Introduction to Software Engineering

CMPT 276 - D300

Today’s Agenda

- What is software engineering?
  - Software engineering vs programming

- Why is it difficult?
  - How Committees Invent
  - Programming as Theory-Building

- Why is it worth studying?
  - To build a career
  - To learn how to solve problems in any domain
  - To help the world

Software Engineering

Software Engineering

- Programming?

Software Engineering

- Programming?
  - The act of writing a computer program

Software Engineering

- Programming?
  - The act of writing a computer program

NO

Software Engineering

⬆

Software Engineering

- The application of mathematics or systematic knowledge beyond the routine skills of practise, for the design of any complex system which performs useful functions
  - GNU version of the Collaborative International Dictionary of English

Software Engineering

- The application of mathematics or systematic knowledge beyond the routine skills of practise, for the design of any complex system which performs useful functions
  - GNU version of the Collaborative International Dictionary of English

Software Engineering

- The application of mathematics or systematic knowledge beyond the routine skills of practise, for the design of any complex system which performs useful functions
  - GNU version of the Collaborative International Dictionary of English

complex **software** systems

Software Engineering

- Software engineering encompasses the
  - design
  - implementation
  - deployment
  - and maintenance

  of software, in order to achieve tangible outcomes

Software Engineering

- “I want to schedule waiters in my restaurant”

Software Engineering

- “I want to schedule waiters in my restaurant”
  - Determine requirements:
    - how many waiters?
    - what kind of coverage do you need?
    - what happens if someone is sick?
    - how many hours per day/week can each waiter work?
    - does seniority matter?
    - how often will the scheduler need to run?
    - where will the code execute?
    - what kind of interface do you need?
    - how secure does this thing need to be?

Software Engineering

- “I want to schedule waiters in my restaurant”
  - Make design decisions:
    - what programming language?
    - what execution environment?
    - what architecture/data structures?
    - what scheduling algorithm?
    - how will the user interface look and behave?

Software Engineering

- “I want to schedule waiters in my restaurant”
    - Actually implement the thing

Software Engineering

- “I want to schedule waiters in my restaurant”
  - Actually implement the thing
  - Fix inevitable bugs

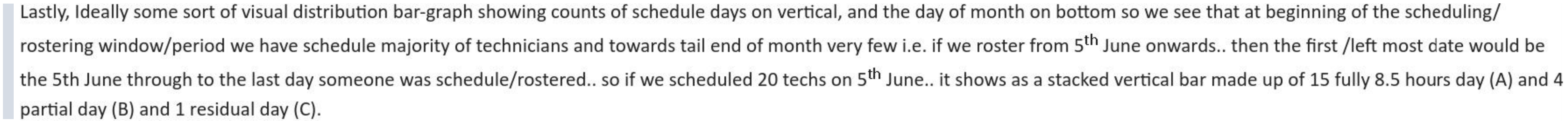
Software Engineering

- “I want to schedule waiters in my restaurant”
  - Actually implement the thing
  - Fix inevitable bugs
  - Address miscommunication of requirements

Software Engineering

- “I want to schedule waiters in my restaurant”
  - Actually implement the thing
  - Fix inevitable bugs
  - Address miscommunication of requirements
  - Address new requirements

Software Engineering



> Lastly, Ideally some sort of visual distribution bar-graph showing counts of schedule days on vertical, and the day of month on bottom so we see that at beginning of the scheduling/ rostering window/period we have schedule majority of technicians and towards tail end of month very few i.e. if we roster from 5th June onwards.. then the first /left most date would be the 5th June through to the last day someone was schedule/rostered.. so if we scheduled 20 techs on 5th June.. it shows as a stacked vertical bar made up of 15 fully 8.5 hours day (A) and 4 partial day (B) and 1 residual day (C).

Software Engineering

- “I want to schedule waiters in my restaurant”
    - Actually implement the thing
    - Fix inevitable bugs
    - Address miscommunication of requirements
    - Address new requirements
    - Deploy the thing

Software Engineering

- “I want to schedule waiters in my restaurant”
  - Actually implement the thing
  - Fix inevitable bugs
  - Address miscommunication of requirements
  - Address new requirements
  - Deploy the thing
  - Fix more bugs

Software Engineering

- Software engineering encompasses the
  - design
  - implementation
  - deployment
  - and maintenance

  of software, in order to achieve tangible outcomes

Software Engineering

- Software engineering encompasses the
  - design
  - implementation → programming
  - deployment
  - and maintenance

  of software, in order to achieve tangible outcomes

Software Engineering

- Software engineering encompasses the
  - design
  - impleme
  - deploym
  - and mai

  of software

Software Engineering

- Software engineering encompasses the
  - design
  - implementation
  - deployment
  - and maintenance

  of software, in order to achieve tangible outcomes

Software Engineering

- Software engineering encompasses the
  - design
  - implementation
  - deployment
  - and maintenance → ownership

  of software, in order to achieve tangible outcomes

Software Engineering

- Software engineering encompasses the
  - design
  - implementation
  - deployment
  - and maintenance

  of software, in order to achieve tangible outcomes

Why is software engineering difficult?

Software Engineering

- COMPLEXITY

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

- Tons of theories on how complexity is introduced, why, and methods to prevent it

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

- Tons of theories on how complexity is introduced, why, and methods to prevent it

- But complexity is inevitable

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Essential

Complexities of the domain and requirements

E.g., business logic for scheduling waiters is complicated, with many dependencies, rules, and exceptions to rules

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Essential

Complexities of the domain and requirements

E.g., business logic for scheduling waiters is complicated, with many dependencies, rules, and exceptions to rules

Accidental

Complexities arising from translating requirements into software

E.g., interfaces of objects in your system and in other libraries you use are complicated

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Essential

Complexities of the domain and requirements

E.g., busine
sched
complica
depe
exceptio

Bounds on human cognition

Accidental

Complexities arising from translating requirements into software

E.g., interfaces of objects in your system and in other libraries you use are complicated

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Essential

Complexities of the domain and requirements

E.g., business logic for scheduling waiters is complicated, with many dependencies, rules, and exceptions to rules

Accidental

Complexities ari translating requ into software

Imperfect communication

E.g., interfaces of objects in your system and in other libraries you use are complicated

Software Engineering

- **COMPLEXITY**
  - Where does it come from?

Essential

Complexities of the domain and requirements

E.g., business logic for scheduling waiters is complicated, with many dependencies, rules, and exceptions to rules

Accidental

Complexities ari
translating requ
into software

E.g., interfaces of object
in your system an o
libraries you use o
complicated

Imperfect communication

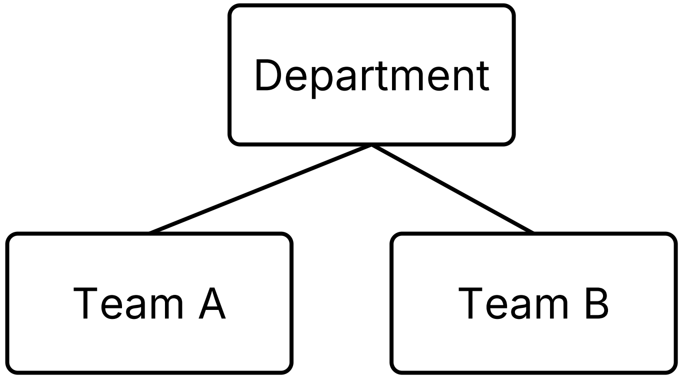
Bounds on human cognition

How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968

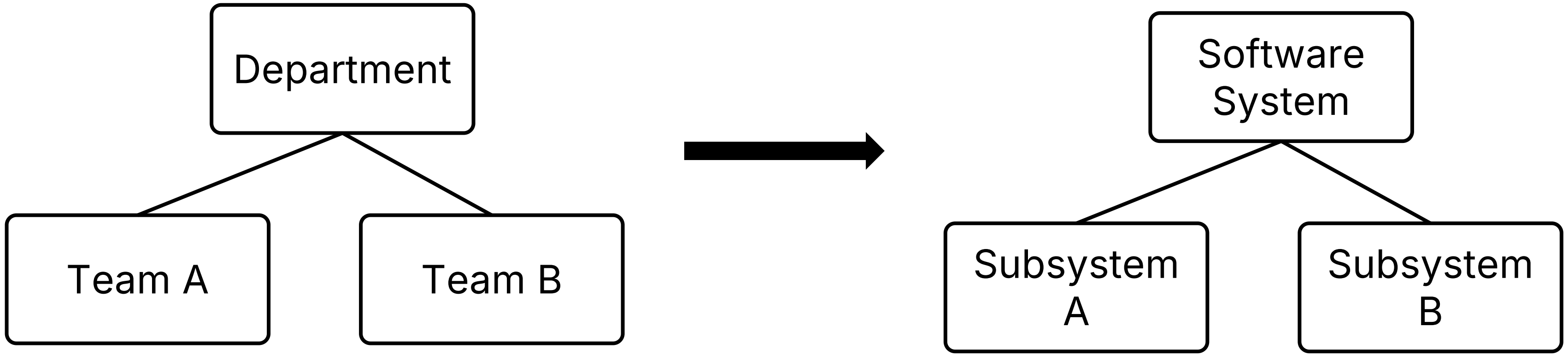
How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



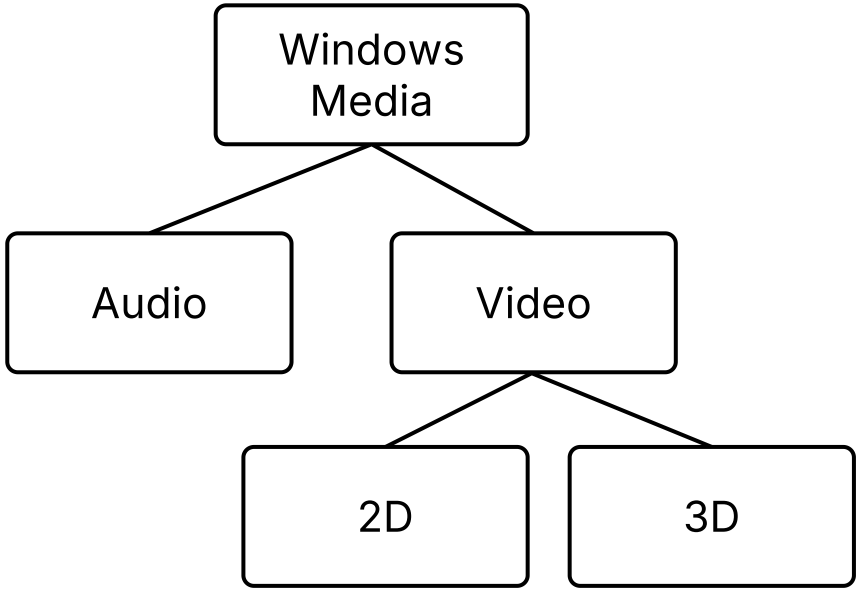
How Do Committees Invent?

- “...there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



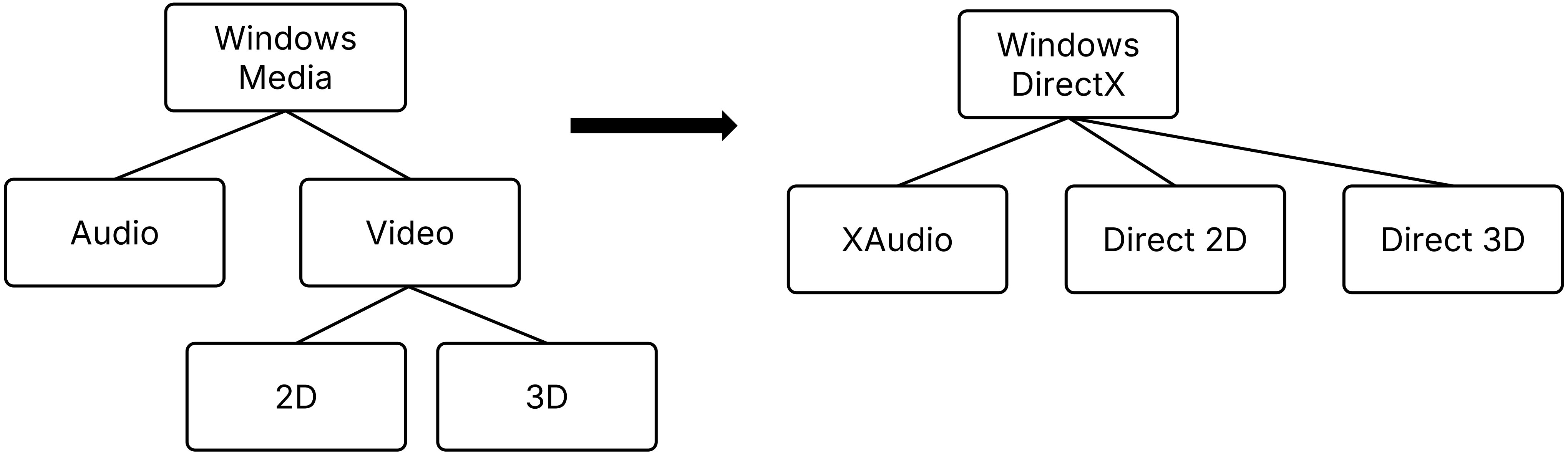
How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
    - How Do Committees Invent? Mel Conway, 1968



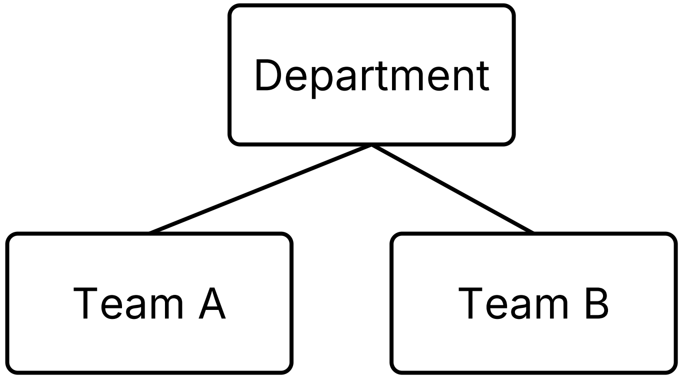
How Do Committees Invent?

- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



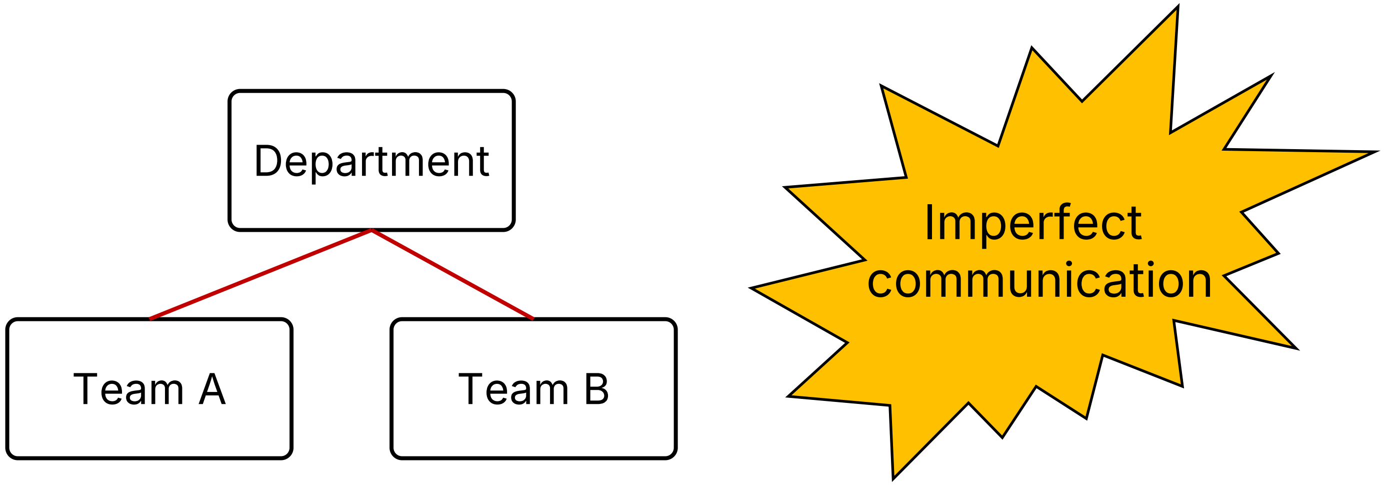
How Do Committees Invent?

- “...there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



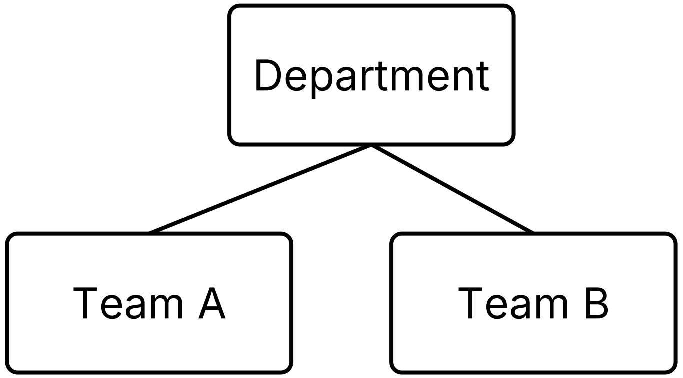
How Do Committees Invent?

- "…there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



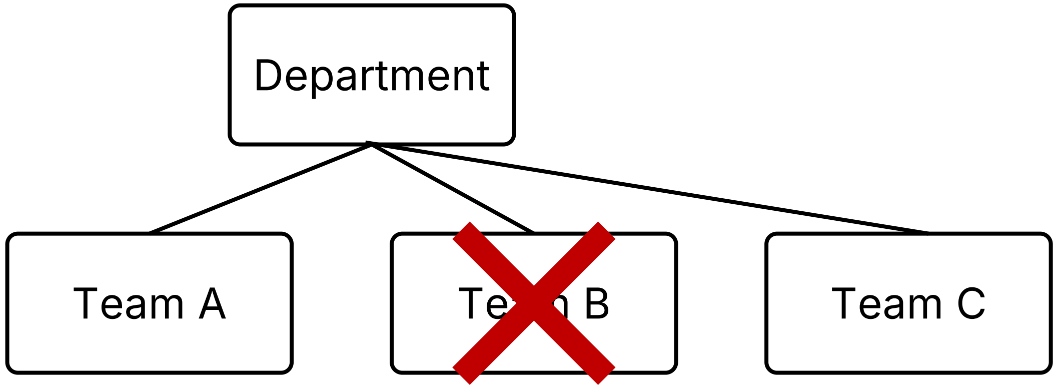
How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



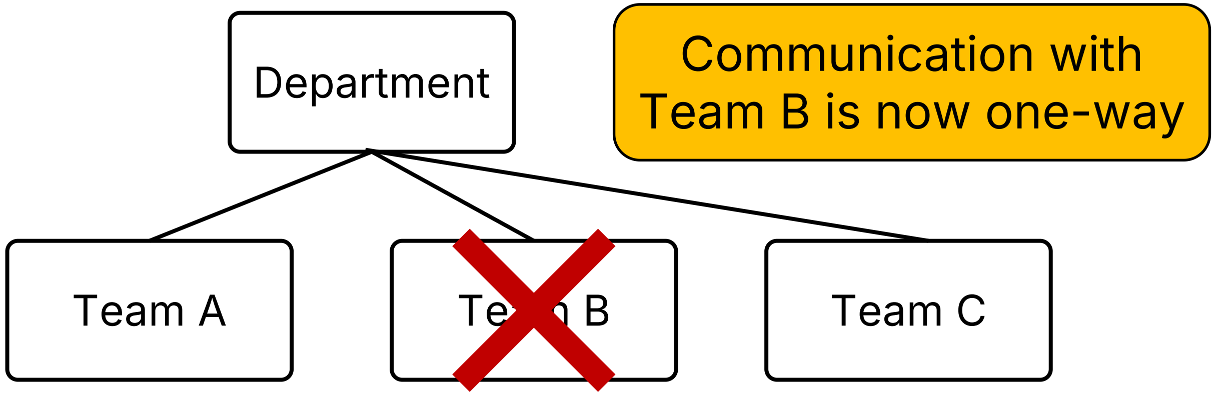
How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



How Do Committees Invent?

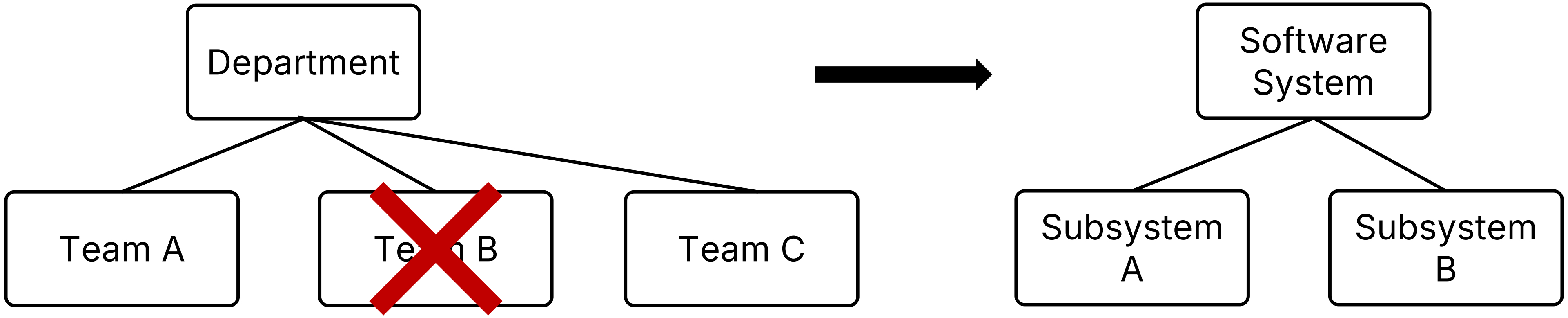
- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



Communication with Team B is now one-way

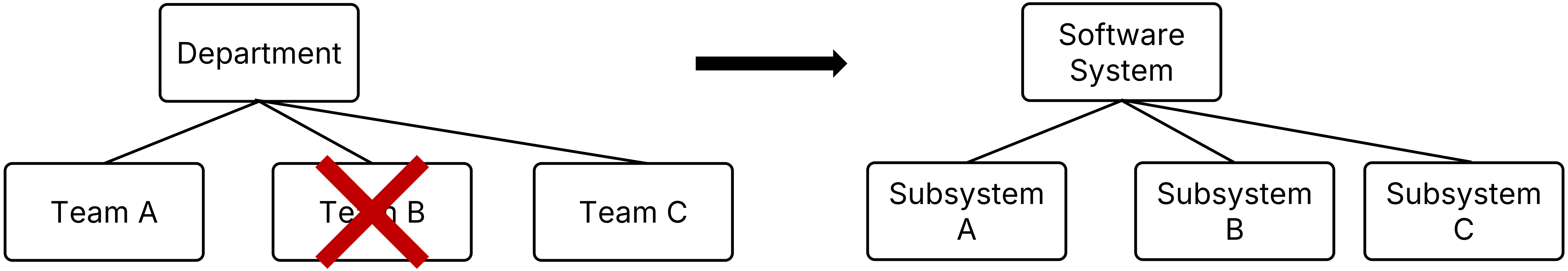
How Do Committees Invent?

- "…there is a very close relationship between the structure of a system and the structure of the organization which designed it."
    - How Do Committees Invent? Mel Conway, 1968



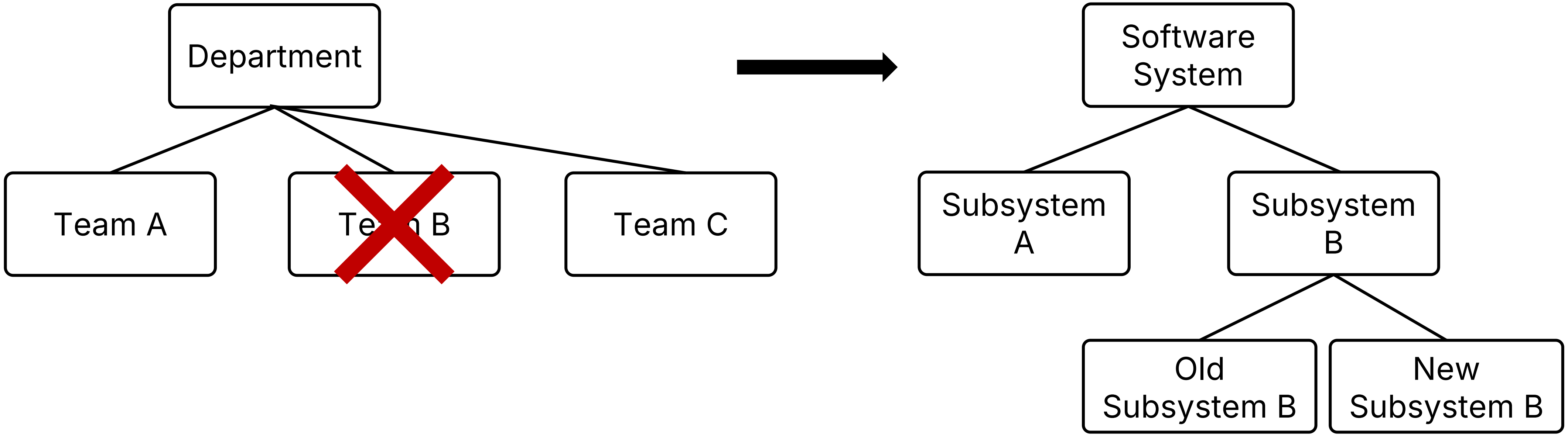
How Do Committees Invent?

- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



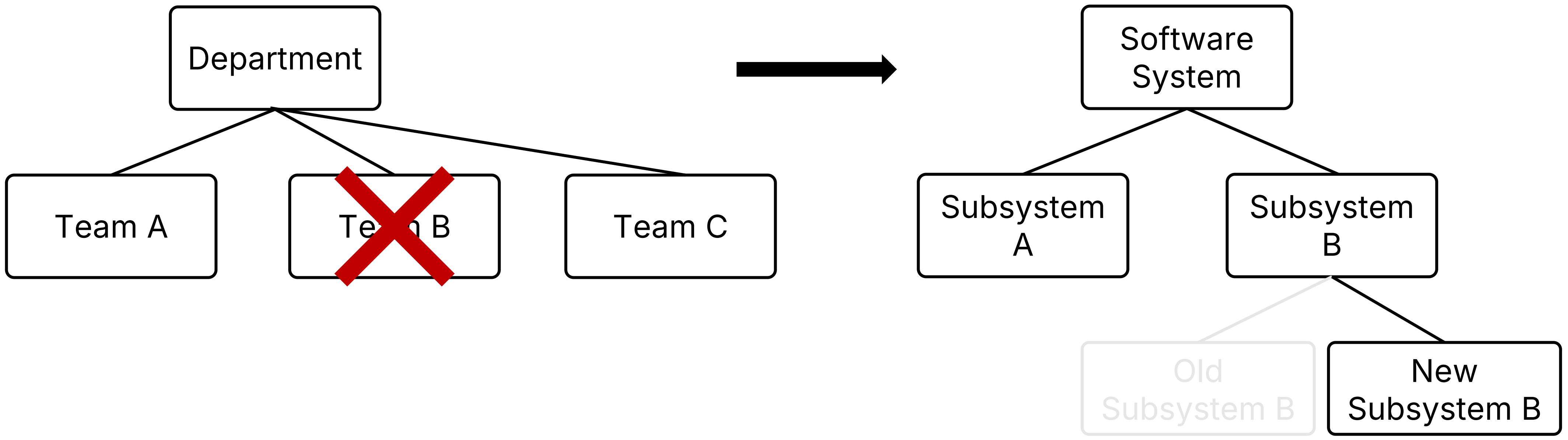
How Do Committees Invent?

- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



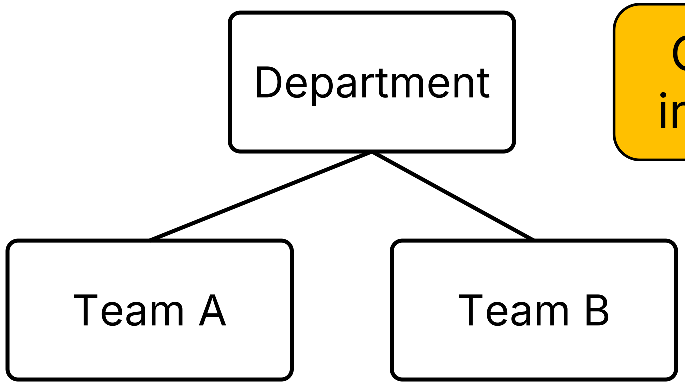
How Do Committees Invent?

- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



How Do Committees Invent?

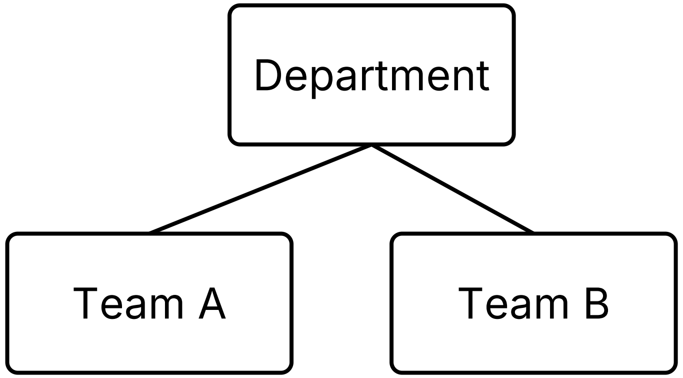
- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



Change in personnel: increase in complexity

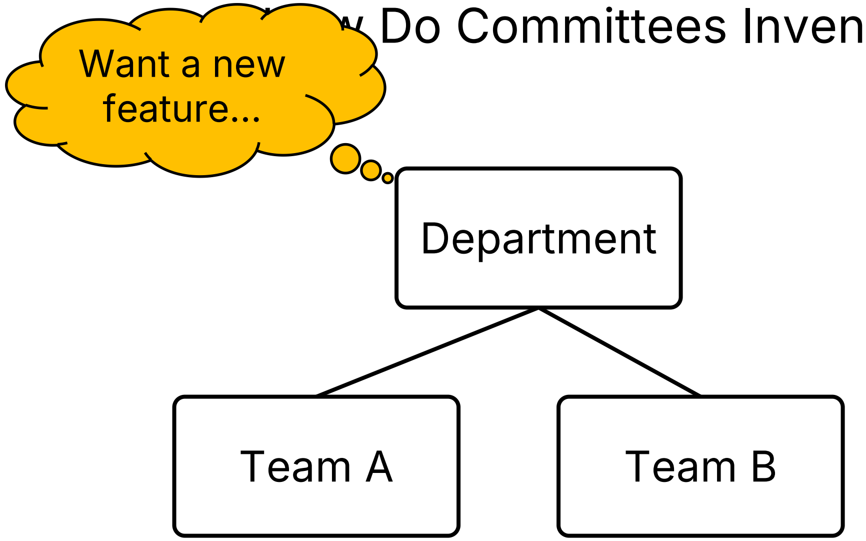
How Do Committees Invent?

- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - How Do Committees Invent? Mel Conway, 1968



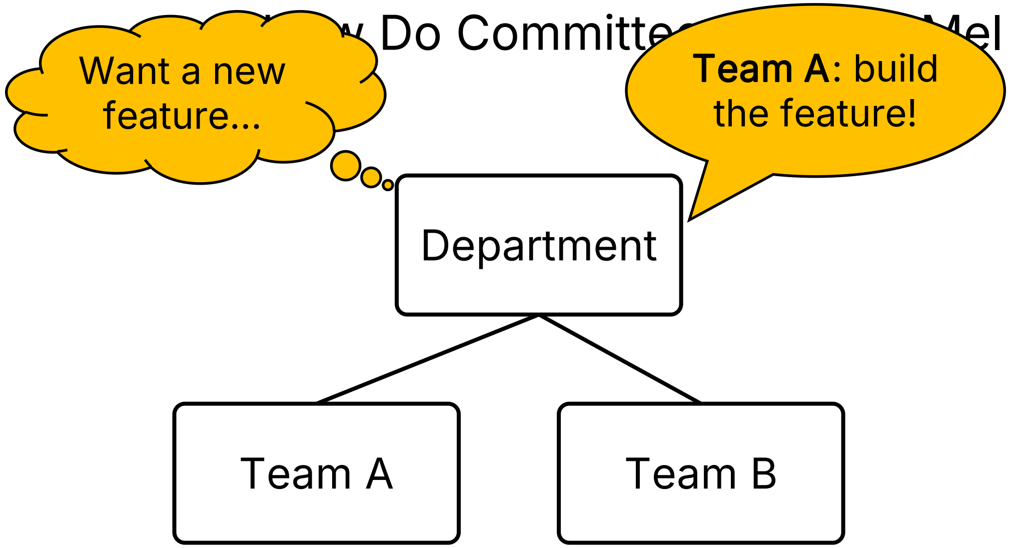
How Do Committees Invent?

- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - w Do Committees Invent? Mel Conway, 1968



How Do Committees Invent?

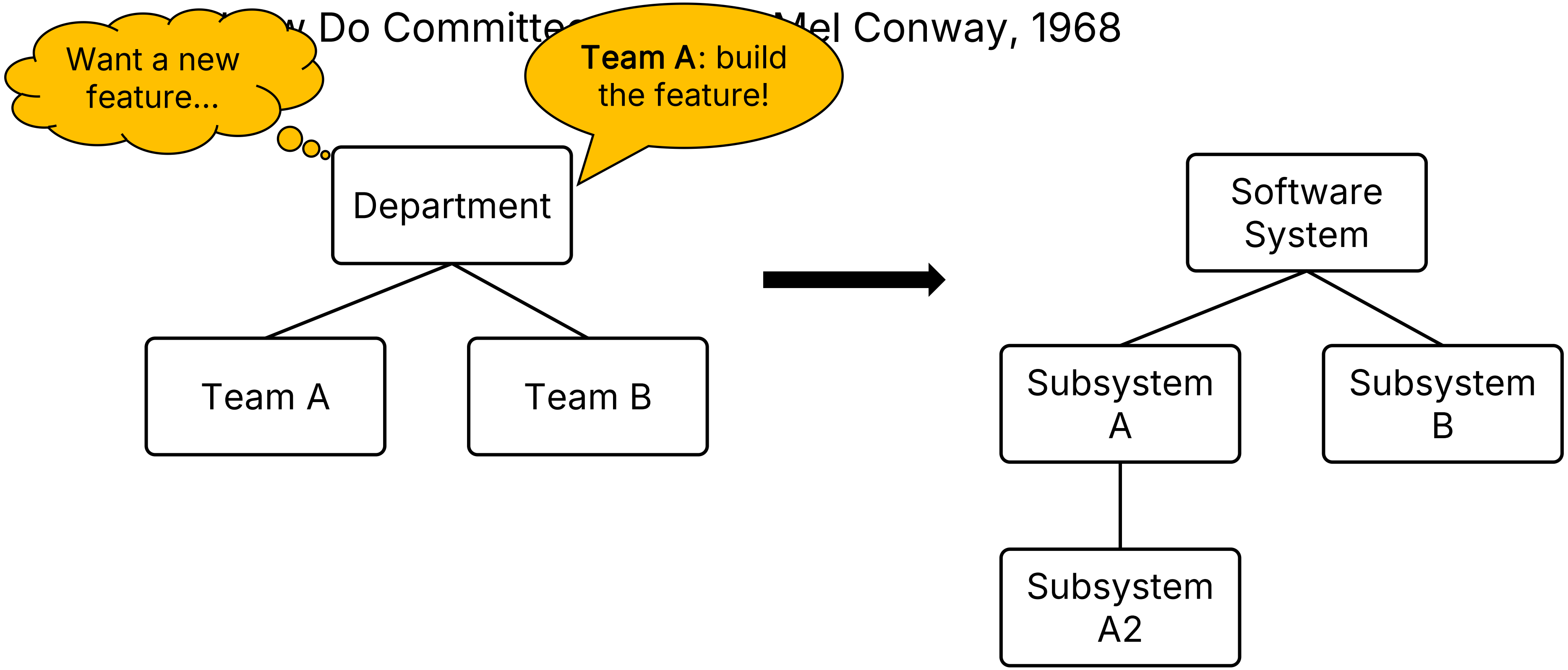
- "...there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - w Do Committe Mel Conway, 1968



How Do Committees Invent?

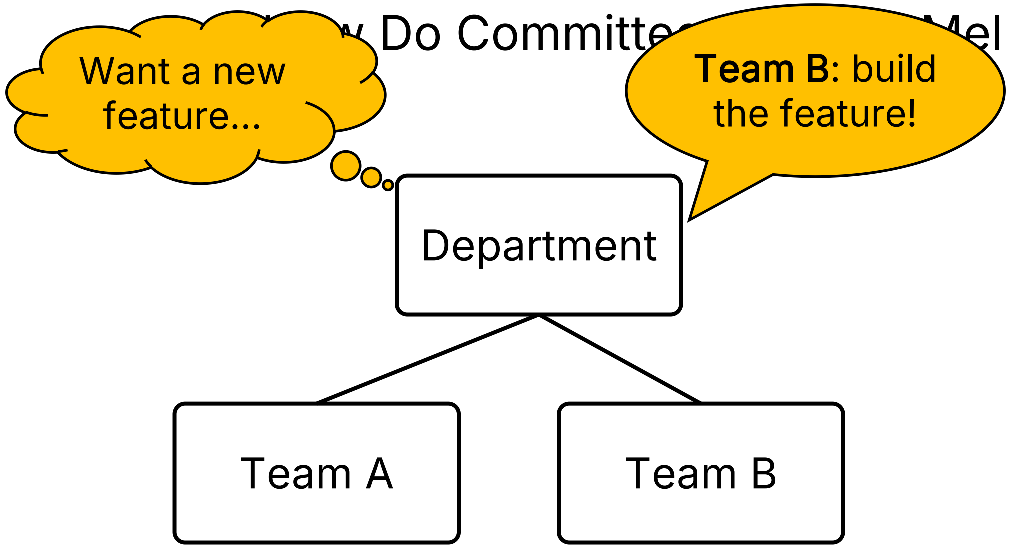
- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”

w Do Committe el Conway, 1968



How Do Committees Invent?

