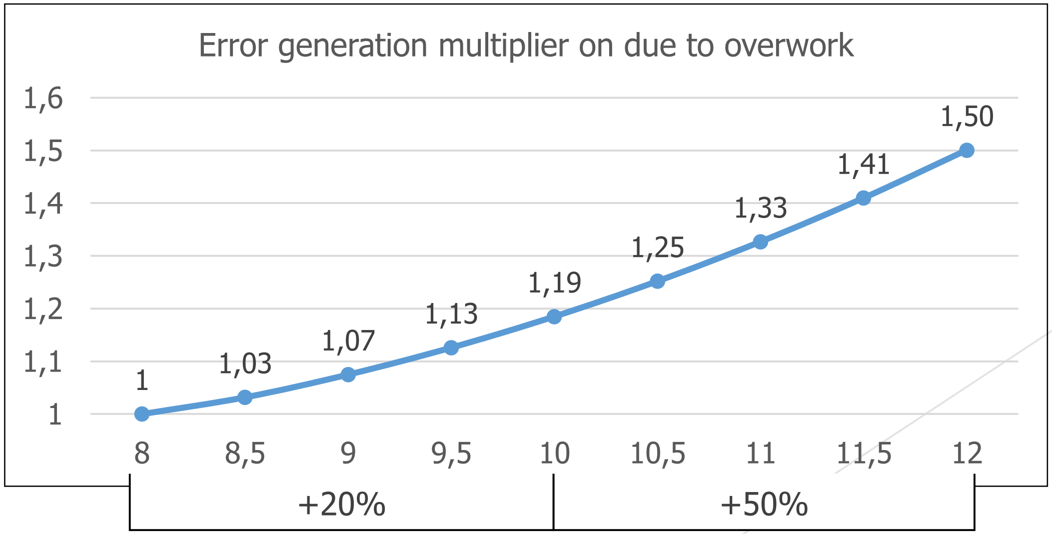
Modeling a software project

- In the end, we have a complex schedule with effort and error dynamics



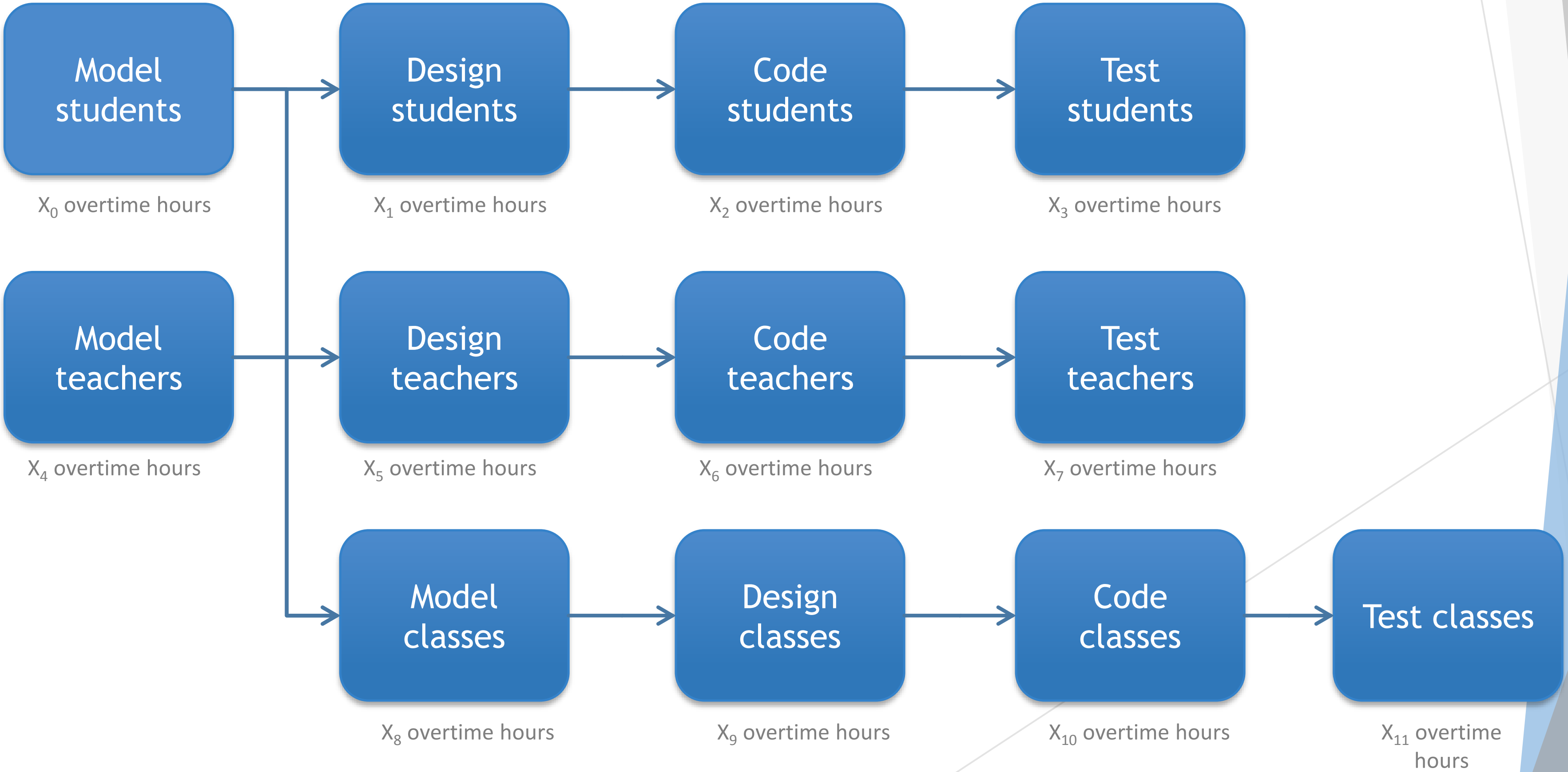
Modeling a software project

- Now suppose we have tired developers …
  - They work more hours with the same productivity (conservative approach)
  - Error generation rates are increased due to overtime
  - The number of errors increases even more due to regeneration
  - Project costs increases, but duration may be shorter
  - Testing activities may last longer to correct the new errors



The optimization problem

- The question is: how much overtime work should be planned for each activity comprising the schedule of the software project?



The optimization problem

A potential solution

For each activity, we define how many overtime hours it uses.

Multiples of 30 minutes.
Minimum: zero hours.
Maximum: 4 hours.

Solution evaluation

The number of overtime hours for each activity is fed into the software model.

The model is simulated (takes some seconds) and we collect and compare the results.

Given N activities, we have nine alternatives for each of them, leading to a solution space comprising 9^N potential solutions.

We cannot examine all alternatives!

The optimization problem

- We want a balance between …
