Thinking Above the Code

Leslie Lamport
Microsoft Research

Why Think?

Why Think?

It helps us do most things:

Why Think?

It helps us do most things:

Hunting a sabre-toothed tiger.

Why Think?

It helps us do most things:

Hunting a sabre-toothed tiger.

Building a house.

Why Think?

It helps us do most things:

- Hunting a sabre-toothed tiger.
- Building a house.
- Writing a program.

When to Think

When to Think

Hunting a sabre-toothed tiger.

When to Think

Hunting a sabre-toothed tiger.

Before leaving the cave.

When to Think

Hunting a sabre-toothed tiger.

Before leaving the cave.

Building a house.

When to Think

Hunting a sabre-toothed tiger.

- Before leaving the cave.

Building a house.

- Before beginning construction.

When to Think

Hunting a sabre-toothed tiger.

Before leaving the cave.

Building a house.

Before beginning construction.

Writing a program.

When to Think

Hunting a sabre-toothed tiger.

Before leaving the cave.

Building a house.

Before beginning construction.

Writing a program.

Before writing any code.

How to Think

How to Think

"Writing is nature's way of letting you know how sloppy your thinking is."

Guindon

How to Think

"Writing is nature's way of letting you know how sloppy your thinking is."

Guindon

To think, you have to write.

How to Think

"Writing is nature's way of letting you know how sloppy your thinking is."

Guindon

To think, you have to write.

If you're thinking without writing, you only think you're thinking.

What to Write

What to Write

Hunting a sabre-toothed tiger.

What to Write

Hunting a sabre-toothed tiger.

Writing not invented, dangerous activity.

What to Write

Hunting a sabre-toothed tiger.

Writing not invented, dangerous activity.

Building a house.

What to Write

Hunting a sabre-toothed tiger.

Writing not invented, dangerous activity.

Building a house.

Draw blueprints.

What to Write

Hunting a sabre-toothed tiger.

  Writing not invented, dangerous activity.

Building a house.

  Draw blueprints.

Writing a program.

What to Write

Hunting a sabre-toothed tiger.

- Writing not invented, dangerous activity.

Building a house.

- Draw blueprints.

Writing a program.

- Write a blueprint

What to Write

Hunting a sabre-toothed tiger.

Writing not invented, dangerous activity.

Building a house.

Draw blueprints.

Writing a program.

Write a ~~blueprint~~ specification.

Specifications

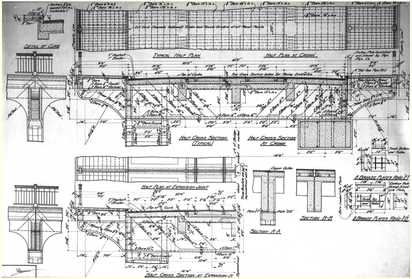
Specifications

Don’t Panic!

Specifications

Don’t Panic!

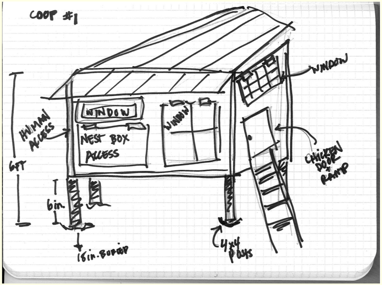
This is a blueprint:



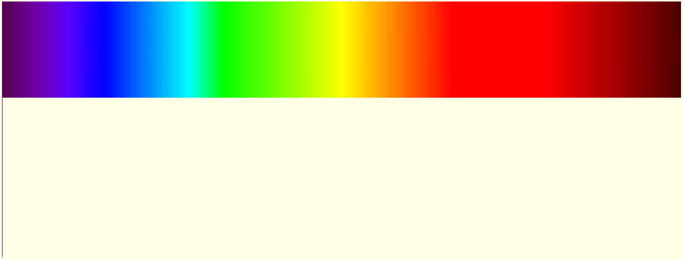
Specifications

Don’t Panic!

