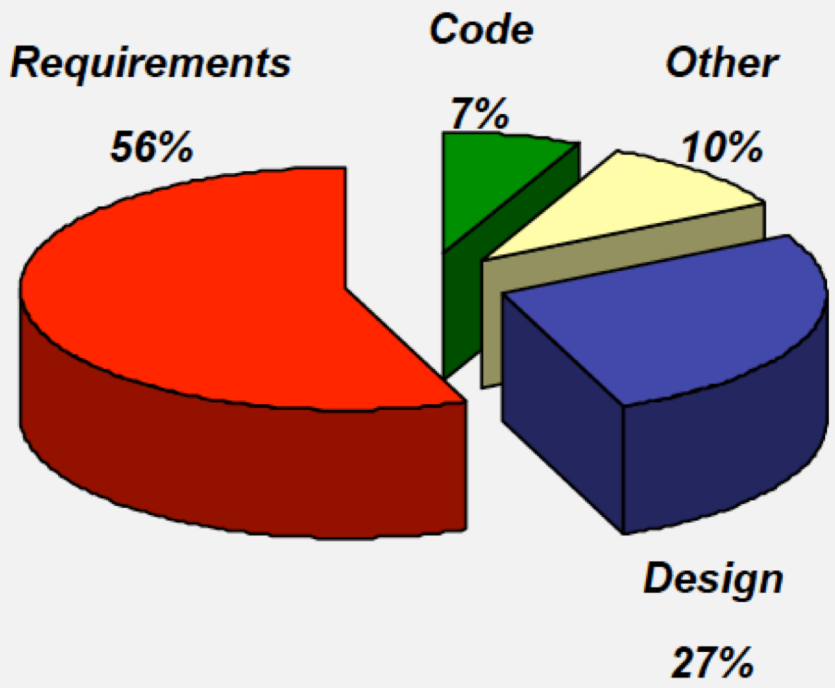
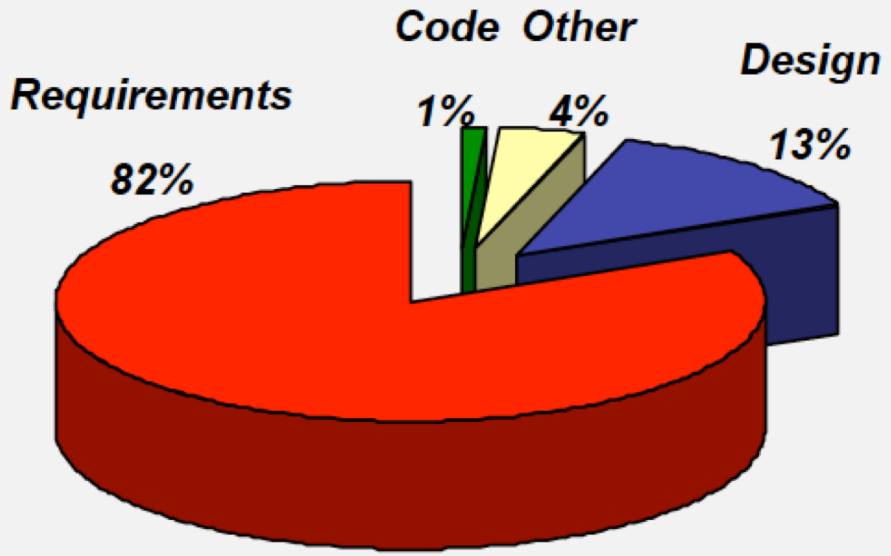
What is Design?

- Why Use Eiffel in a Software Design Course?

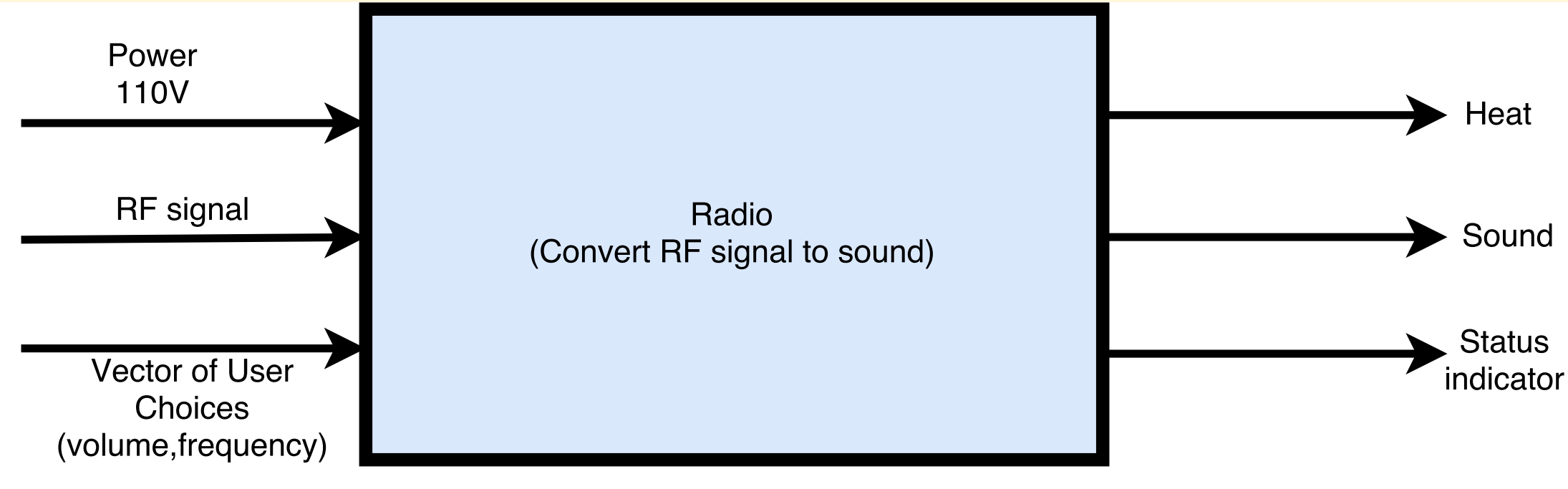
Distribution of Defects



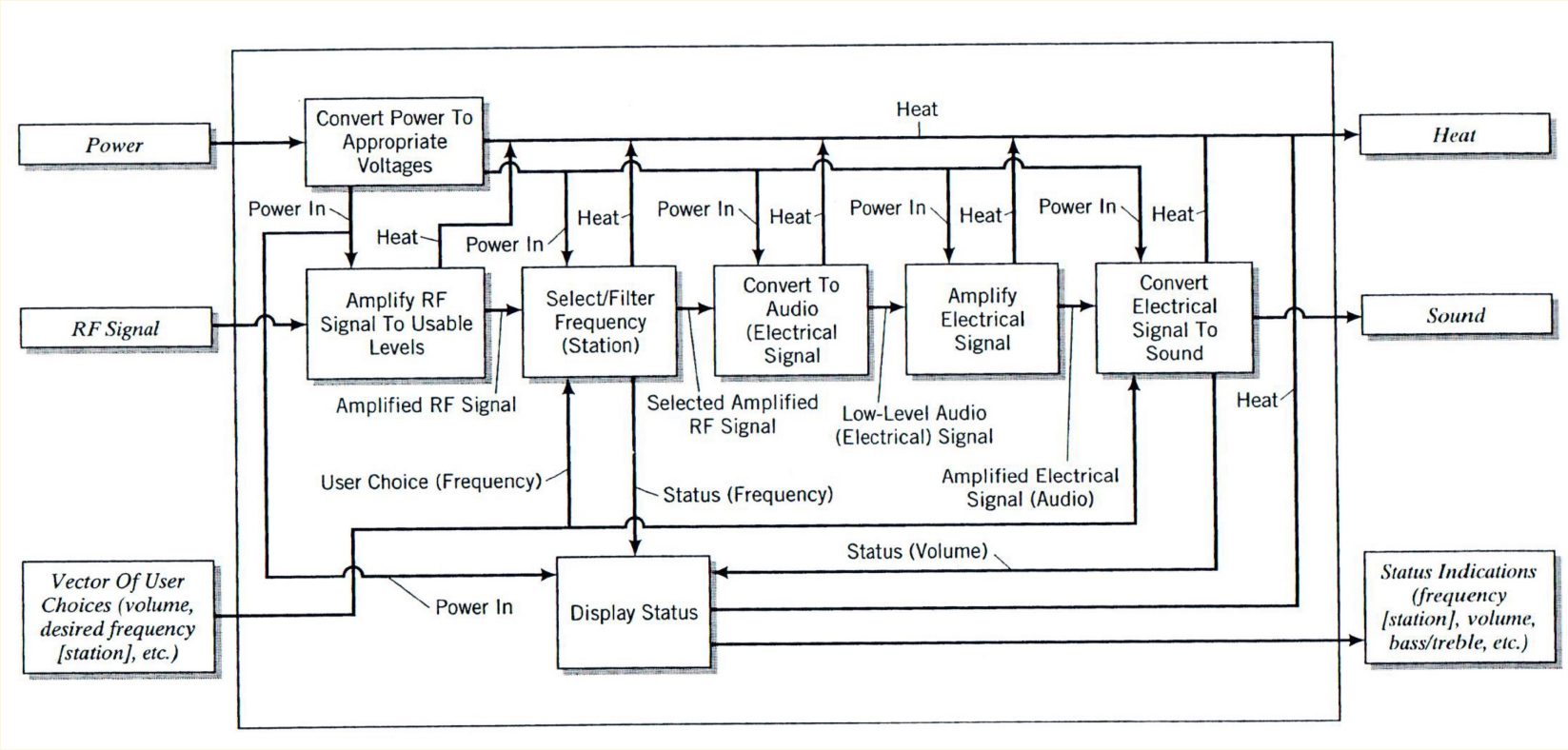
Distribution of Effort to Fix Defects



What is Design?