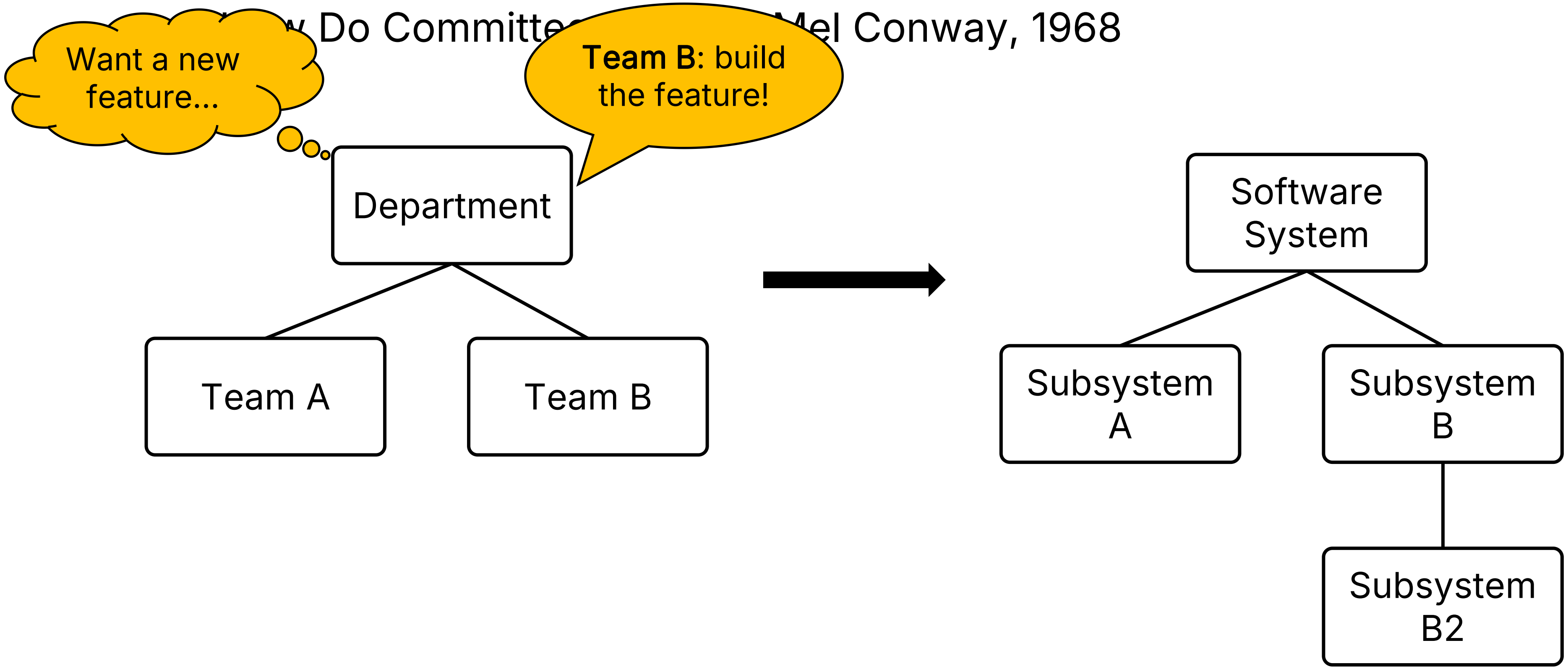
- "…there is a very close relationship between the structure of a system and the structure of the organization which designed it."
  - w Do Committe el Conway, 1968



How Do Committees Invent?

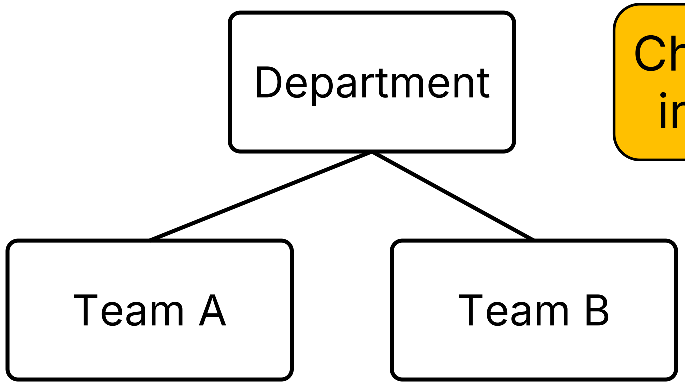
- “…there is a very close relationship between the structure of a system and the structure of the organization which designed it.”

v Do Committe el Conway, 1968



How Do Committees Invent?

- “...there is a very close relationship between the structure of a system and the structure of the organization which designed it.”
  - How Do Committees Invent? Mel Conway, 1968



Change in requirements: increase in complexity

Software Engineering

- Maintenance is forever

Software Engineering

- Maintenance is forever
    - Requirements will grow

Software Engineering

- Maintenance is forever
  - Requirements will grow
  - “Owners” of the codebase change

Software Engineering

- Maintenance is forever
  - Requirements will grow
  - “Owners” of the codebase change
  - Implementation will grow

Software Engineering

- Maintenance is forever
  - Requirements will grow
  - “Owners” of the codebase change
  - Implementation will grow

Increase in complexity

Quick Break

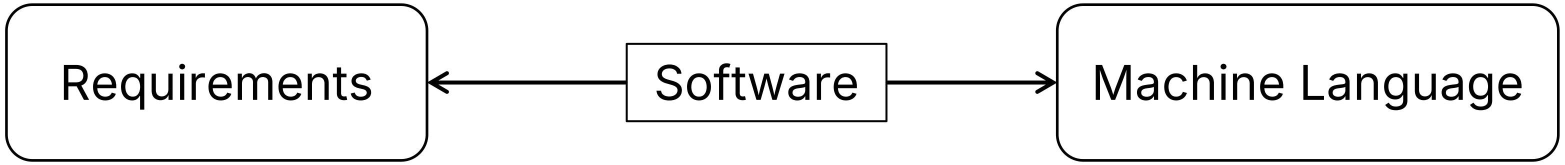
Programming as Theory-Building

- “…programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.”
    - Programming as Theory-Building, Peter Naur (of Backus-Naur fame), 1985

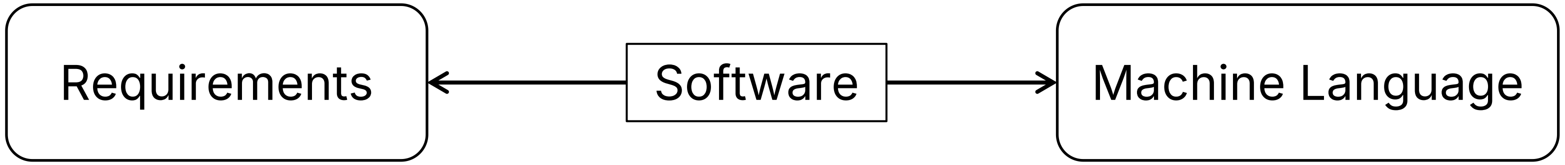
Programming as Theory-Building

- "…programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts."
    - Programming as Theory-Building, Peter Naur (of Backus-Naur fame), 1985
- Code communicates your understanding of the mapping between requirements and the programming constructs available to you

Programming as Theory-Building

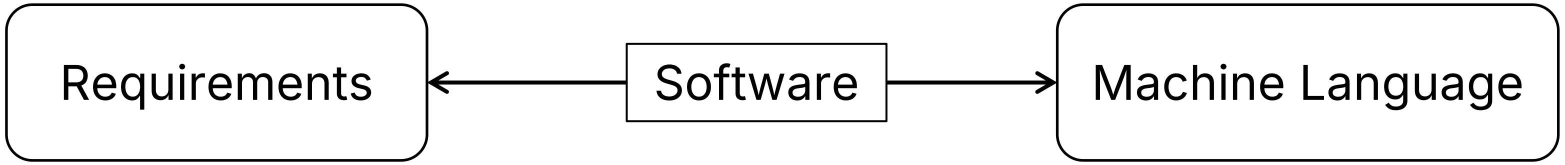


Programming as Theory-Building

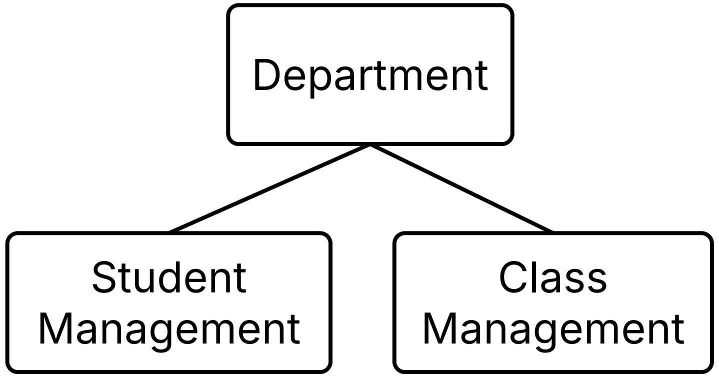


“Find students
who share classes”

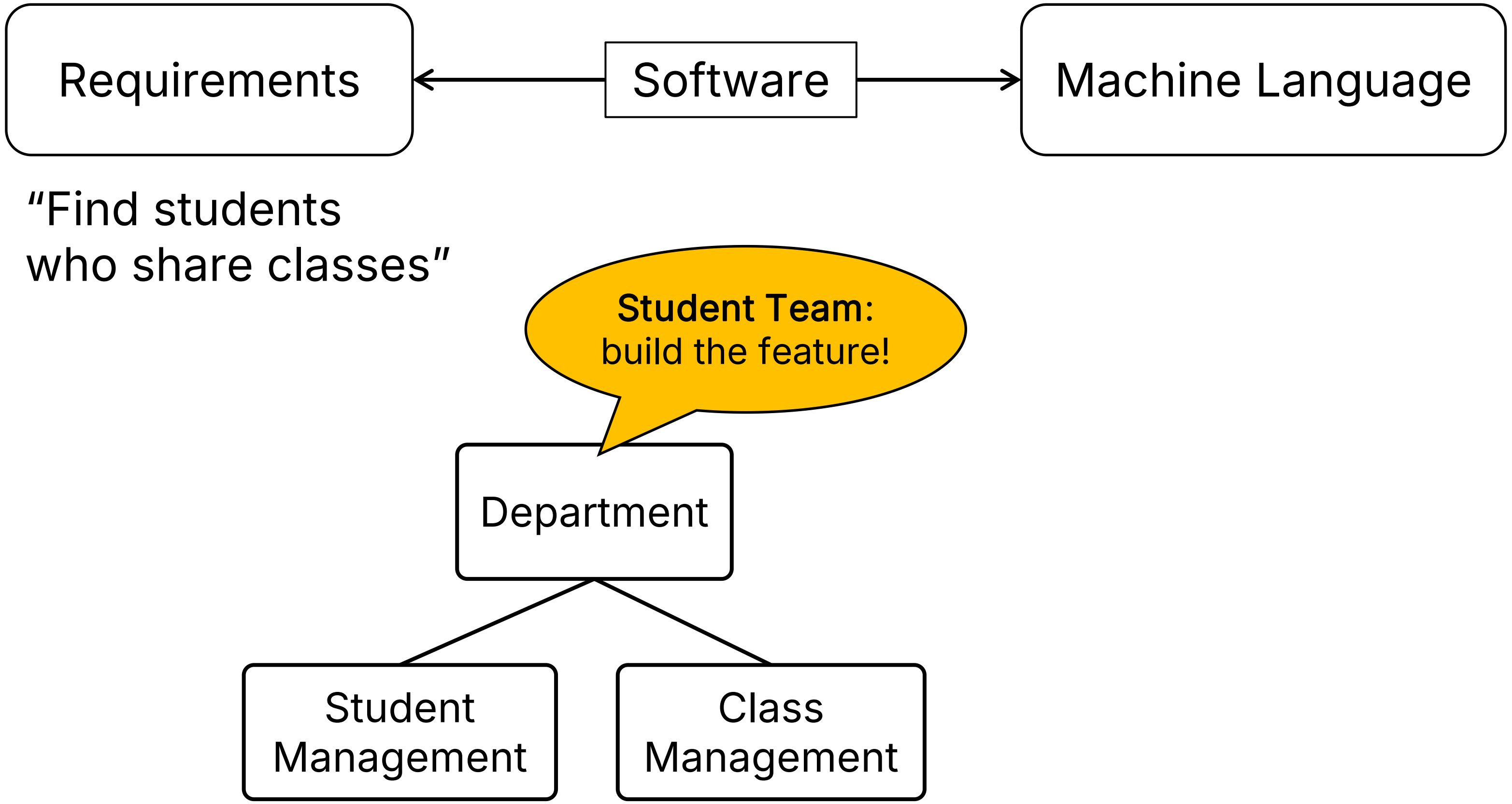
Programming as Theory-Building



“Find students
who share classes”



Programming as Theory-Building



Requirements ← Software → Machine Language

“Find students
who share classes”

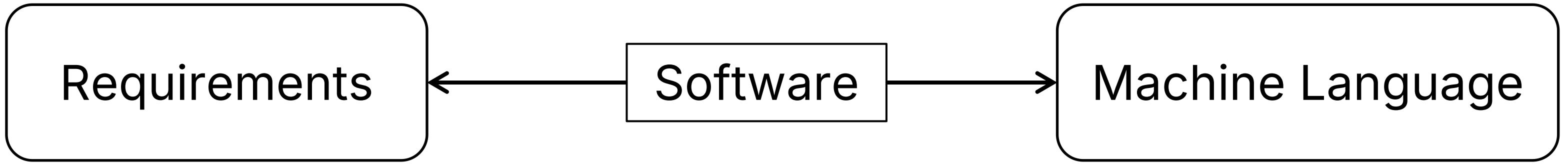
Student Team:
build the feature!