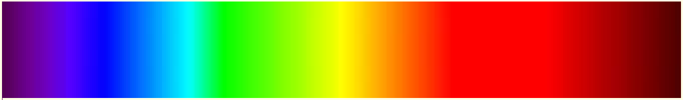
This is also a blueprint:



A spectrum of blueprints

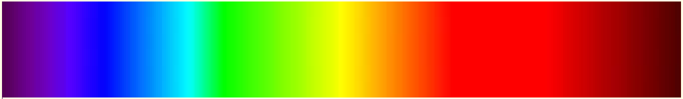


A spectrum of blueprints



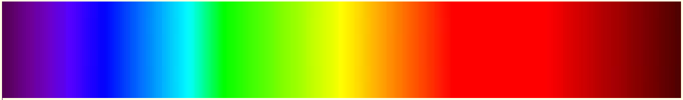
A spectrum of blueprints

Very Detailed



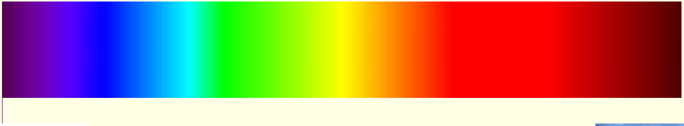
A spectrum of blueprints

Very Detailed



A spectrum of blueprints

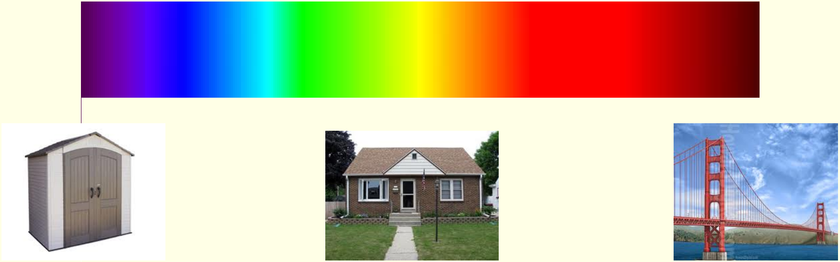
Rough Sketch Very Detailed



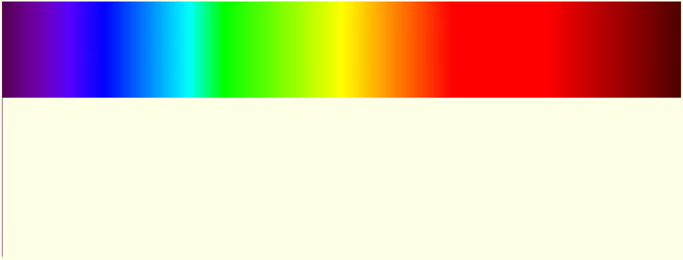
A spectrum of blueprints

A spectrum of blueprints

Rough Sketch Ordinary Very Detailed

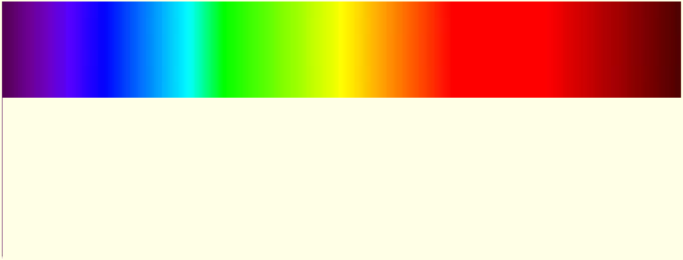


A spectrum of specifications

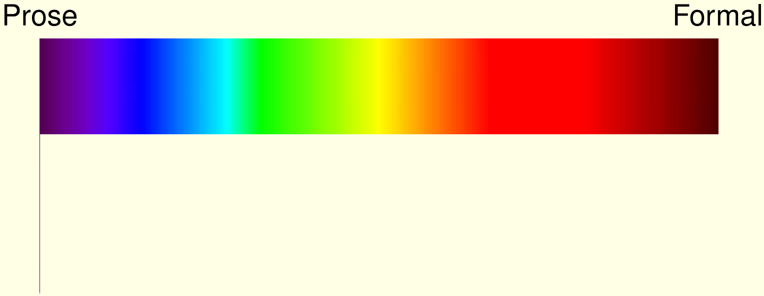


A spectrum of specifications

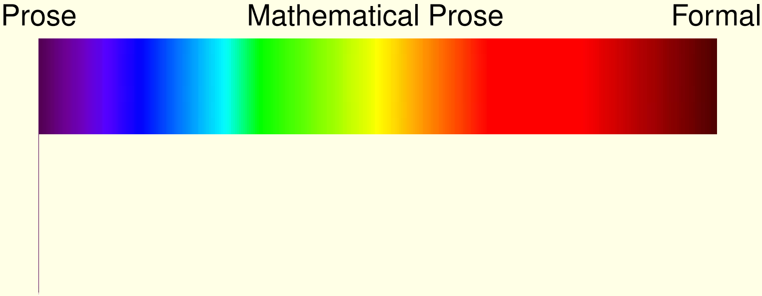
Formal



A spectrum of specifications

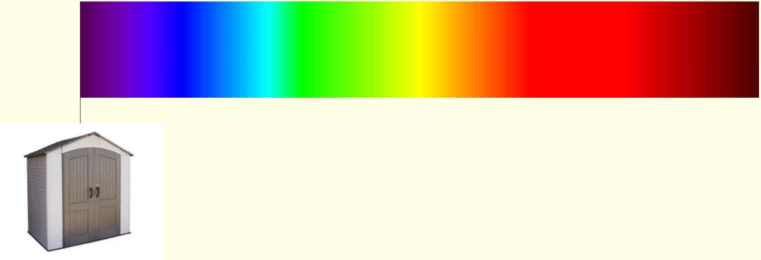


A spectrum of specifications



A spectrum of specifications

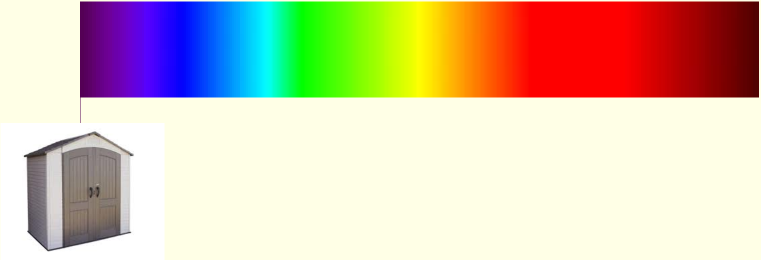
Prose



Most code is really simple.

A spectrum of specifications

Prose

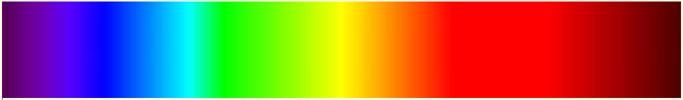


Most code is really simple.

It can be specified with a couple of lines of prose.

A spectrum of specifications

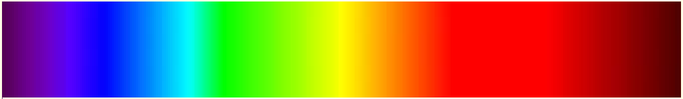
Mathematical Prose



Some code is subtle.

A spectrum of specifications

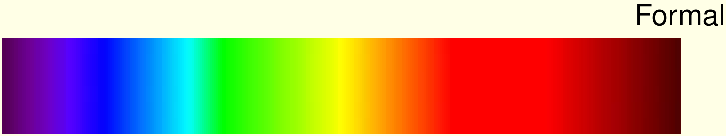
Mathematical Prose



Some code is subtle.