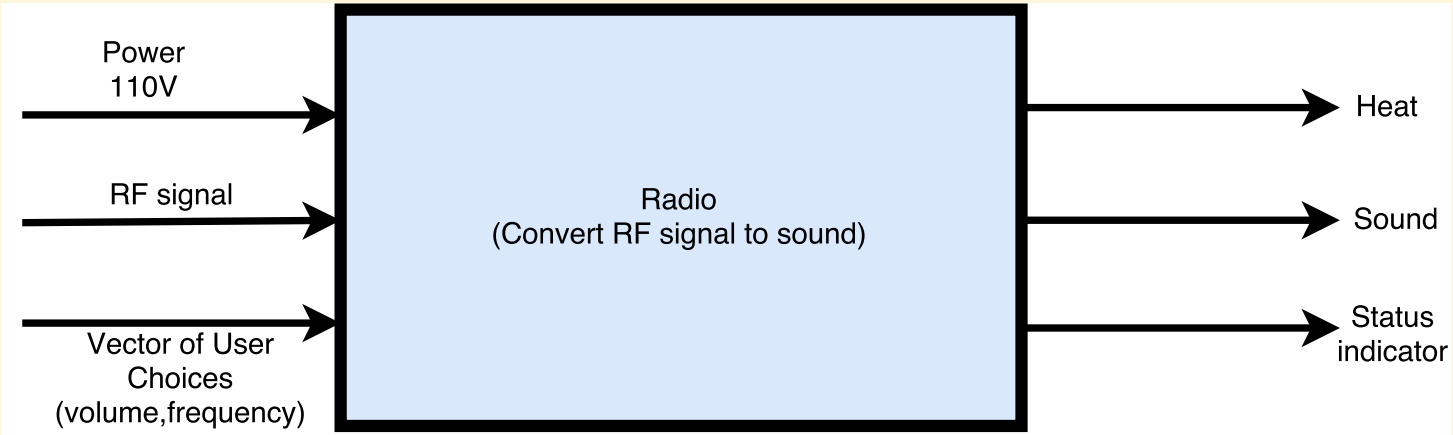


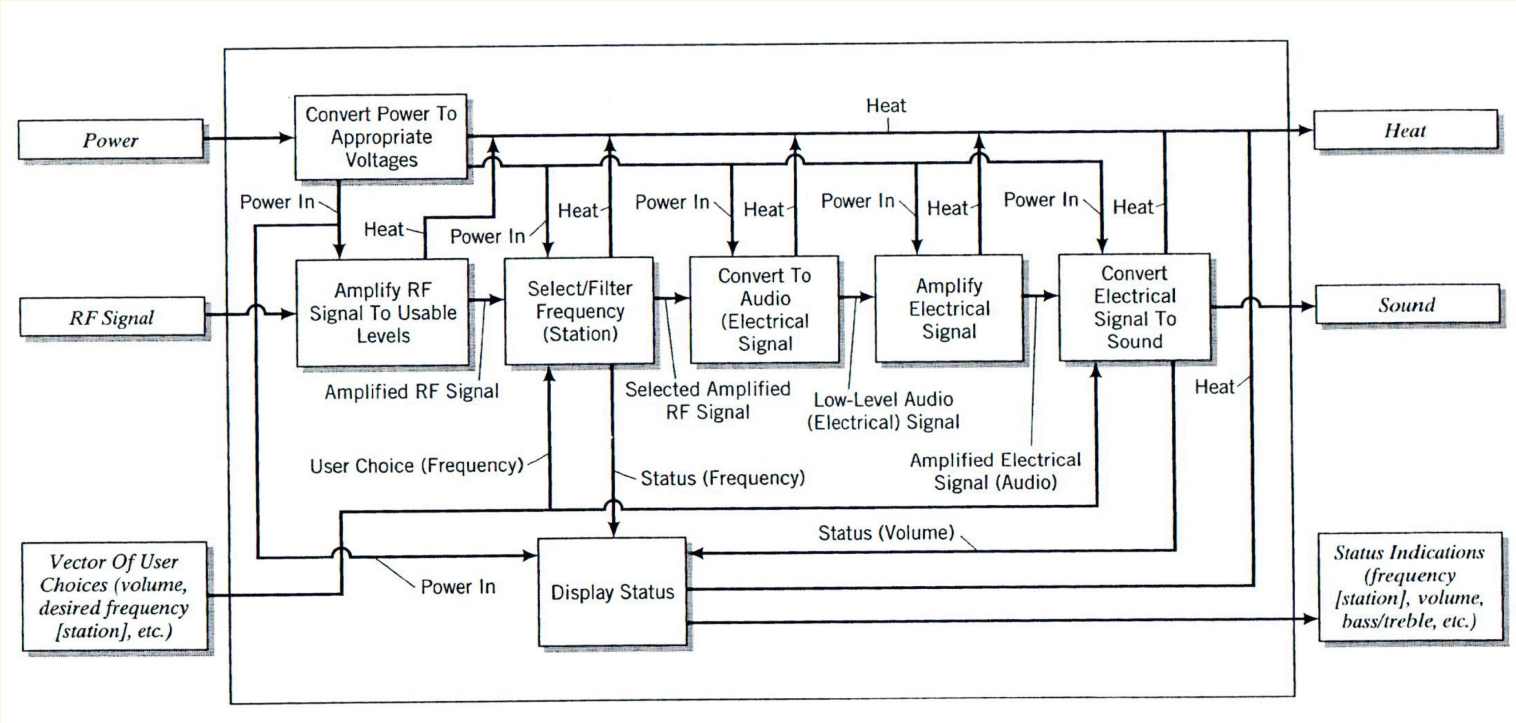
What is Design?



What is Design?

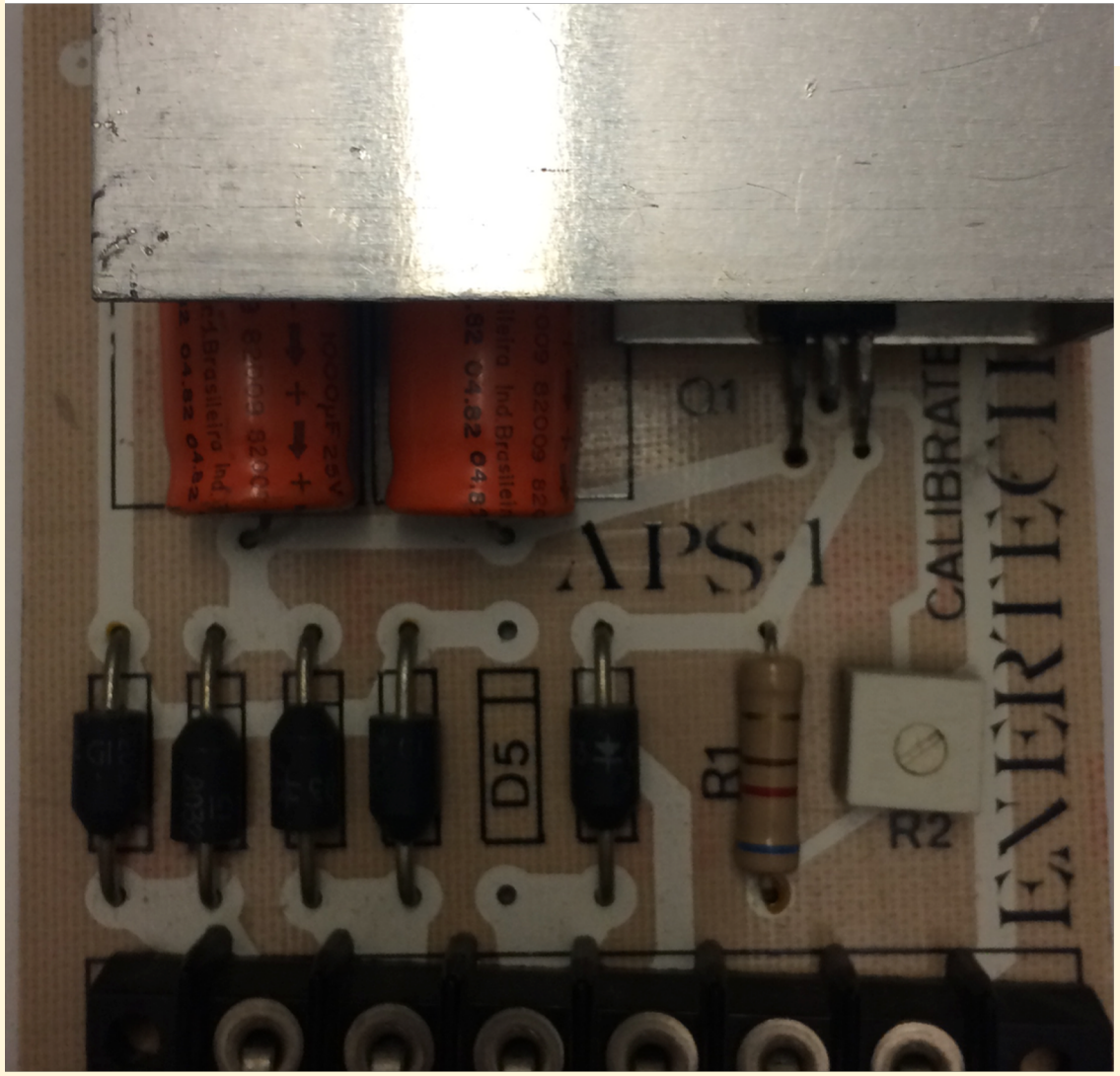
Specification





Architecture (Blueprint)