  - The total cost for the project (in monetary units, to be minimized)
  - The duration of the project (in days, to be minimized)
  - The amount of overtime work (in hours, to be minimized)

- A solution is represented by a number of overtime hours assigned for each activity (from zero to four hours)

- Given a solution, we run the software model in a simulator and calculate the objectives

- A MOGA (we have used NSGA-II) finds the best allocation of overtime hours for each activity

Evaluation of the proposed approach

- RQ1 (Competitiveness): How does NSGAII perform if compared to currently used overtime planning strategies?

- RQ2 (Usefulness): Do results of overtime planning change if we consider the dynamics of error generation and the loss of quality caused by overtime?

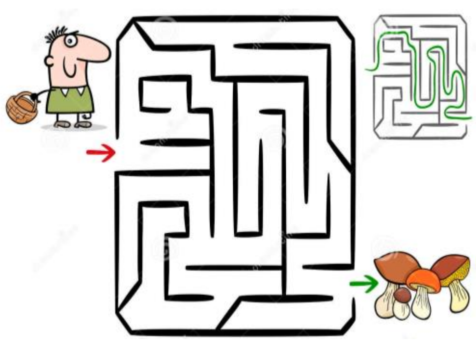
Project	Function Points	Activities
ACAD	185	40
WMET	225	44
WAMS	381	60
PSOA	290	72
PARM	451	108
OMET	635	84

Evaluation of the proposed approach

- RQ1: Industrial approaches for overtime allocation (Ferrucci et al. 2013)

Margarine (MAR)

Spread the number of overtime hours uniformly across the project.



Shortest Path (CPM)

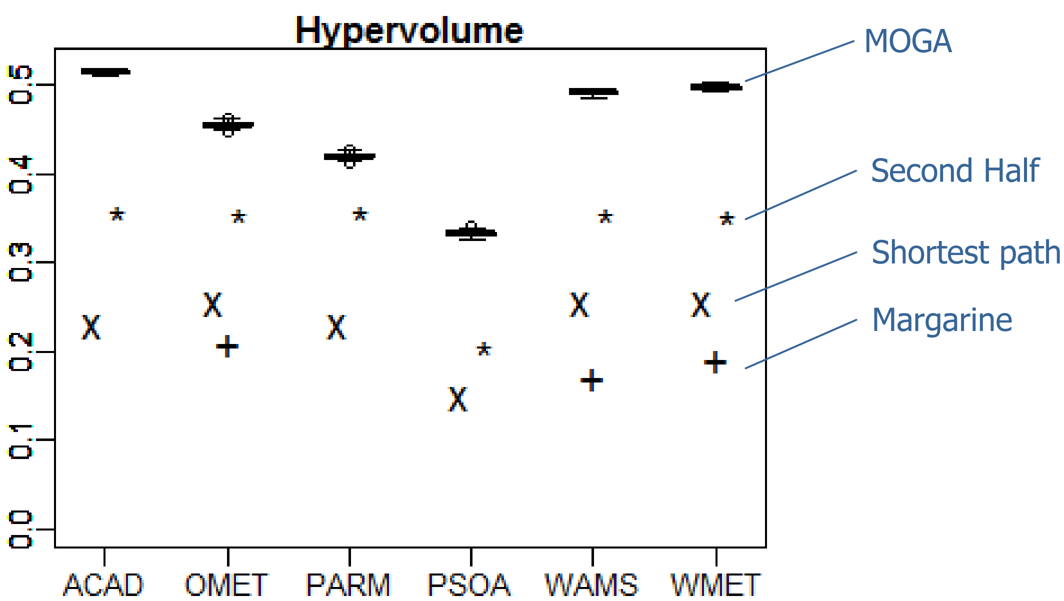
Add overtime hours to critical path activities, i.e., activities that must not be delayed.