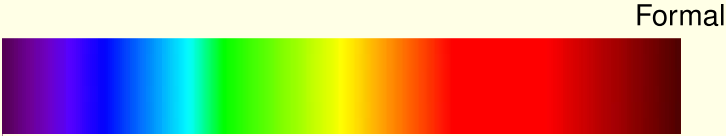
It requires more thought.

A spectrum of specifications



Some code is complex or very subtle or critical.

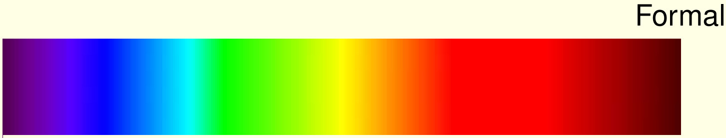
A spectrum of specifications



Some code is complex or very subtle or critical.

Especially in concurrent/distributed systems.

A spectrum of specifications



Some code is complex or very subtle or critical.

We should use tools to check it.

How to Write a Spec

How to Write a Spec

Writing requires thinking.

How to Think About Programs

How to Think About Programs

Like a Scientist

How to Think About Programs

Like a Scientist

A very successful way of thinking.

How to Think About Programs

Like a Scientist

A very successful way of thinking.

Science makes mathematical models of reality.

How to Think About Programs

Like a Scientist

A very successful way of thinking.

Science makes mathematical models of reality.

Astronomy:

How to Think About Programs

Like a Scientist

A very successful way of thinking.

Science makes mathematical models of reality.

Astronomy:

Reality: planets have mountains, oceans, tides, weather, ...

How to Think About Programs

Like a Scientist

A very successful way of thinking.

Science makes mathematical models of reality.

Astronomy:

Reality: planets have mountains, oceans, tides, weather, . . .