Power supply from alarm system circa 2002

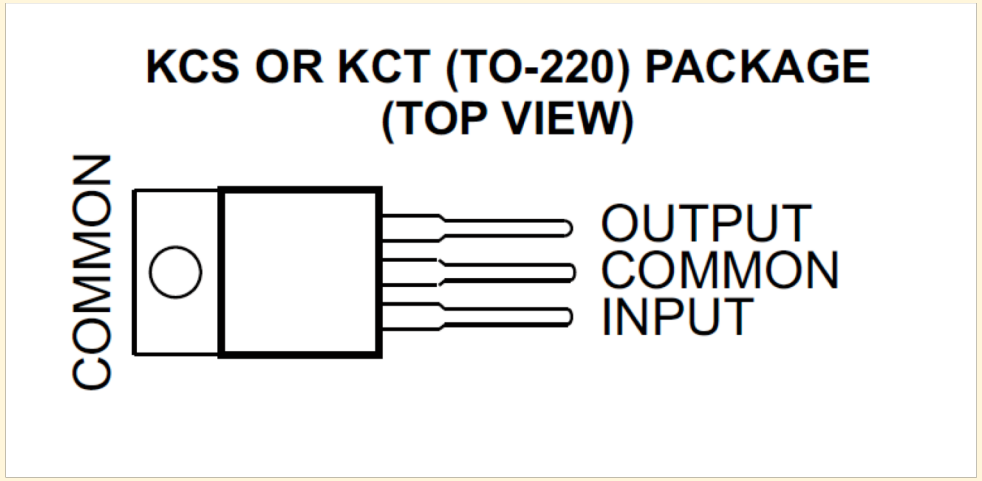


Components of power system

- Rectification diodes in a bridge design
- Resistors
- 2 big capacitors
- Under the heat sink is a converter
- Transformer from 120 Volts AC to 12 Volts AC
- Input is 12 volts AC and the output is a regulated 5 Volts DC
- Each component (even resistors) has a specification sheet (e.g. R = V/I)

Regulator Specification Sheet A7805

Input Voltage	7 -25 Volts DC
Output Voltage	4.75 – 5.25 Volts DC (0C to 125C)



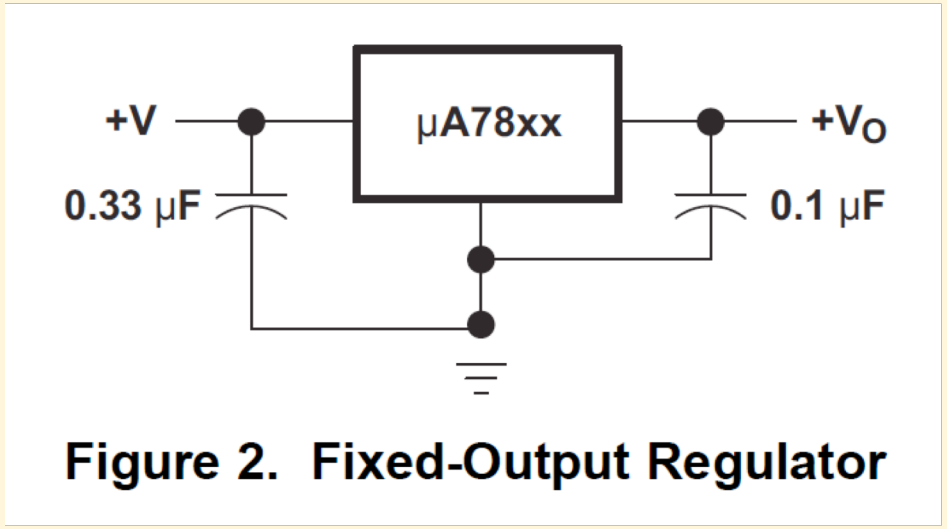


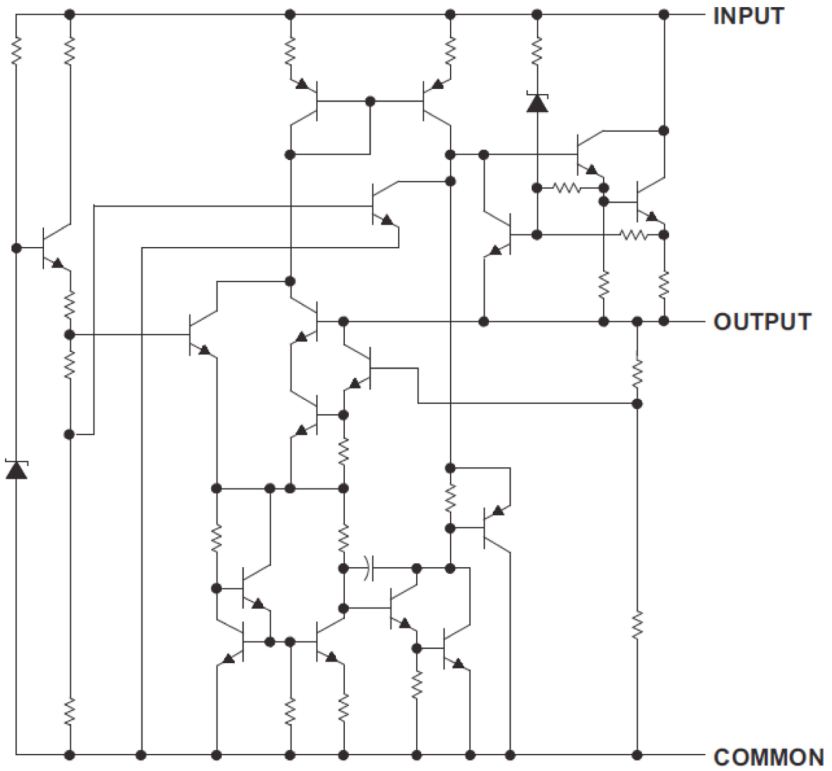
Figure 2. Fixed-Output Regulator

8 Detailed Description

8.1 Overview

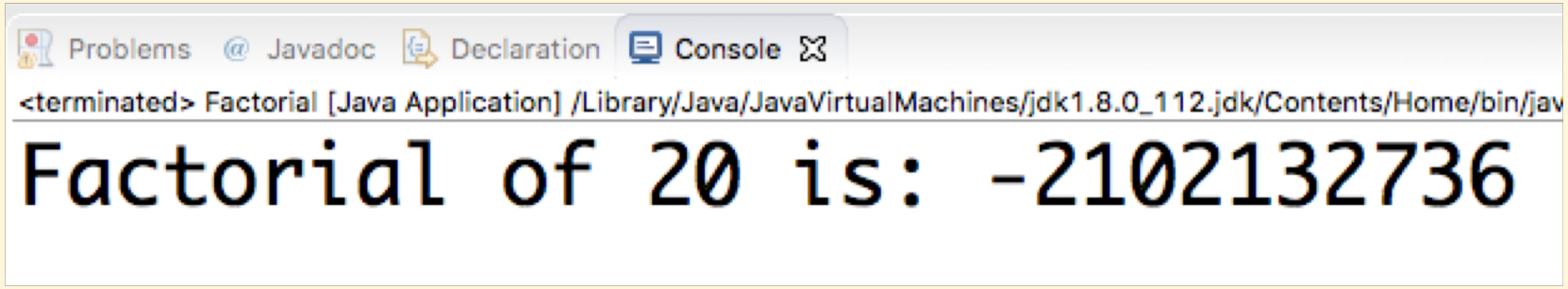
This series of fixed-voltage integrated-circuit voltage regulators is designed for a wide range of applications. These applications include on-card regulation for elimination of noise and distribution problems associated with single-point regulation. Each of these regulators can deliver up to 1.5 A of output current. The internal current-limiting and thermal-shutdown features of these regulators essentially make them immune to overload. In addition to use as fixed-voltage regulators, these devices can be used with external components to obtain adjustable output voltages and currents, and also can be used as the power-pass element in precision regulators.