Department

Student Management

Class Management

Programming as Theory-Building

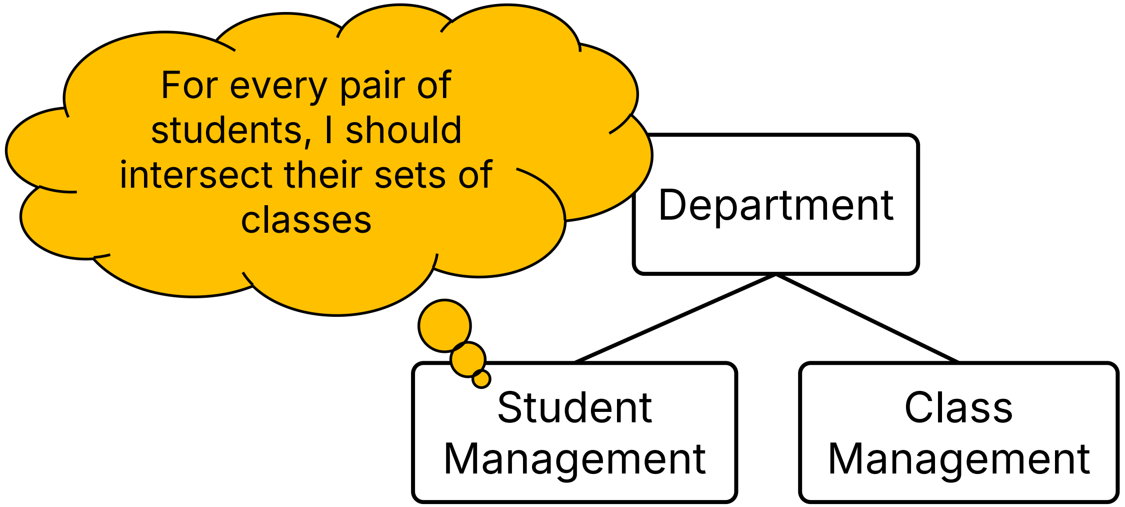


Requirements

Software

Machine Language

“Find students
who share classes”



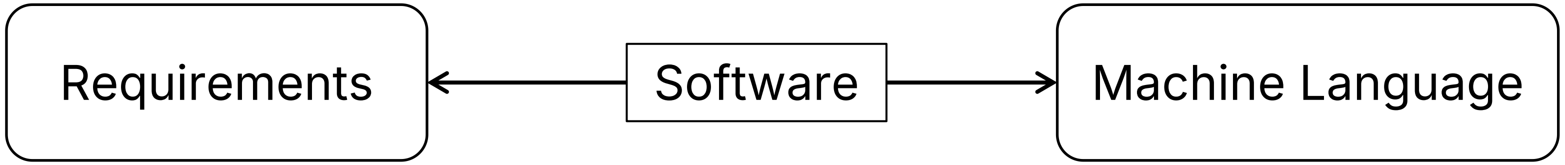
For every pair of students, I should intersect their sets of classes

Department

Student Management

Class Management

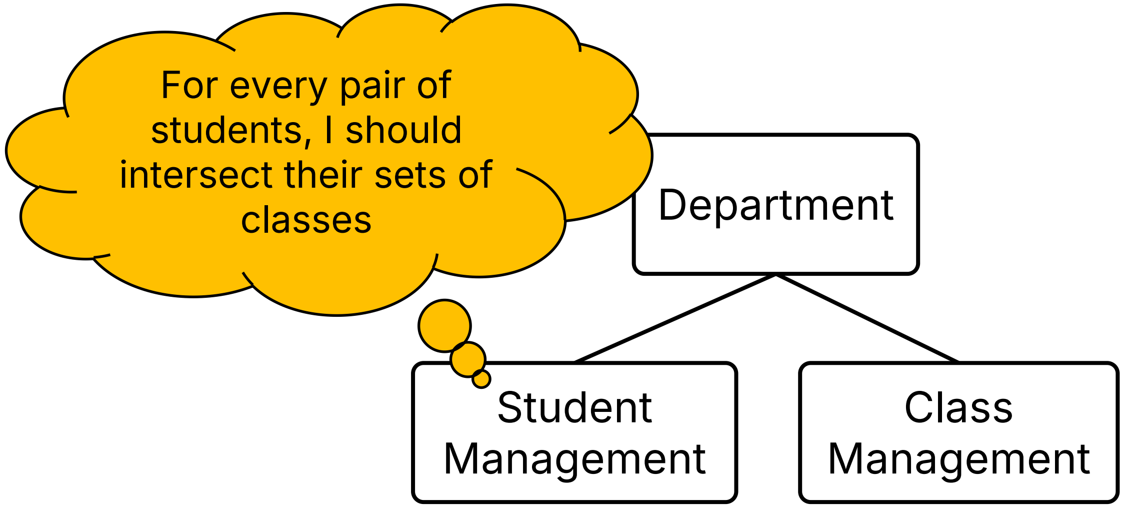
Programming as Theory-Building


Requirements

Software

Machine Language

“Find students
who share classes”


For every pair of students, I should intersect their sets of classes

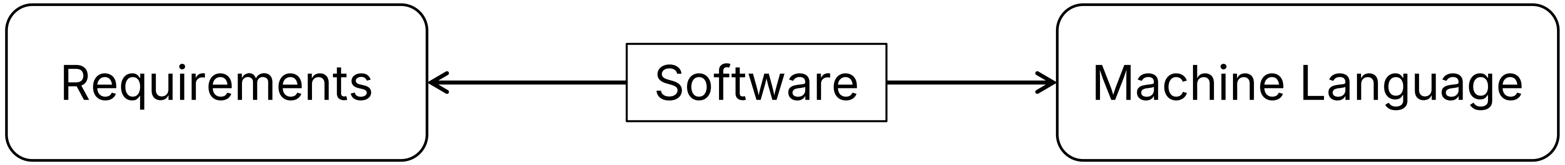
Department

Student Management

Class Management

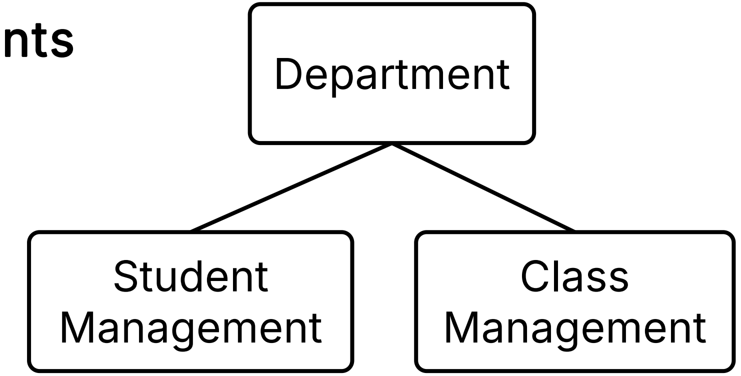
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

Programming as Theory-Building



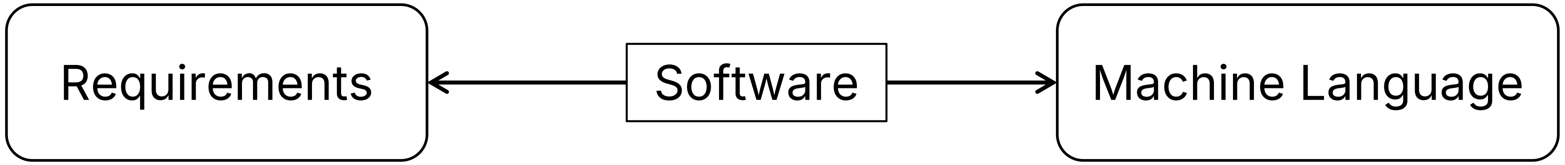
“Find students
who share classes”

“I also want to list all classes that have students enrolled”



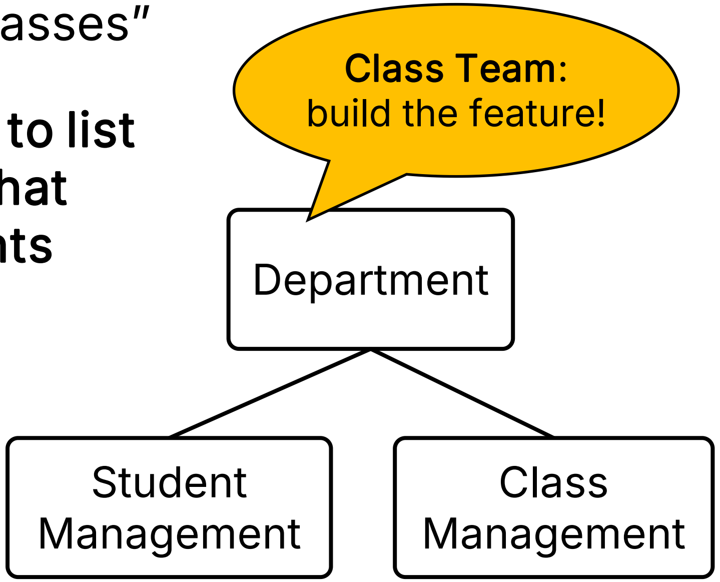
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

Programming as Theory-Building



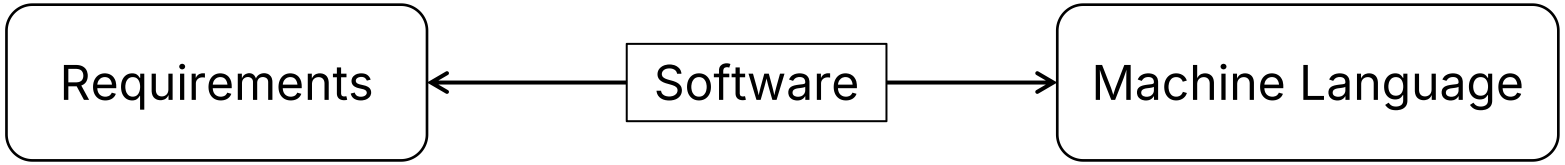
“Find students
who share classes”

“I also want to list all classes that have students enrolled”



```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

Programming as Theory-Building



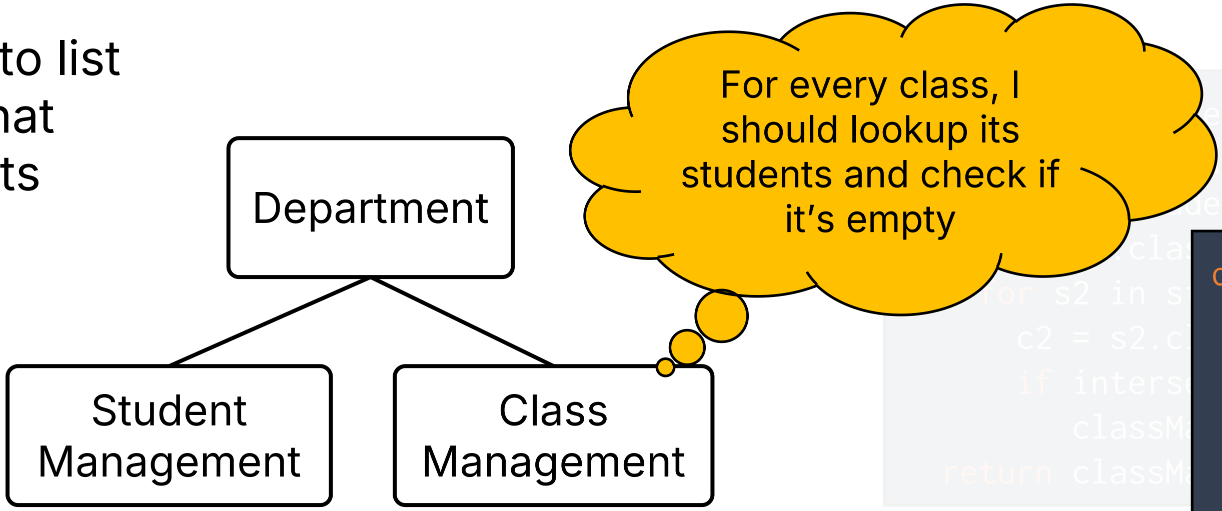
Requirements

Software

Machine Language

“Find students who share classes”

“I also want to list all classes that have students enrolled”



Department

Student Management

Class Management

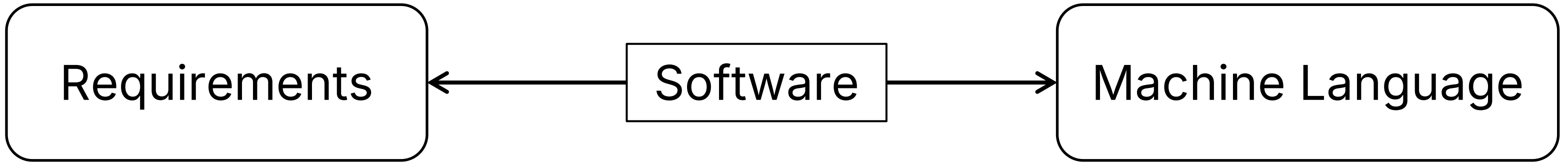
For every class, I should lookup its students and check if it’s empty