Model: planet a point mass with position & momentum.

Computer Science

Computer Science

Reality: Digital systems

Computer Science

Reality: Digital systems

a processor chip

Computer Science

Reality: Digital systems

a processor chip

a game console

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Models:

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Models: Turing machines

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Models: Turing machines

Partially ordered sets of events

Computer Science

Reality: Digital systems

a processor chip

a game console

a computer executing a program

⋮

Models: Turing machines

Partially ordered sets of events

⋮

The Two Most Useful Models

The Two Most Useful Models

Functions

The Two Most Useful Models

Functions

Sequences of States

Functions

Functions

Model a program as a function mapping input(s) to output(s).

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0,0\rangle,\ \langle 1,1\rangle,\ \langle 2,4\rangle,\ \langle 3,9\rangle,\ \ldots\}$$

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0, 0\rangle, \langle 1, 1\rangle, \langle 2, 4\rangle, \langle 3, 9\rangle, \ldots\}$$

$$square(2) = 4$$

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0, 0\rangle, \langle 1, 1\rangle, \langle 2, 4\rangle, \langle 3, 9\rangle, \ldots\}$$

Domain of $square$ is $\{0, 1, 2, 3, \ldots\}$ a.k.a. Nat

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0,0\rangle, \langle 1,1\rangle, \langle 2,4\rangle, \langle 3,9\rangle, \ldots\}$$

To define a function, specify:

Domain of $square$ = Nat

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0,0\rangle, \langle 1,1\rangle, \langle 2,4\rangle, \langle 3,9\rangle, \ldots\}$$

To define a function, specify:

Domain of $square$ = Nat

$square(x) = x^2$ for all x in its domain.

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0,0\rangle,\ \langle 1,1\rangle,\ \langle 2,4\rangle,\ \langle 3,9\rangle,\ \ldots\}$$

Functions in math $\neq$ functions in programming languages.

Functions

Model a program as a function mapping input(s) to output(s).

In math, a function is a set of ordered pairs.

Example: the $square$ function on natural numbers

$$\{\langle 0, 0\rangle, \langle 1, 1\rangle, \langle 2, 4\rangle, \langle 3, 9\rangle, \ldots\}$$

Functions in math $\neq$ functions in programming languages.

Math is much simpler.

Limitations of the Function Model

Limitations of the Function Model

Specifies what a program does, but not how.

Limitations of the Function Model

Specifies what a program does, but not how.

– Quicksort and bubble sort compute the same function.

Limitations of the Function Model

Specifies what a program does, but not how.

– Quicksort and bubble sort compute the same function.

Some programs don’t just map inputs to outputs.

Limitations of the Function Model

Specifies what a program does, but not how.

– Quicksort and bubble sort compute the same function.

Some programs don't just map inputs to outputs.

– Some programs run "forever".

Limitations of the Function Model

Specifies what a program does, but not how.

– Quicksort and bubble sort compute the same function.

Some programs don't just map inputs to outputs.

– Some programs run "forever".

– Operating systems

The Standard Behavioral Model

The Standard Behavioral Model

A program execution is represented by a **behavior.**

The Standard Behavioral Model

A program execution is represented by a **behavior**.

A behavior is a sequence of **states**.

The Standard Behavioral Model

A program execution is represented by a **behavior**.

A behavior is a sequence of **states**.

A state is an assignment of values to variables.

The Standard Behavioral Model

A program execution is represented by a **behavior**.

A behavior is a sequence of **states**.

A state is an assignment of values to variables.

A program is modeled by a set of behaviors

The Standard Behavioral Model

A program execution is represented by a **behavior**.

A behavior is a sequence of **states**.

A state is an assignment of values to variables.

A program is modeled by a set of behaviors:

the behaviors representing possible executions.

An Example: Euclid's Algorithm

An Example: Euclid’s Algorithm

An algorithm is an abstract program.

An Example: Euclid's Algorithm

Computes GCD of M and N by:

– Initialize x to M and y to N.

An Example: Euclid's Algorithm

Computes GCD of M and N by:

– Initialize x to M and y to N.

– Keep subtracting the smaller of x and y from the larger.

An Example: Euclid's Algorithm

Computes GCD of M and N by:

– Initialize x to M and y to N.

– Keep subtracting the smaller of x and y from the larger.

– Stop when $x = y$.

An Example: Euclid's Algorithm

Computes GCD of M and N by:

– Initialize x to M and y to N.

– Keep subtracting the smaller of x and y from the larger.