8.2 Functional Schematic



Given the specification of each component, we can design and use Mathematics to predict the overall behaviour of the system of components

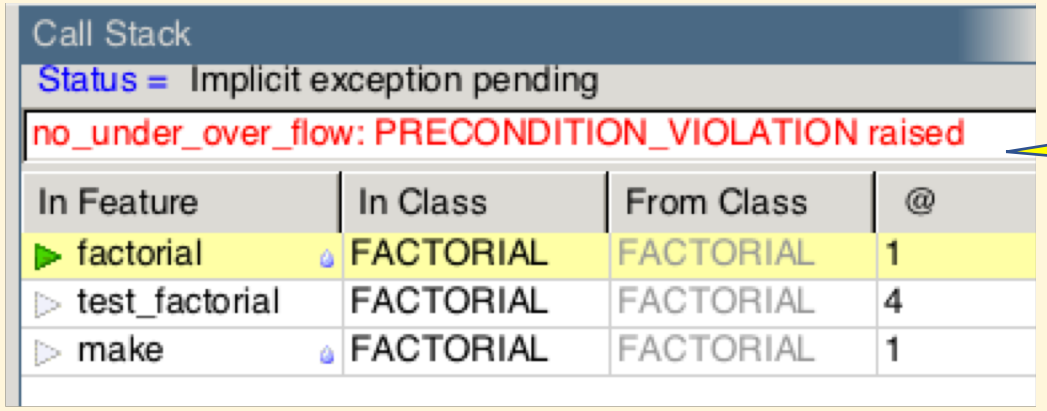
Where are the Specification Sheets and Blueprints for Software Components?




```
static int factorial(int n)
{
    int output;
    if(n==1){
      return 1;
    }
    //Recursion: Function calling itself!!
    output = factorial(n-1)* n;
    return output;
}
```

Precondition needed

```
factorial(n:INTEGER): INTEGER
        require
                no_under_over_flow:
                    0 <= n and n <= 12
        do
                if n = 0 then
                        Result := 1
                else
                        Result := n*factorial(n-1)
                end
        ensure
                correct_result:
                    -- Result = (∏i ∈ 1..n: i)
        end
```



Runtime Assertion Checking

```
factorial(n:INTEGER): INTEGER
    require
        no_under_over_flow: 0 <= n and n <= 12
    do
        if n = 0 then
            Result := 1
        else
            Result := n*factorial(n-1)
        end
    ensure
        correct_result:          -- (lambda i ∈ INT: i)
        Result = product(1, n, agent
                                (i:INTEGER):INTEGER
                                do Result := i end)
    end
```

Functional programming

```
product(i,n: INTEGER; f: FUNCTION[INTEGER]):
    INTEGER
    do
        Result := 1
        across i |..| n as k  loop
            Result := Result * f(k.item)
        end
    ensure
        -- Result = f(i) * f(i+1)*...*f(n)
    end
```

Hard to do contracts in Java

- Comments
  (not enforced)
- Assertions
  (must be enabled & do not appear in the interface documentation)
- Exceptions
  (the negation of what the client needs & requires defensive coding with code bloat)

Declare Exception

Extra code

Postcondition?

```
public class Factorial {
    public static void main(String[] args) {
        System.out.println(factorial(13));
    }

    /**
     * <p> i <= 12
     * @param i an integer
     * @return factorial of i
     */
    public static int factorial(int i)
        throws IllegalArgumentException {
        if(i > 12) {
            throw new IllegalArgumentException();
        }

        if(i == 0) {
            return 1;
        }
        else {
            return i * factorial(i - 1);
        }
    }
}
```

Node/Javascript left_pad fiasco

However, it is not all good news. There are many cases where code reuse has had negative effects, leading to an increase in maintenance costs and even legal action [2, 29, 35, 41]. For example, in a recent incident code reuse of a Node.js package called left-pad, which was used by Babel, caused interruptions to some of the largest Internet sites, e.g., Facebook, Netflix, and Airbnb. Many referred to the incident as the case that 'almost broke the Internet' [33, 45]. That incident lead to many heated discussions about code reuse, sparked by David Haney's blog post: *"Have We Forgotten How to Program?"* [26].

While the real reason for the left-pad incident was that *npm* allowed authors to unpublish packages (a problem which has been resolved [40]), it raised awareness of the broader issue of taking on dependencies for trivial tasks that can be easily implemented [26].

NPM & left-pad: Have We Forgot

2016-03-23 · blog · 930 words · 5 mins read

Intro

Okay developers, time to have a serious talk. As yo
and a bunch of other high-profile packages on NPI

A simple NPM package called left-pad that was a

left-pad, at the time of writing this, has 11 stars on
implement a basic left-pad string function. In case
pad package:

Code

```
1   module.exports = leftpad;
2   function leftpad (str, len, ch) {
3     str = String(str);
4     var i = -1;
5     if (!ch && ch !== 0) ch = ' ';
6     len = len - str.length;
7     while (++i < len) {
8       str = ch + str;
9     }
10    return str;
11  }
```

Code

```
1   module.exports = leftpad;
2   function leftpad (str, len, ch) {
3     str = String(str);
4     var i = -1;
```

```
) ch = ' ';
gth;
```

What concerns me here is that *so many packages and projects* took on a **dependency** for a simple left padding string function, rather than their developers taking 2 minutes to write such a basic function themselves.

As a result of learning about the left-pad disaster, I started investigating the NPM ecosystem. Here are some of the things that I observed:

- There's a package called isArray that has 880,000 downloads a day, and 18 million downloads in February of 2016. It has **72 dependent NPM packages**. Here's its **entire 1 line of code**:

```
return toString.call(arr) == '[object Array]';
```

Code

```
1   module.exports = leftpad;
2   function leftpad (str, len, ch) {
3     str = String(str);
4     var i = -1;
5     if (!ch && ch !== 0) ch = ' ';
6     len = len - str.length;
7     while (++i < len) {
8       str = ch + str;
9     }
10    return str;
11  }
```

In the original

No comments

No tests

Wrong outputs:

```
console.log(leftpad ('abc', 5, 345))
  // result: 345345abc

str = new Foo();
console.log(leftpad (str, 25, 'x'))
  // result: xxxxxxxxxx[object Object]
```

```
left_pad (str: STRING; len: INTEGER; ch: CHARACTER): STRING
                -- return a string of length `len`
                -- that is padded to the left with `len - str.count` characters `ch`
                -- retaining for the rest the original string `str`
                -- Time complexity of this feature is O(logn)
        require
                len_greater_than_str_length: len > str.count
        local
                pad: STRING
                pad_len: INTEGER
        do
                Result := str.twin
                create pad.make_filled (ch, 1)
                pad_len := len - str.count
                from
                until
                        pad_len = 0
                loop
                        if pad_len \\ 2 = 1 then
                                Result.prepend (pad)
                        end
                        pad_len := pad_len // 2
                        if pad_len > 0 then
                                pad.append (pad)
                        end
                end
        ensure
                correct_length: Result.count = len
                original_string_not_changed: str ~ old str.twin
                padding_correct:
                        across 1 |..| (len - str.count) as
                                i all Result [i.item] = ch
                        end
                original_correct:
                        across (len - str.count + 1) |..| len as
                                i all Result [i.item] = str [i.item - (len - str.count)]
                        end
        end
```

Or just use:
Result.prepend (**create** {STRING}.make_filled (ch, len - str.count))

Specification

```
left_pad (str: STRING; len: INTEGER; ch: CHARACTER): STRING
                -- return a string of length `len`
                -- that is padded to the left with `len - str.count` characters `ch`
                -- retaining for the rest the original string `str`
                -- Time complexity of this feature is O(logn)
        require
                len > str.count
        ensure
                correct_length: Result.count = len
                original_string_not_changed: str = old str
                padding_correct:
                   -- ∀i ∈ 1..(len - #str): Result[i] = ch
                original_correct:
                   -- ∀i ∈ (len - #str + 1)..len: Result[i] = str [i - (len - #str)]

        end
```