```
ePairs(students):

ents:
 cla
or s2 in s
 c2 = s2.c
 if inters
   classM
return classM
```

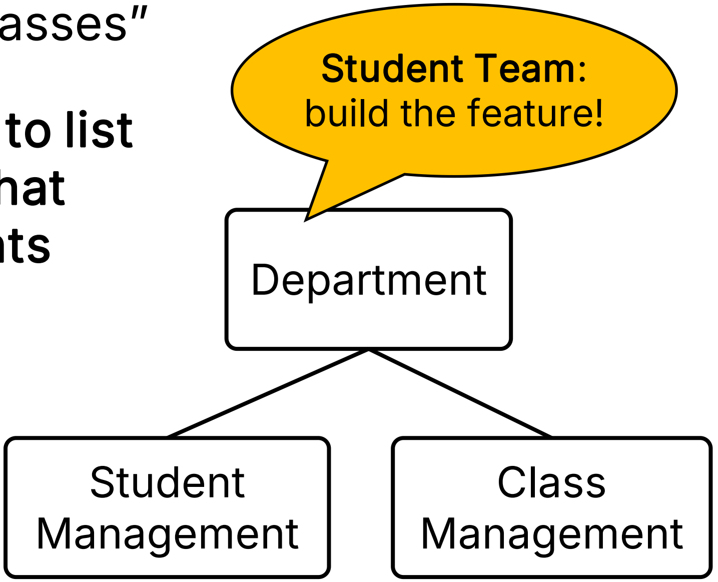
```
def findNonEmptyClasses(classes):
  activeClasses = []
  for c1 in classes:
    if len(c1.students) > 0:
      classes.append(c1)
  return classes
```

Programming as Theory-Building



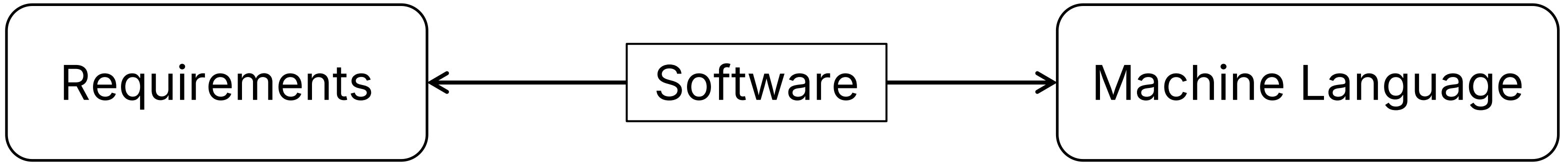
"Find students who share classes"

"I also want to list all classes that have students enrolled"



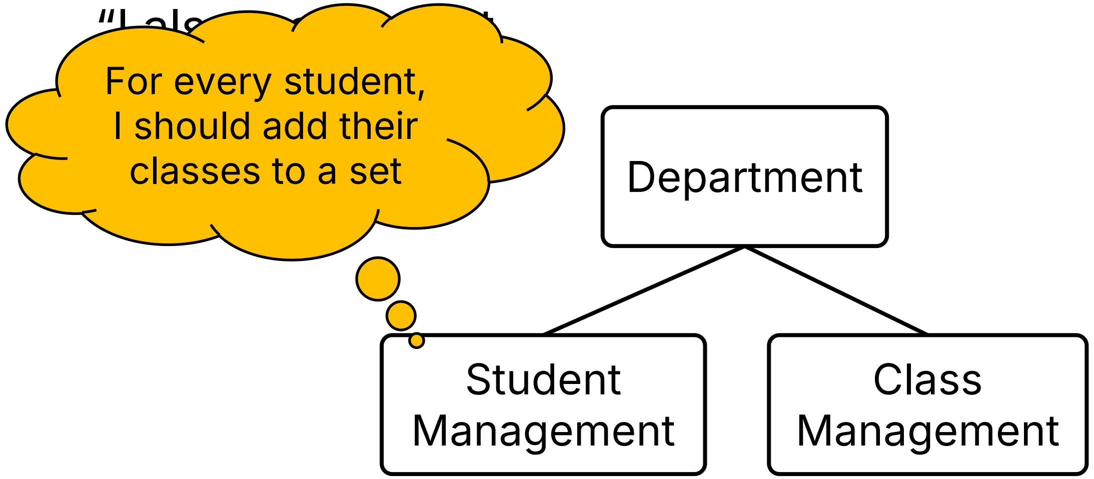
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

Programming as Theory-Building



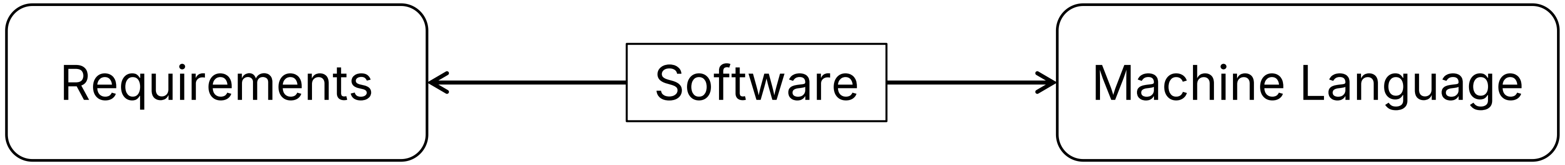
“Find students
who share classes”

“I als



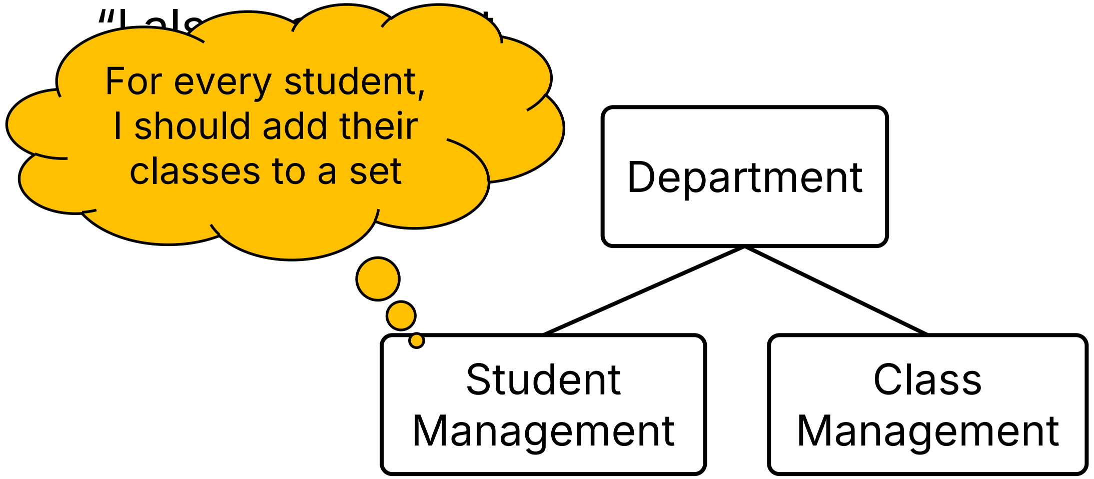
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

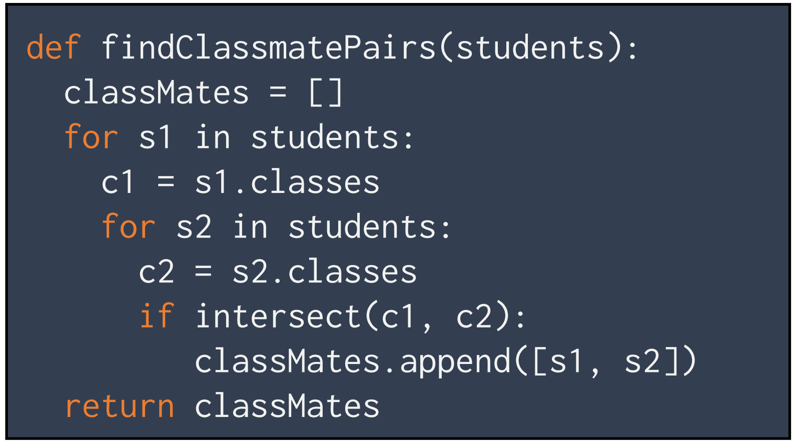
Programming as Theory-Building



“Find students
who share classes”

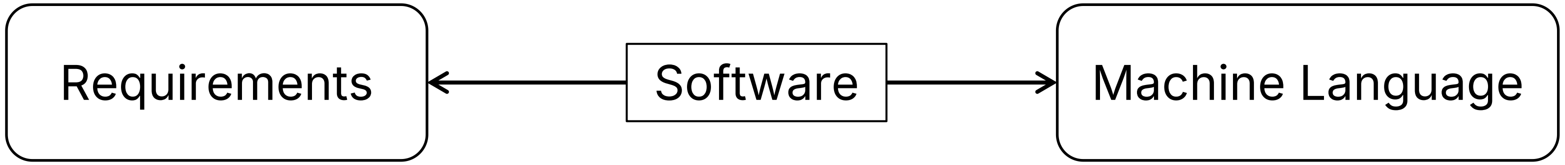
“I als





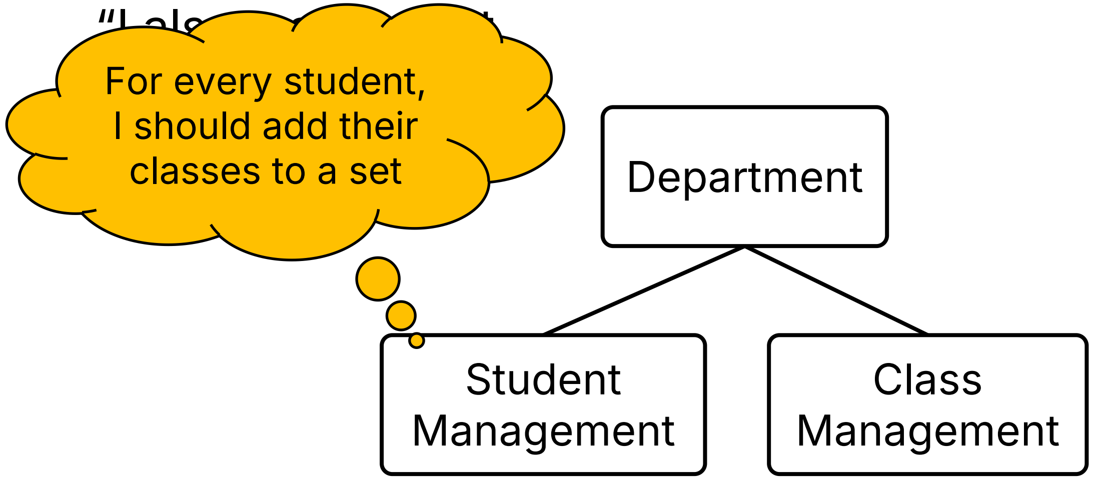
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

Programming as Theory-Building



“Find students
who share classes”

“I als



```
activeClasses = set()
classMates = set()
def buildIndex(students):
  for s1 in students:
    c1 = s1.classes
    activeClasses.add(c1)
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.add([s1, s2])