Second Half (SH)

Add overtime hours to those activities in the second part of a project's schedule.

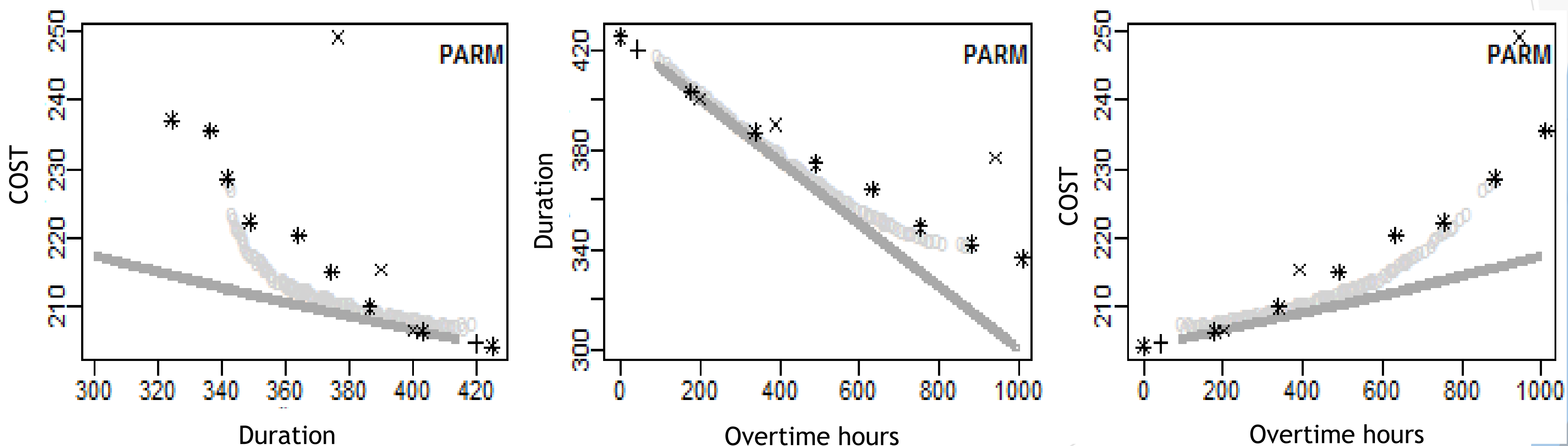
Evaluation of the proposed approach

- RQ1: Comparison of quality indicators
  - MOGA clearly shows better results than Shortest Path and Margarine, but Second Half is competitive



Evaluation of the proposed approach

- RQ2: Impact of increased error rates due to overtime dynamics
  - Results without the increased error rates (dark gray) are very distinct from those accounting for the increase in error generation (light gray)
  - The results from the Second Half industrial strategy (*) are close to those accounting for the increase in error generation



Evaluation of the proposed approach

- Conclusion: if we do not take into account the new errors generated due to overtime, project duration and cost might be underestimated

Project	Average daily overtime hours	No Increase < Increased	
		Duration	Cost
ACAD	2,5 to 3,0	5,13 %	3,51 %
OMET	2,7 to 3,3	8,99 %	5,86 %
PARM	2,5 to 2,8	7,27 %	5,50 %
PSOA	2,7 to 3,0	2,23 %	0,99 %
WAMS	3,0 to 3,5	9,21 %	5,24 %
WMET	3,0 to 3,5	8,56 %	5,83 %

Evaluation of the proposed approach

- Major threats to validity are related to model limitations
  - Selection of parameters to describe effort estimation for each activity
  - Selection of parameters to model error generation and regeneration
  - Does not account for particular skills that developers may have
  - Assumes a single developer per activity
  - Assumes a linear cost for error identification and correction
  - Considers that testing activities correct all errors

Conclusions

- Conclusions
  - Experimental results support the Second Half overtime planning strategy, adding evidence to project manager's field experience
  - The adverse effects of overtime working should be taken into account for credible duration and cost estimations in software projects
- Contributions
  - Software project model to evaluate the effects of overtime working
  - A new multiobjective formulation for the overtime planning problem
  - Experimental study based on real-world software project
- Future work involves testing other MOGA and integration to project management tools

Thank you!