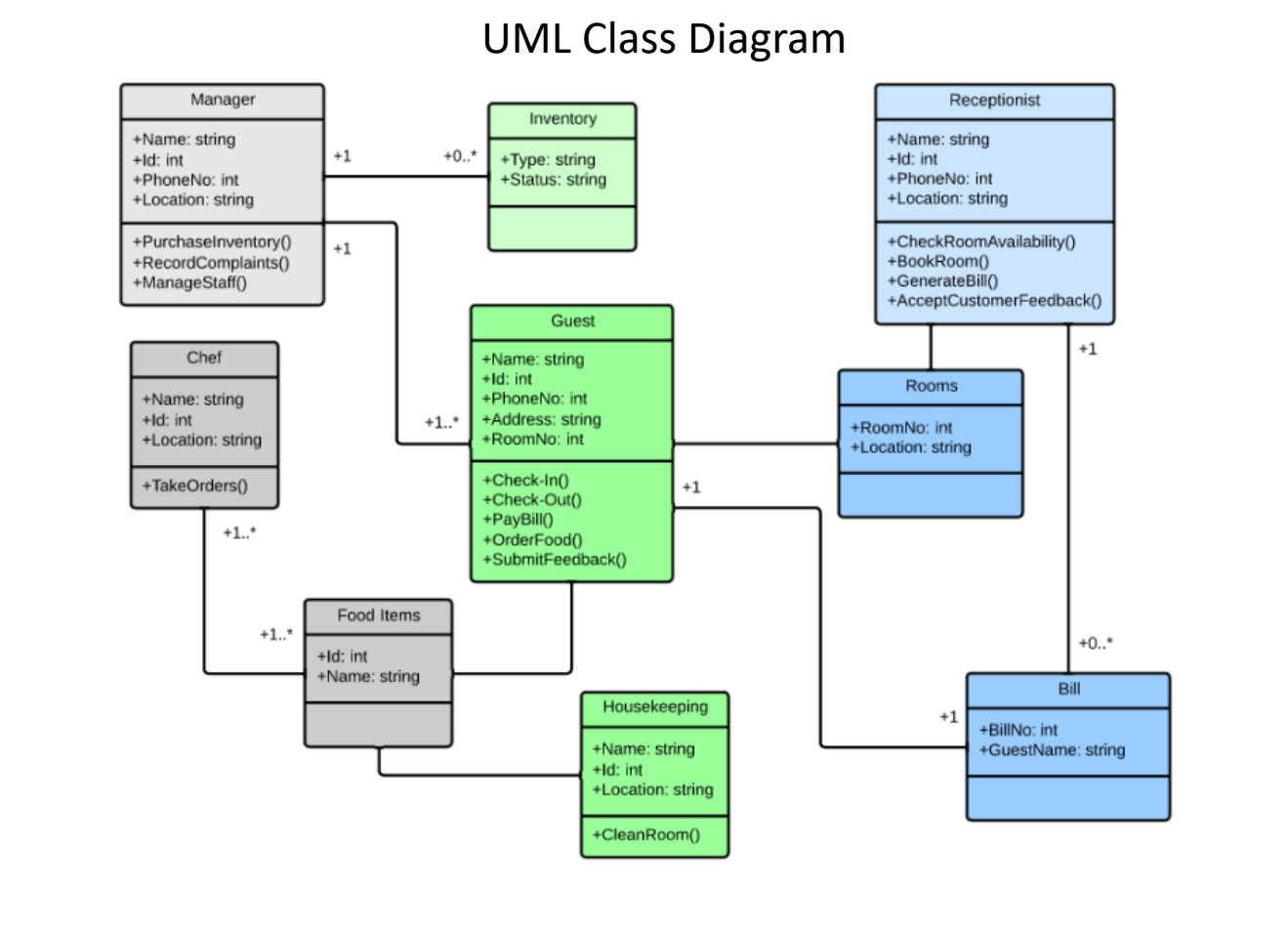
Precondition

Postcondition

Invariant

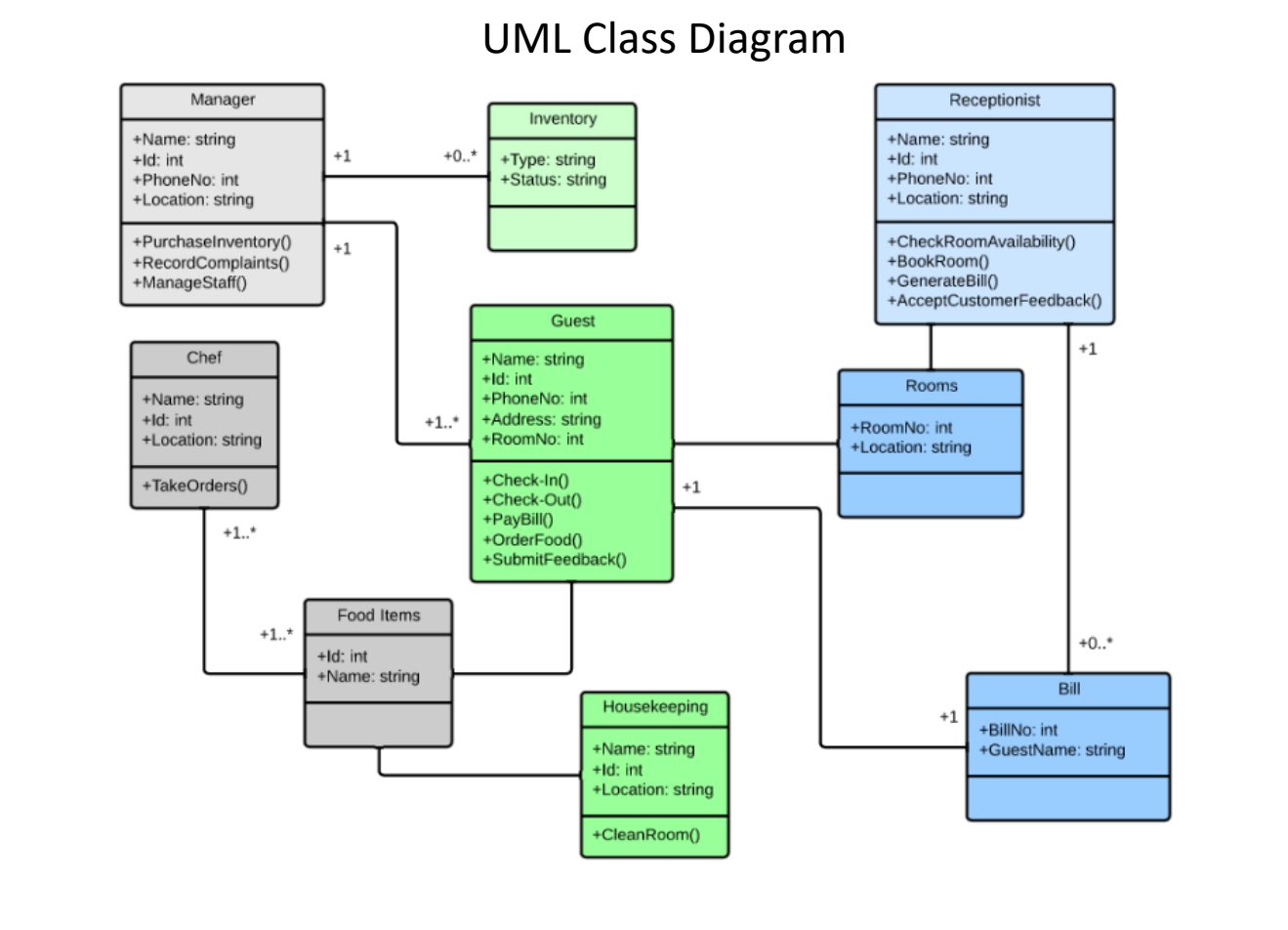
```
feature -- sqrt
    epsilon: REAL_64

    sqrt(x:REAL_64): REAL_64
            -- return square root of `x'
        require
            argument_non_negative: x >= 0
        local
            guess: REAL_64
        do
            from
                guess := 1
            until
                (guess*guess - x).abs <= epsilon
            loop
                guess := .5*(guess + x/guess)
            end
            Result := guess
        ensure
            result_accurate: (Result * Result - x).abs <= epsilon
            epsilon_unchanged: epsilon = old epsilon
        end

invariant
    correct_to_3_decimal_places:
        0 < epsilon and epsilon <= .001
```

What is Design?



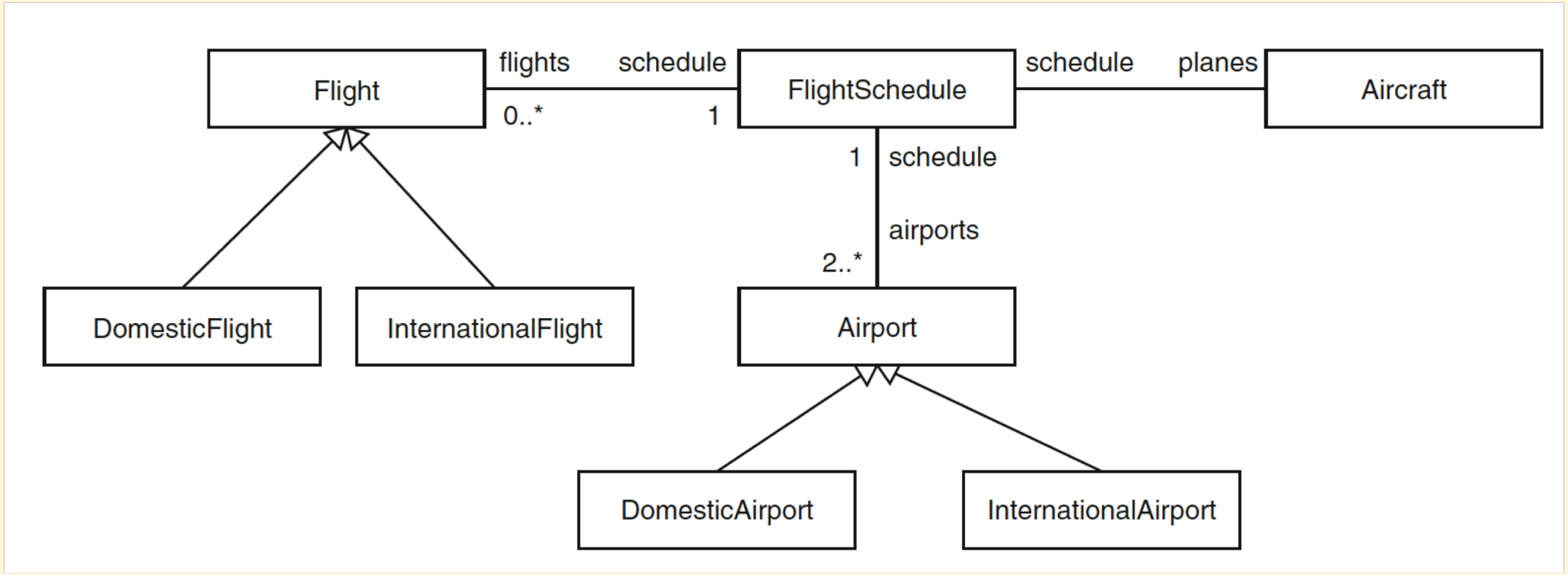
Architecture + Specifications

What is Design?