def findClassmatePairs(students):
  return classMates

def findNonEmptyClasses(students):
  return activeClasses
```

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - “Goodness” of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program’s purpose

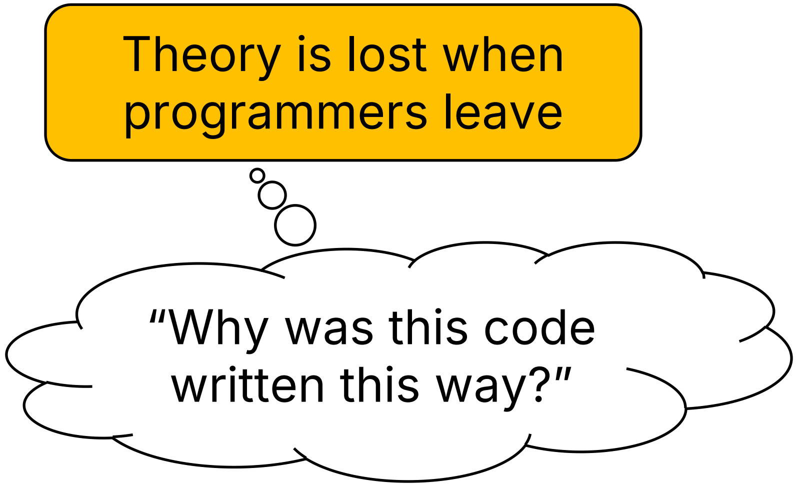
Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - “Goodness” of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program’s purpose

Theory is lost when programmers leave

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
    - “Goodness” of software is relative to other possible implementations
    - You can only imagine those implementations with a theory of the program’s purpose



Theory is lost when programmers leave

“Why was this code written this way?”

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - "Goodness" of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program's purpose

Theory is lost when programmers leave

Theory changes alongside requirements

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - “Goodness” of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program’s purpose

- The more accurate your theory, the better your code can be

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - “Goodness” of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program’s purpose

- The more accurate your theory, the better your code can be
  - Converse is also true!

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
    - “Goodness” of software is relative to other possible implementations
    - You can only imagine those implementations with a theory of the program’s purpose

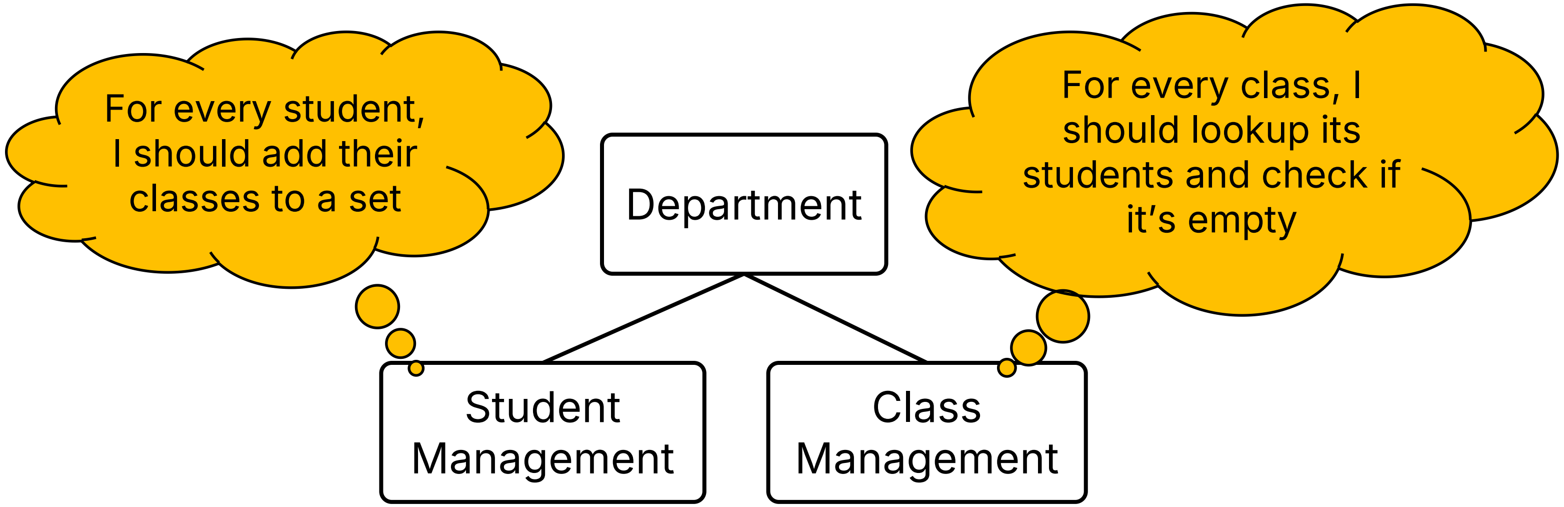
- The more accurate your theory, the better your code can be
    - Converse is also true!

- The larger the codebase, the harder to have an accurate theory

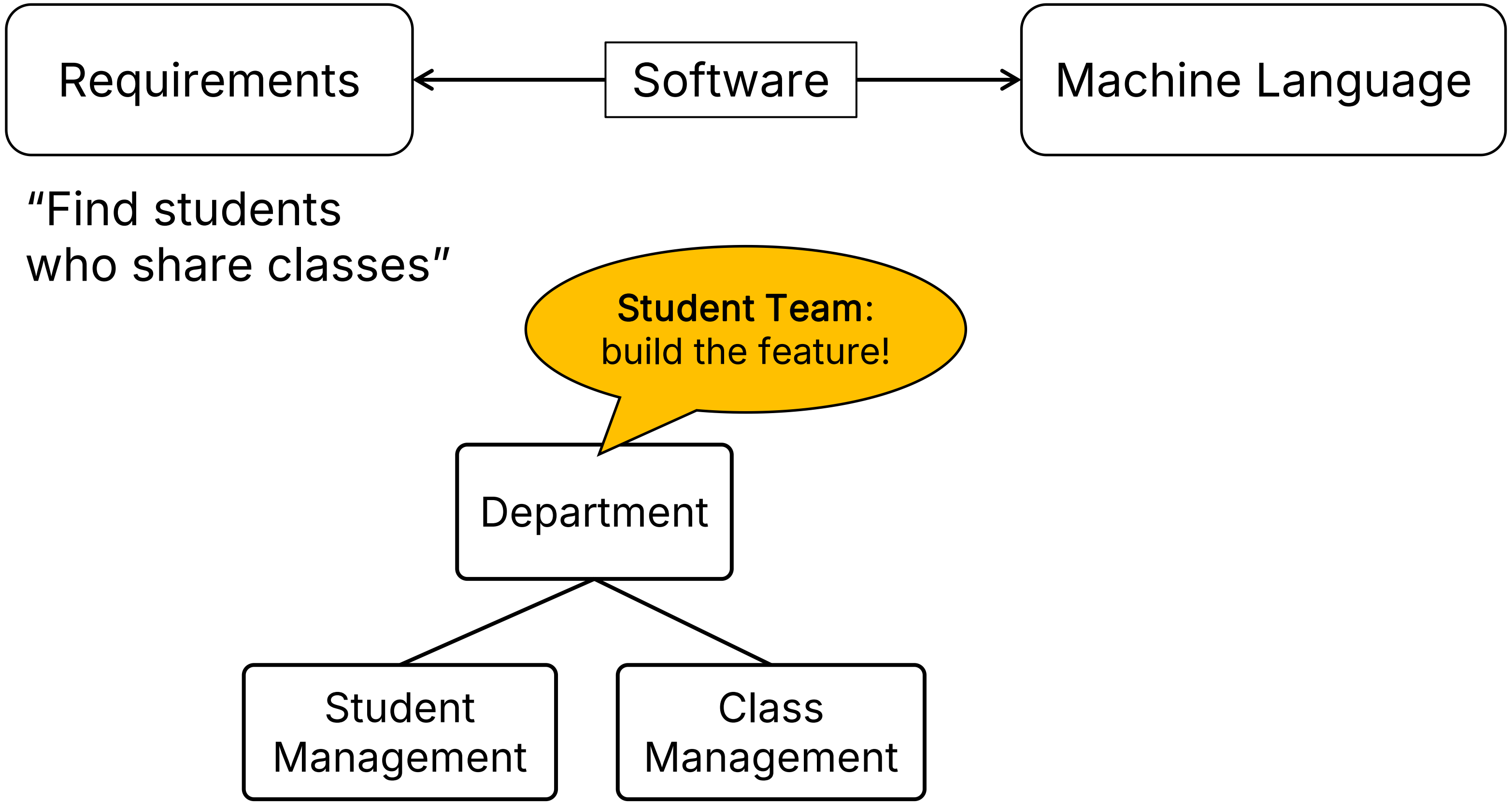
Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
    - “Goodness” of software is relative to other possible implementations
    - You can only imagine those implementations with a theory of the program’s purpose
- The more accurate your theory, the better your code can be
    - Converse is also true!
- The larger the codebase, the harder to have an accurate theory

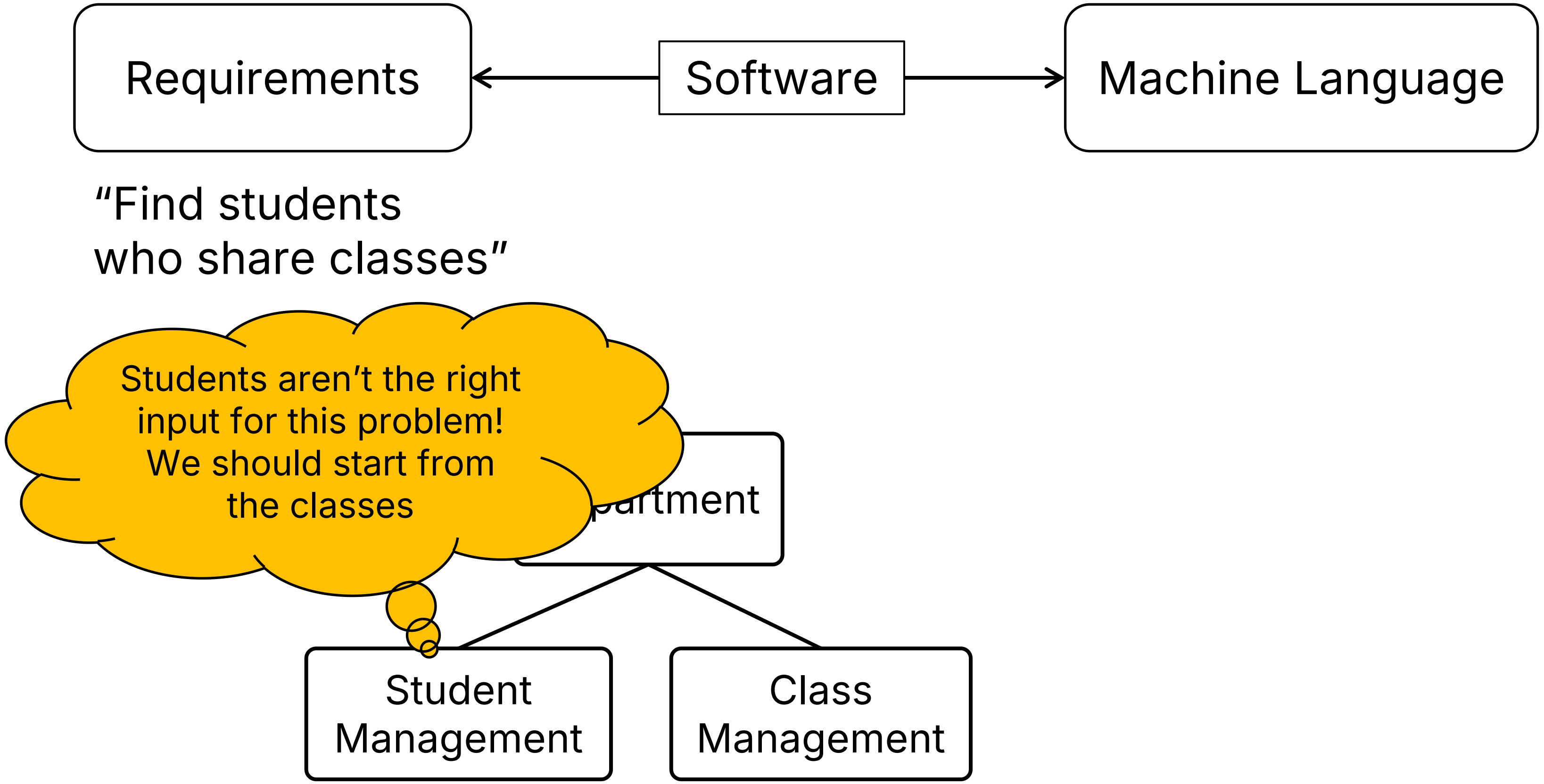
Programming as Theory-Building


For every student, I should add their classes to a set
Department
For every class, I should lookup its students and check if it's empty
Student Management
Class Management

Programming as Theory-Building



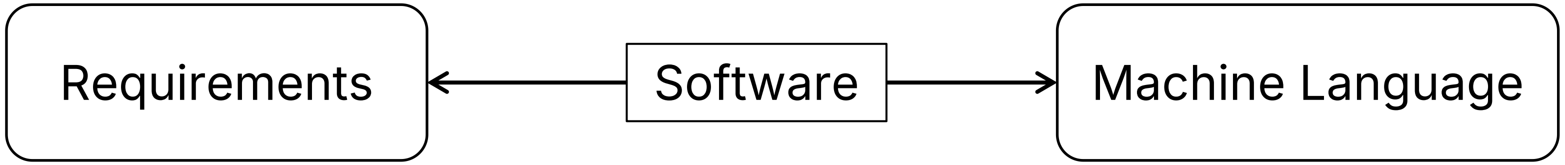
Programming as Theory-Building



"Find students who share classes"

Students aren't the right input for this problem! We should start from the classes

Programming as Theory-Building

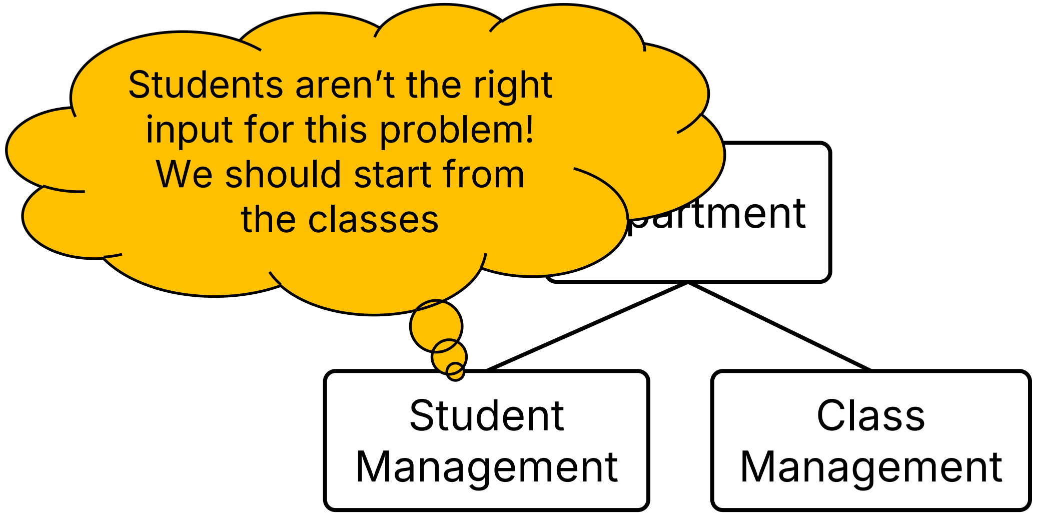


Requirements

Software

Machine Language

“Find students
who share classes”



Students aren’t the right
input for this problem!
We should start from
the classes

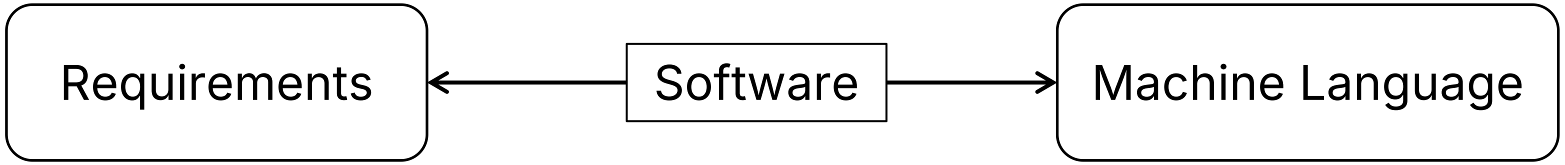
partment

Student
Management

Class
Management