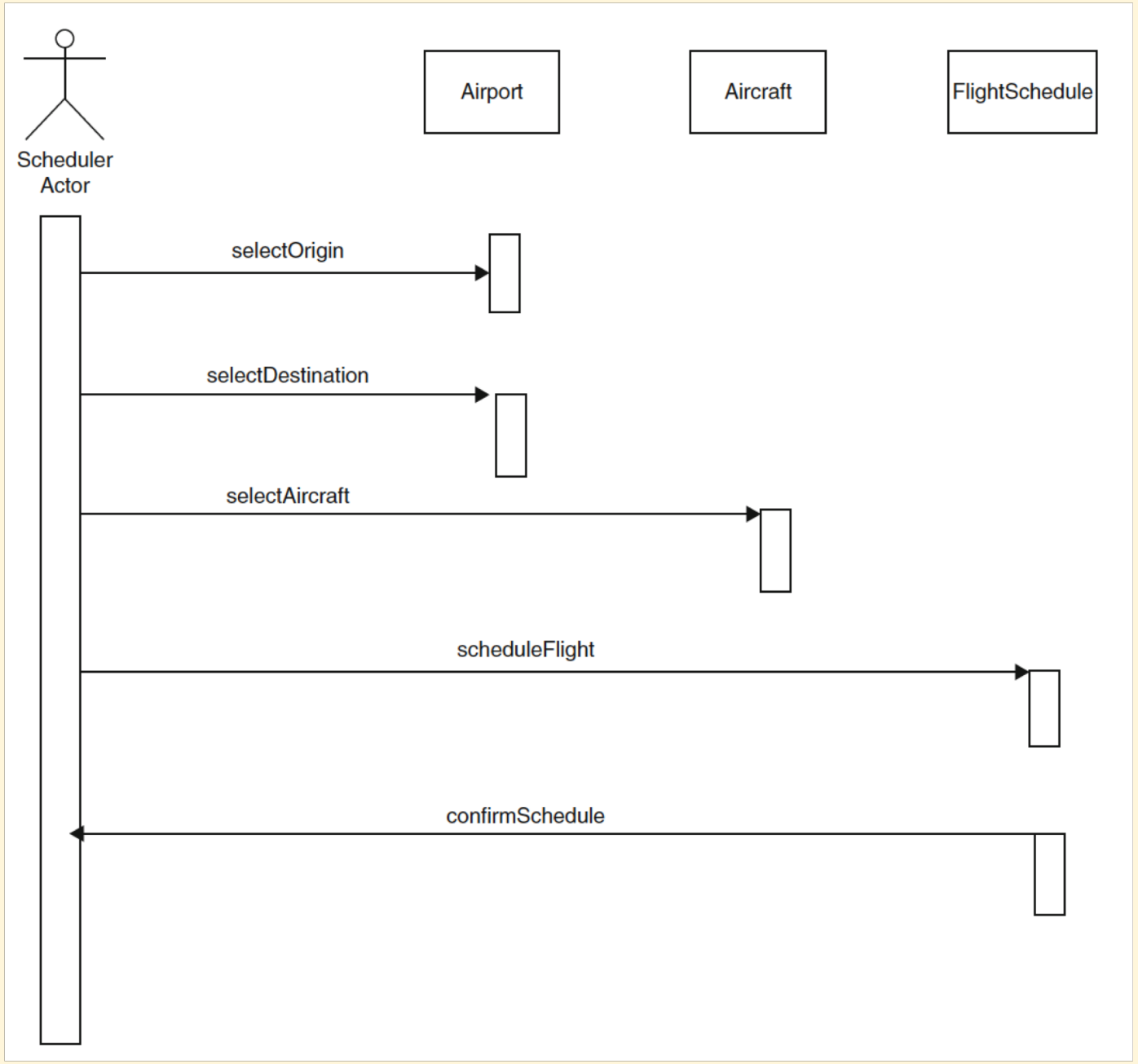


- Architecture (Blue Print)
  - Modules
  - Their features (attributes & methods)
  - How they are connected (inheritance or composition)
- Specification
  - Design By Contract
  - Preconditions, Postconditions & Class Invariants

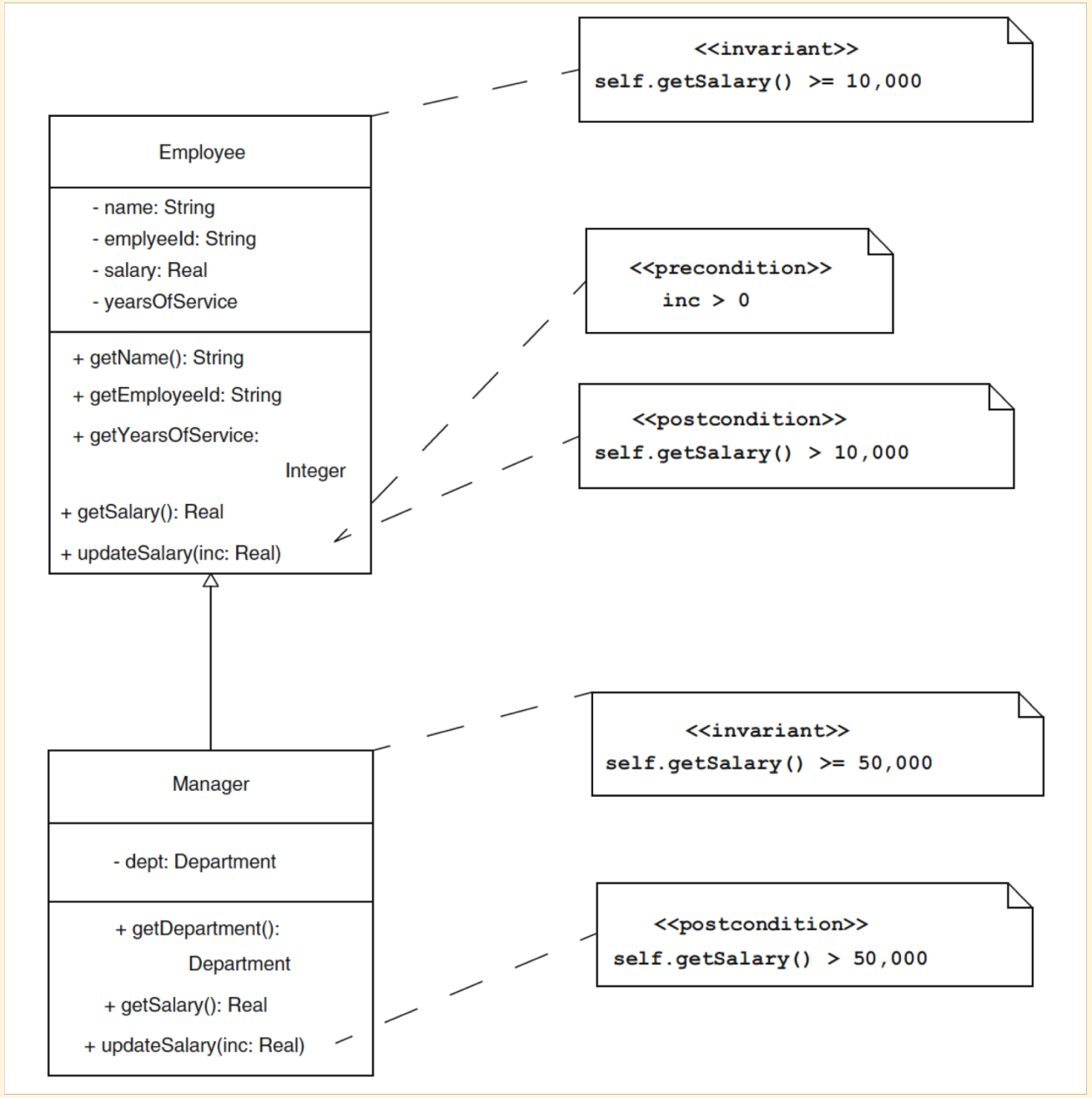
UML Class Diagram



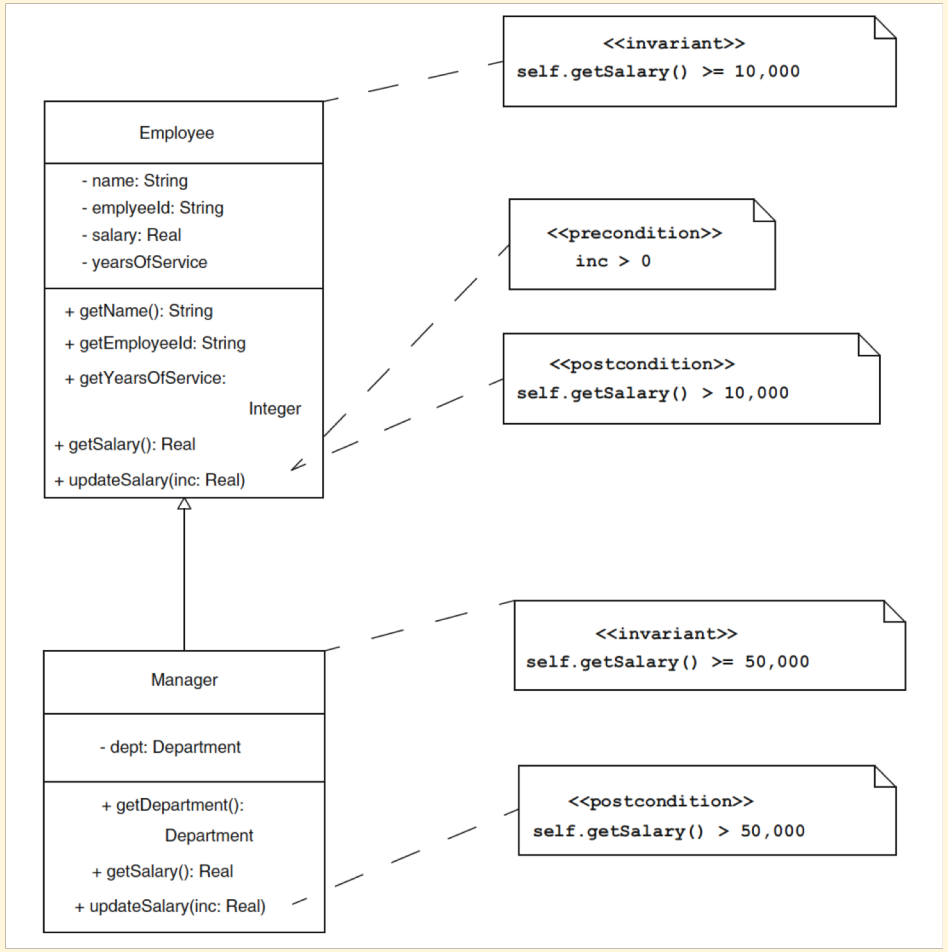
Sequence Diagram



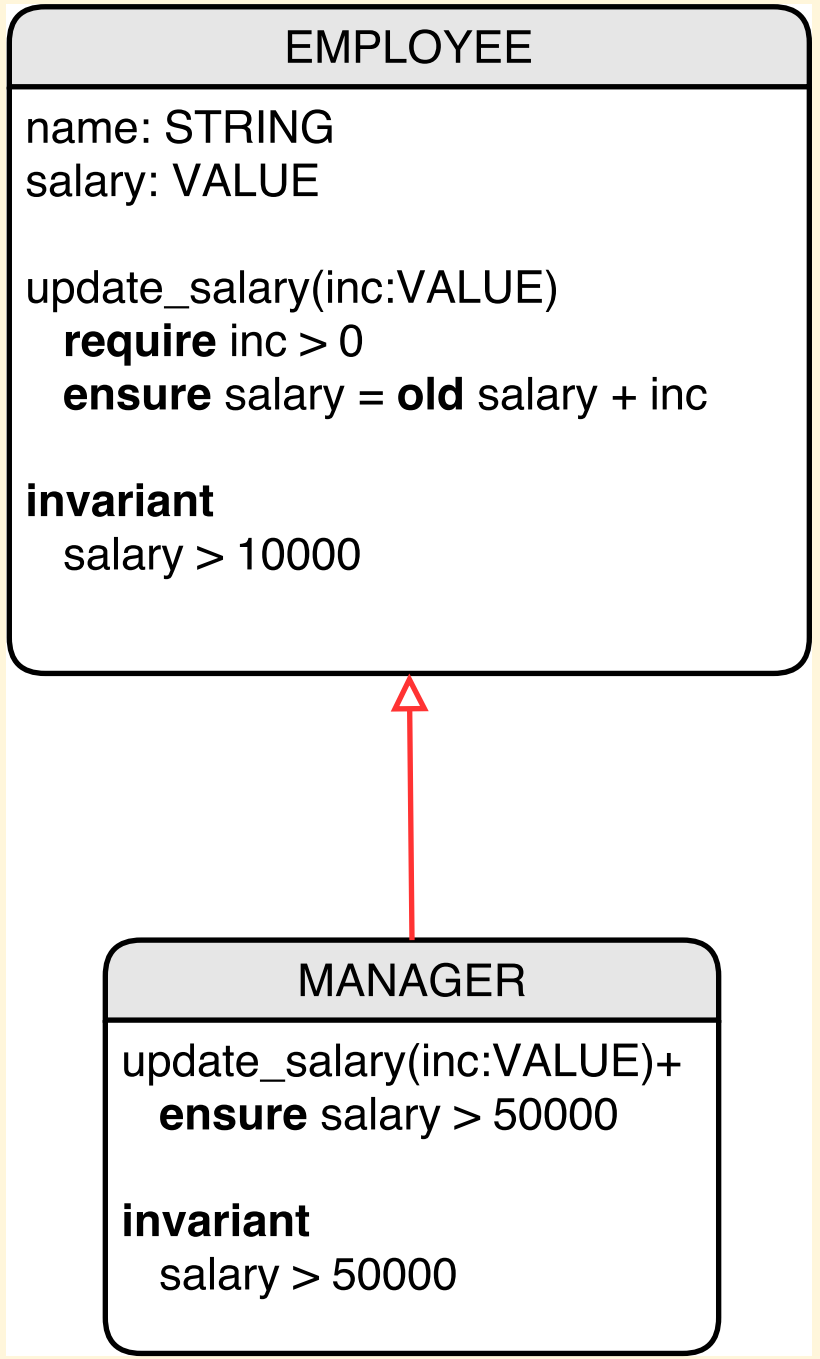
UML/OCL and inheritance



UML/OCL


<<invariant>>
self.getSalary() >= 10,000
Employee
- name: String
- emplyeeId: String
- salary: Real
- yearsOfService
+ getName(): String
+ getEmployeeId: String
+ getYearsOfService:
Integer
+ getSalary(): Real
+ updateSalary(inc: Real)
<<precondition>>
inc > 0
<<postcondition>>
self.getSalary() > 10,000
<<invariant>>
self.getSalary() >= 50,000
Manager
- dept: Department
+ getDepartment():
Department
+ getSalary(): Real
+ updateSalary(inc: Real)
<<postcondition>>
self.getSalary() > 50,000

BON


EMPLOYEE
name: STRING
salary: VALUE
update_salary(inc:VALUE)
require inc > 0
ensure salary = old salary + inc
invariant
salary > 10000
MANAGER
update_salary(inc:VALUE)+
ensure salary > 50000
invariant
salary > 50000

Eiffel DbC

EHEALTH

patients: SET [PATIENT]
medications: SET [MEDICATION]
prescriptions: REL [PATIENT, MEDICATION]
interactions: SET [INTERACTION] -- dangerous interactions

invariant
$\forall m1,m2 \in$ medications $\forall p \in$ PATIENT:
[m1,m2] $\in$ interactions $\Rightarrow \neg($ [p,m1] $\in$ prescriptions $\wedge$ [p,m2] $\in$ prescriptions$)$

ADD_PRESCRIPTION

add_prescription(p: PATIENT; m: MEDICATION)
require [p,m] $\notin$ prescriptions
$\forall x \in$ medications: [p,x] $\in$ prescriptions $\Rightarrow$ [x,m] $\notin$ interactions
ensure prescriptions = **old** prescriptions $\cup$ {[p,m]}

Precondition must preserve safety invariant

Eiffel the Method

- Multiple Inheritance (faithful to UML)
- OO and Functional (e.g. event driven programming)
- High-level concurrency model based on contracting
- Architecture
  - BON/UML diagram auto-generated and in sync with code
- Design by Contract
  - Specification
  - Self-documentation
  - Verification
  - Sub-contracting
  - Exceptions only when the contract is broken

OO Basics

Memory model: stack vs. heap
Encapsulation
Composition vs. Inheritance
Polymorphism
Static Typing, Dynamic Binding

Design

Abstraction
Divide and Conquer
Specifications vs, Implementations
Program to interface not to implementation
Modularity
Encapsulate what varies
Information Hiding
Uniform Access Principal
Open-Close Principle
Single Choice Principle
Justified Design Decisions

Applying Design Patterns

Strategy
Decorator
Observer
Visitor
CQRS (Command Query Responsibility Segregation)
...

anti-Patterns

Outside Student Comfort Zone

Thomas Ball (tball@microsoft.com) is a principal researcher and co-manager of the Research in Software Engineering
nd, WA.

a principal
ch in Software
ch, Redmond, WA.

DOI:10.1145/266

Viewp
Teach
Langu

Industry is ready
educated in the pr

Our recommendations are threefold, ... First, computer science majors, many of whom will be the designers and implementers of next-generation systems, should get a grounding in logic, ... "To designers of complex systems, the need for formal specs should be as obvious as the need for blueprints of a skyscraper." (Lesley Lamport)

The methods, tools, and materials for educating students about "formal specs" are ready for prime time. Mechanisms such as "design by contract," now available in mainstream programming languages, should be taught as part of introductory programming, as is done in the introductory programming language sequence at Carnegie Mellon University. … We are failing our computer science majors if we do not teach them about the value of formal specifications.