```
def findClassmatePairs(classes):
  classMates = set()
  for c in classes:
    for s1, s2 in zip(c.students, c.students):
      classMates.add([s1, s2])
  return classMates
```

Programming as Theory-Building

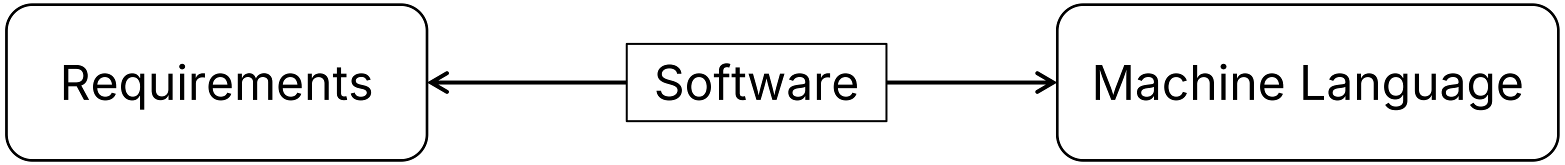


“Find students
who share classes”

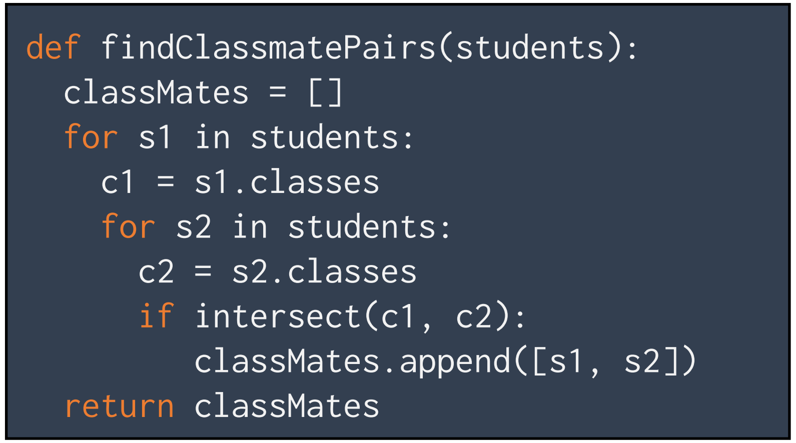
```
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

```
def findClassmatePairs(classes):
  classMates = set()
  for c in classes:
    for s1, s2 in zip(c.students, c.students):
      classMates.add([s1, s2])
  return classMates
```

Programming as Theory-Building

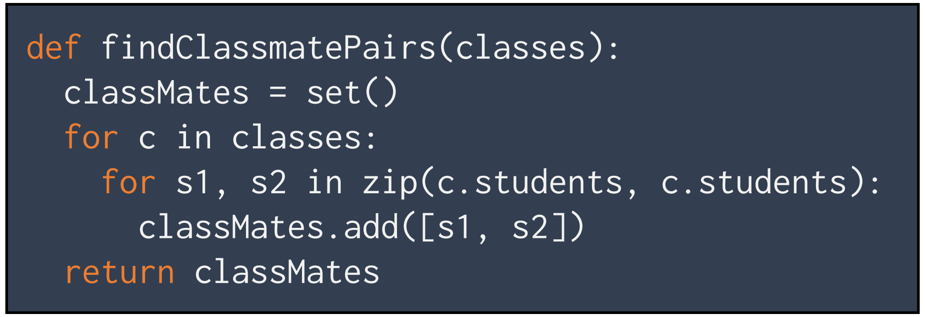


“Find students
who share classes”



```python
def findClassmatePairs(students):
  classMates = []
  for s1 in students:
    c1 = s1.classes
    for s2 in students:
      c2 = s2.classes
      if intersect(c1, c2):
         classMates.append([s1, s2])
  return classMates
```

$|students|^2 \cdot 2|classes|$



```python
def findClassmatePairs(classes):
  classMates = set()
  for c in classes:
    for s1, s2 in zip(c.students, c.students):
      classMates.add([s1, s2])
  return classMates
```

$|students|^2 \cdot |classes|$

Programming as Theory-Building

- For software to retain its quality as it undergoes changes, it is mandatory that each modification is grounded in the theory of the software
  - “Goodness” of software is relative to other possible implementations
  - You can only imagine those implementations with a theory of the program’s purpose
- The more accurate your theory, the better your code can be
  - Converse is also true!
- The larger the codebase, the harder to have an accurate theory
- In order to make good changes, you need access to other people’s theories

Software Engineering

- Why is it difficult?

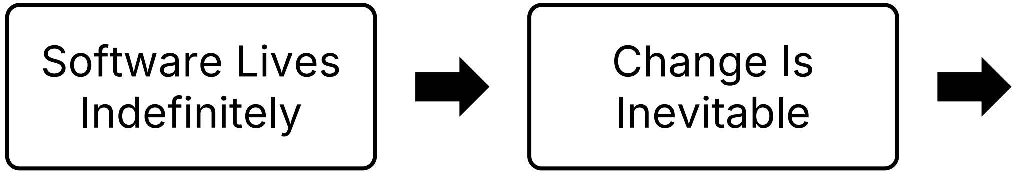
Software Engineering

- Why is it difficult?

Software Lives Indefinitely

Software Engineering

- Why is it difficult?

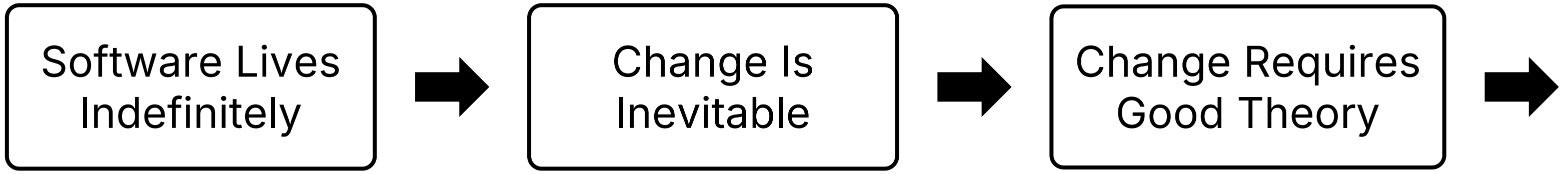


Software Lives Indefinitely

Change Is Inevitable

Software Engineering

- Why is it difficult?



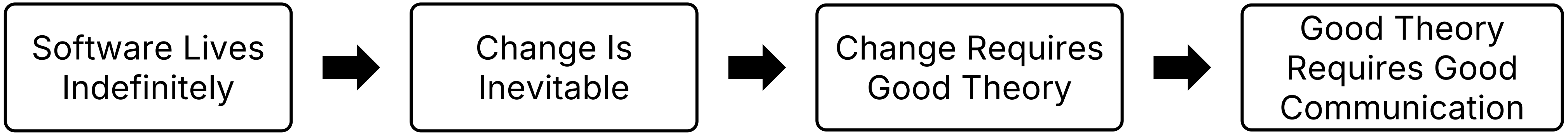
Software Lives Indefinitely

Change Is Inevitable

Change Requires Good Theory

Software Engineering

- Why is it difficult?



Software Lives Indefinitely

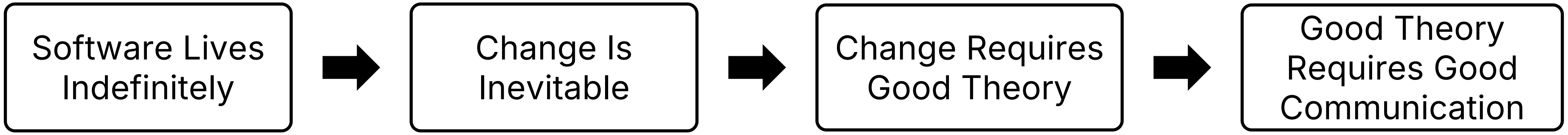
Change Is Inevitable

Change Requires Good Theory

Good Theory Requires Good Communication

Software Engineering

- Why is it difficult?



Software Lives Indefinitely

Change Is Inevitable

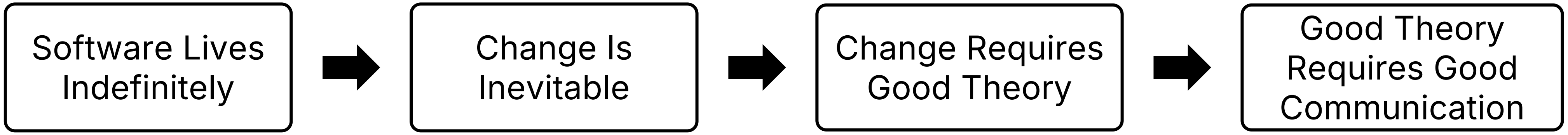
Change Requires Good Theory

Good Theory Requires Good Communication

- “Communication”:
    - conversations
    - code
    - documentation

Software Engineering

- Why is it difficult?



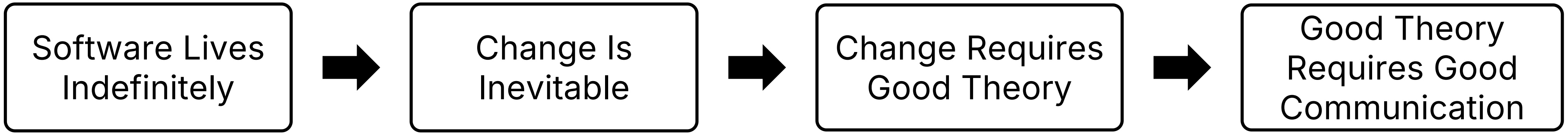
Software Lives Indefinitely ➡ Change Is Inevitable ➡ Change Requires Good Theory ➡ Good Theory Requires Good Communication

- "Communication":
  - ~~conversations~~
  - code
  - documentation

People Leave

Software Engineering

- Why is it difficult?



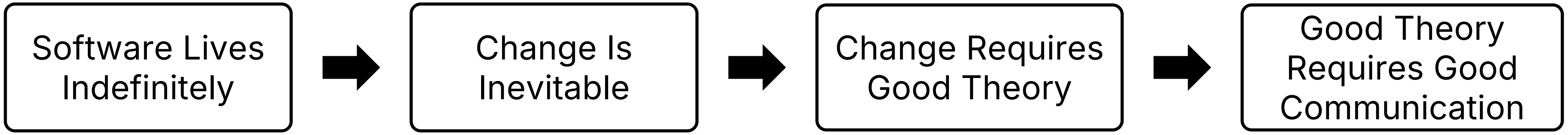
Software Lives Indefinitely → Change Is Inevitable → Change Requires Good Theory → Good Theory Requires Good Communication

- “Communication”:
  - ~~conversations~~
  - ~~code~~
  - documentation

Code Remains After Requirements Change

Software Engineering

- Why is it difficult?


Software Lives Indefinitely
Change Is Inevitable
Change Requires Good Theory
Good Theory Requires Good Communication

- “Communication”:
  - ~~conversations~~
  - ~~code~~
  - ~~documentation~~

Doesn’t Match

Why Study Software Engineering?

Why Study Software Engineering?

- To learn how to develop theories of programs

Why Study Software Engineering?

- To learn how to develop theories of programs
  - Theory-Building applies to more than just software systems
  - Applies to organizing any kind of information in your head
  - Essentially: how to nail down precise details

Why Study Software Engineering?

- To learn how to develop theories of programs
  - Theory-Building applies to more than just software systems
  - Applies to organizing any kind of information in your head
  - Essentially: how to nail down precise details

- To make racks
  - A good dev can make $250k+ after less than a decade of experience

Why Study Software Engineering?

- To learn how to develop theories of programs
  - Theory-Building applies to more than just software systems
  - Applies to organizing any kind of information in your head
  - Essentially: how to nail down precise details

- To make racks
  - A good dev can make $250k+ with less than a decade of experience

- To make the world a better place
  - Software has transformative power in our society, for good or ill
  - If you don’t chase money, there are lots of opportunities for good