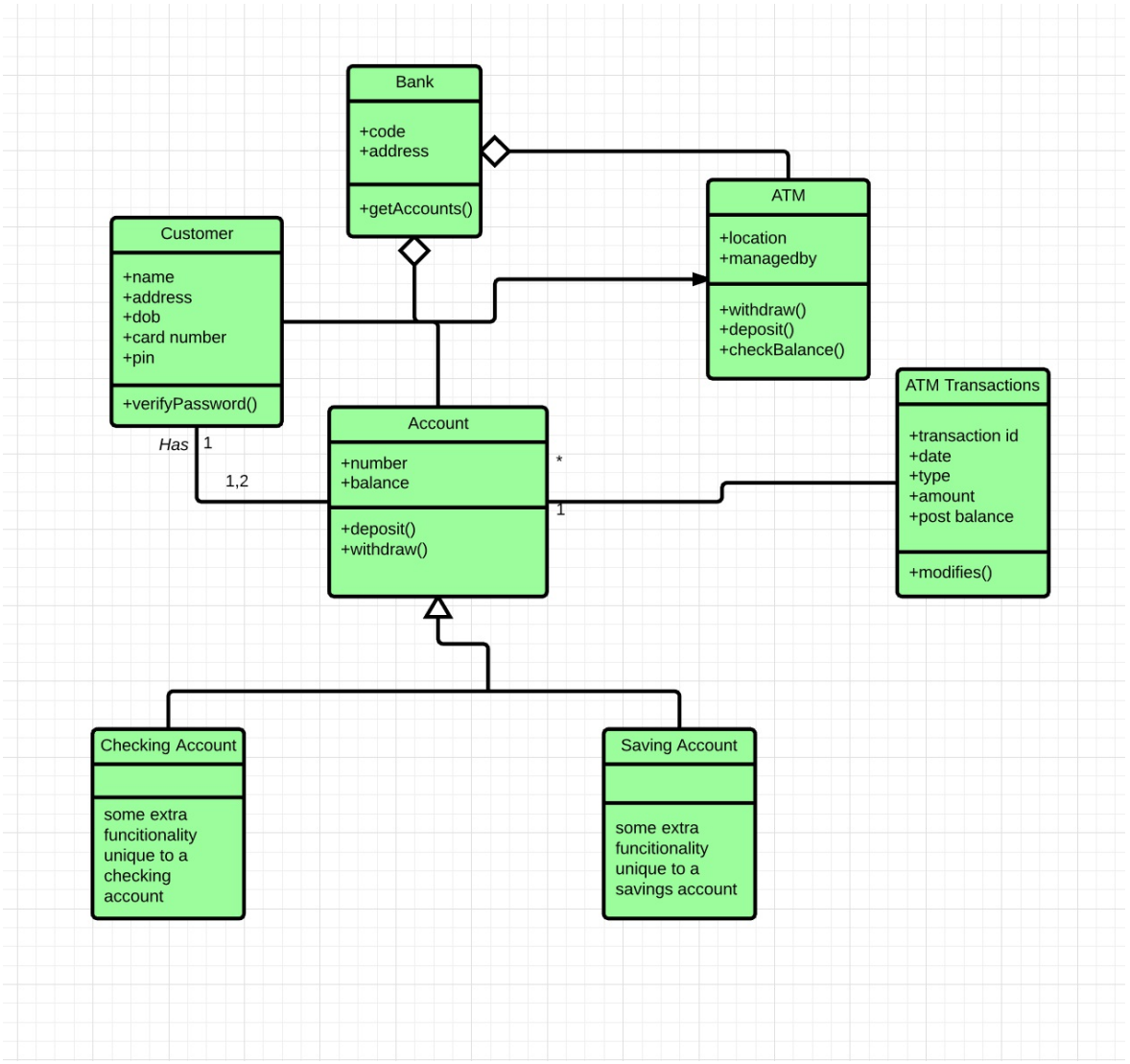
Two Tools

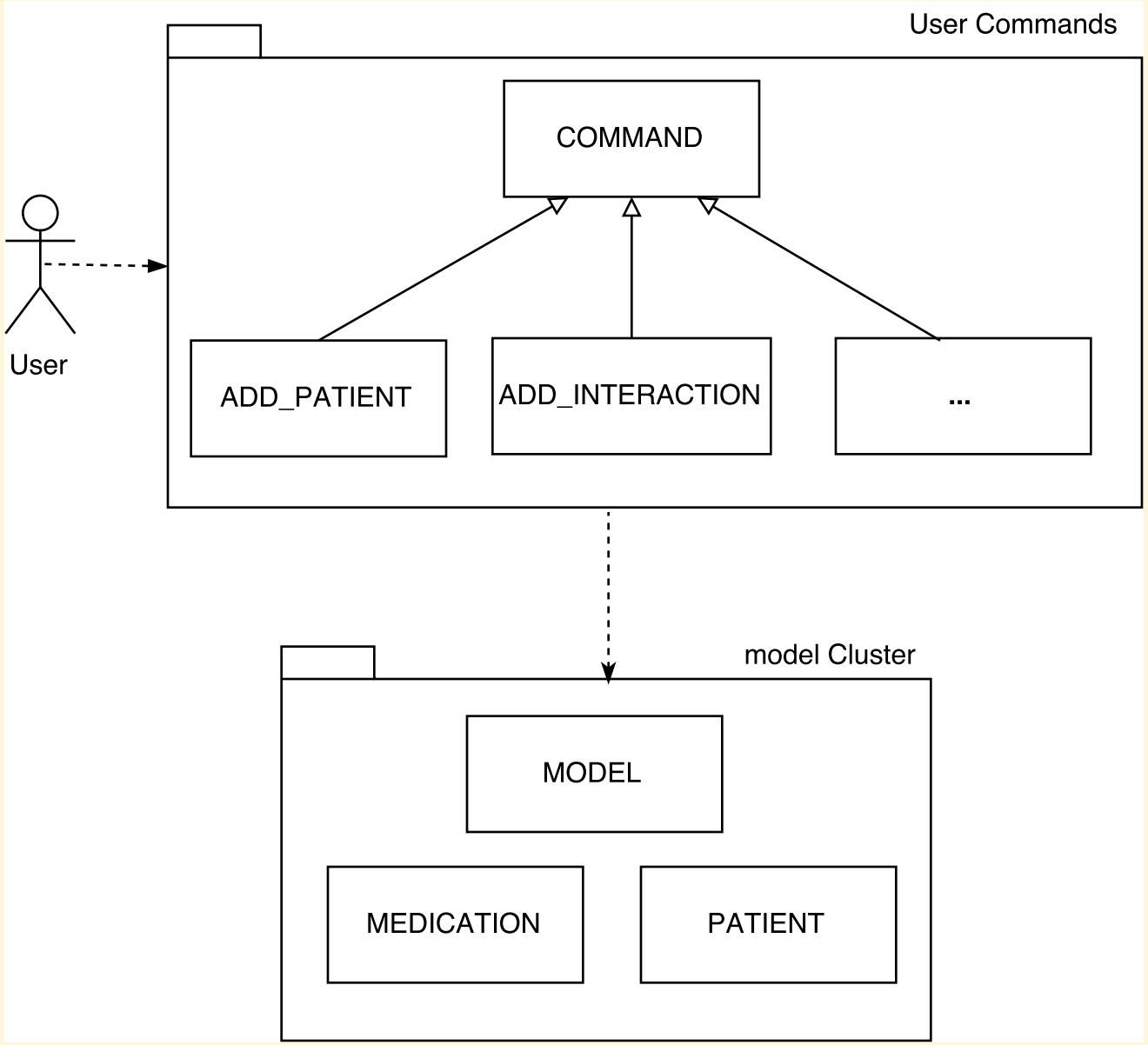
- ETF
  - Students design the business logic (architecture + specifications)
  - Test the design via end-to-end acceptance tests
- Mathmodels
  - SEQ[G]
  - SET[G]
  - FUN[G,H]
  - REL[G,H]
- Complete specifications
- Executable

Business Logic (the model)

Actor

User interface




User Commands
COMMAND
User
ADD_PATIENT
ADD_INTERACTION
...
model Cluster
MODEL
MEDICATION
PATIENT


MARCH 10, 2014
Ukraine's Future / Minimum-Wage Wars / Wes Anderson
TIME
CODE RED_
Inside the nightmare launch of HealthCare.gov
and the team that figured out how to fix it
BY STEVEN BRILL
time.com

Nanaimo doctors say electronic health record system unsafe, should be shut down

http://www.theprovince.com/

BY CINDY E. HARNETT, VICTORIA TIMES COLONIST MAY 27, 2016

Implementation of a $174-million Vancouver Island-wide electronic health record system in Nanaimo Regional General Hospital — set to expand to Victoria by late 2017 — is a huge failure, say senior physicians.

America's Voting Machines Are a Disaster in the Making

Forget Russian hackers or Donald Trump's fear-mongering about voter fraud. This election could be compromised for another reason entirely.

Software bug disrupts e-vote count in Belgian election

Amsterdam Correspondent, IDG News Service • May 26, 2014 6:25 AM PT

PCWorld NEWS

A bug in an e-voting application halted the release of European, federal and regional election results in Belgium, the country's interior ministry said Monday.

A bug in fMRI software could invalidate 15 years of brain research

http://www.sciencealert.com/a-bug-in-fmri-software-could-invalidate-decades-of-brain-research-scientists-discover

BEC CREW 6 JUL 2016

There could be a very serious problem with the past 15 years of research into human brain activity, with a new study suggesting that a bug in fMRI software could invalidate the results of some 40,000 papers.