– Stop when $x = y$.

For $M = 12$ and $N = 18$, one behavior:

$$[x = 12,\ y = 18] \ \rightarrow\ [x = 12,\ y = 6] \ \rightarrow\ [x = 6,\ y = 6]$$

How to Describe a Set of Behaviors

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors $=$

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

Safety Property:

False iff violated at some point in behavior.

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

Safety Property:

False iff violated at some point in behavior.

Example: partial correctness.

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

Liveness Property:

Need to see complete behavior to know if it's false.

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

Liveness Property:

Need to see complete behavior to know if it's false.

Example: termination.

How to Describe a Set of Behaviors

Theorem

Any set $\mathcal{B}$ of behaviors =

all behaviors satisfying a safety property S

$\cap$ all behaviors satisfying a liveness property L

Specify a set of behaviors with

– a safety property

– a liveness property

In practice, specifying safety is more important.

In practice, specifying safety is more important.

That’s where errors are most likely to occur.

In practice, specifying safety is more important.

That's where errors are most likely to occur.

To save time, I'll ignore liveness.

How to Specify a Safety Property

How to Specify a Safety Property

With two things:

How to Specify a Safety Property

With two things:

– The set of possible initial states.

How to Specify a Safety Property

With two things:

– The set of possible initial states.

– A next-state relation,
describing all possible successor states of any state.

How to Specify a Safety Property

With two things:

– The set of possible initial states.

– A next-state relation,
describing all possible successor states of any state.

What language should we use?

How to Specify a Safety Property

With two things:

– The set of possible initial states.

– A next-state relation,
describing all possible successor states of any state.

What language should we use?

Let's act like scientists.

How to Specify a Safety Property

With two things:

– The set of possible initial states.

– A next-state relation,
describing all possible successor states of any state.

What language should we use?

Let's act like scientists.

Let's use math.

The Set of Initial States

The Set of Initial States

Described by a formula.

The Set of Initial States

Described by a formula.

For Euclid's Algorithm: $(x = M) \land (y = N)$

The Set of Initial States

Described by a formula.

For Euclid's Algorithm: $(x = M) \wedge (y = N)$

Only possible initial state: $[x = M, \ y = N]$

The Next-State Relation

Described by a formula.

The Next-State Relation

Described by a formula.

Unprimed variables for current state,

Primed variables for next state.

The Next-State Relation

Described by a formula.

Unprimed variables for current state,

Primed variables for next state.

For Euclid's Algorithm:

The Next-State Relation

Described by a formula.

Unprimed variables for current state,

Primed variables for next state.

For Euclid's Algorithm:

$$
\begin{array}{ll}
 & (\quad x > y \\
 & \quad\quad\quad\quad) \\
\vee & (\quad y > x \\
 & \quad\quad\quad\quad)
\end{array}
$$

The Next-State Relation

Described by a formula.

Unprimed variables for current state,

Primed variables for next state.

For Euclid's Algorithm:

$$
\begin{array}{ll}
 & (\quad x > y \\
 & \quad \wedge\ x' = x - y \\
 & \quad \wedge\ y' = y \quad) \\
\vee & (\quad y > x \\
 & \\
 & \qquad\qquad\qquad)
\end{array}
$$

The Next-State Relation

Described by a formula.

Unprimed variables for current state,

Primed variables for next state.

For Euclid's Algorithm:

$$
\begin{array}{ll}
 & (\quad x > y \\
 & \quad \wedge\ x' = x - y \\
 & \quad \wedge\ y' = y \ ) \\
\vee & (\quad y > x \\
 & \quad \wedge\ y' = y - x \\
 & \quad \wedge\ x' = x \ )
\end{array}
$$

For Euclid's Algorithm

For Euclid's Algorithm

Take $M = 12$, $N = 18$

Behavior:

For Euclid's Algorithm

$$Init: \quad (x = M) \wedge (y = N)$$

Behavior:

For Euclid's Algorithm

$Init$: $(x = 12) \wedge (y = 18)$

Behavior:

$[x = 12,\ y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & \quad (\quad x > y \\
& \quad\ \ \land\ x' = x - y \\
& \quad\ \ \land\ y' = y \quad) \\
& \lor\ (\quad y > x \\
& \quad\ \ \land\ y' = y - x \\
& \quad\ \ \land\ x' = x \quad)
\end{aligned}
$$

Behavior:

$[x = 12,\ y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{array}{lll}
Next: & & (\quad 12 > 18 \\
& & \;\; \land \;\; x' = 12 - 18 \\
& & \;\; \land \;\; y' = 18 \;\;) \\
& \lor & (\quad 18 > 12 \\
& & \;\; \land \;\; y' = 18 - 12 \\
& & \;\; \land \;\; x' = 12 \;\;)
\end{array}
$$

Behavior:

$[x = 12, \; y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & (\ \boxed{12 > 18} \quad \textbf{FALSE} \\
& \ \land\ x' = 12 - 18 \\
& \ \land\ y' = 18\ ) \\
\lor\ & (\ \boxed{18 > 12} \quad \textbf{TRUE} \\
& \ \land\ y' = 18 - 12 \\
& \ \land\ x' = 12\ )
\end{aligned}
$$

Behavior:

$[x = 12,\ y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \wedge (y = N)$

$$
\begin{aligned}
Next: \quad & (\quad 12 > 18 \\
& \;\wedge\; x' = 12 - 18 \\
& \;\wedge\; y' = 18 \;)
\end{aligned}
$$

FALSE

$$
\begin{aligned}
\vee \; & (\quad 18 > 12 \\
& \;\wedge\; y' = 18 - 12 \\
& \;\wedge\; x' = 12 \;)
\end{aligned}
$$

Behavior:

$[x = 12,\ y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & (\quad 12 > 18 \\
& \;\land\; x' = 12 - 18 \\
& \;\land\; y' = 18 \quad) \\
\lor \; & (\quad 18 > 12 \\
& \;\land\; y' = 18 - 12 \\
& \;\land\; x' = 12 \quad)
\end{aligned}
$$

FALSE

Behavior:

$[x = 12,\ y = 18]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & (\quad 12 > 18 \\
& \;\land\; x' = 12 - 18 \\
& \;\land\; y' = 18 \quad) \\
\lor\; & (\quad 18 > 12 \\
& \;\land\; y' = 18 - 12 \\
& \;\land\; x' = 12 \quad)
\end{aligned}
$$

FALSE

Behavior:

$[x = 12,\ y = 18] \rightarrow [x = 12,\ y = 6]$

For Euclid's Algorithm

$$Init: \quad (x = M) \land (y = N)$$

$$
\begin{aligned}
Next: \quad & (\quad x > y \\
& \;\; \land \; x' = x - y \\
& \;\; \land \; y' = y \;\;) \\
\lor \; & (\quad y > x \\
& \;\; \land \; y' = y - x \\
& \;\; \land \; x' = x \;\;)
\end{aligned}
$$

Behavior:

$[x = 12,\ y = 18] \ \rightarrow\ [x = 12,\ y = 6]$

For Euclid's Algorithm

$Init: \quad (x = M) \wedge (y = N)$

$$\begin{aligned} Next: \quad & (\quad 12 > 6 \\ & \quad \wedge\ x' = 12 - 6 \\ & \quad \wedge\ y' = 6 \quad) \\ \vee\ & (\quad 6 > 12 \\ & \quad \wedge\ y' = 6 - 12 \\ & \quad \wedge\ x' = 12 \quad) \end{aligned}$$

Behavior:

$$[x = 12,\ y = 18] \rightarrow [x = 12,\ y = 6]$$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & (\ \boxed{12 > 6}\ \ \textbf{TRUE} \\
& \ \land\ x' = 12 - 6 \\
& \ \land\ y' = 6\ ) \\
\lor\ & (\ \boxed{6 > 12}\ \ \textbf{FALSE} \\
& \ \land\ y' = 6 - 12 \\
& \ \land\ x' = 12\ )
\end{aligned}
$$

Behavior:

$[x = 12,\ y = 18] \rightarrow [x = 12,\ y = 6]$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$
\begin{aligned}
Next: \quad & (\quad 12 > 6 \\
& \land \; x' = 12 - 6 \\
& \land \; y' = 6 \;) \\
\lor \; & (\quad 6 > 12 \\
& \land \; y' = 6 - 12 \\
& \land \; x' = 12 \;)
\end{aligned}
$$

FALSE

Behavior:

$[x = 12,\; y = 18] \;\rightarrow\; [x = 12,\; y = 6]$

For Euclid's Algorithm

$$Init: \quad (x = M) \land (y = N)$$

$$
\begin{aligned}
Next: \quad & (\quad 12 > 6 \\
& \;\land\; \boxed{x' = 12 - 6} \\
& \;\land\; \boxed{y' = 6 \;)} \\
\lor\; & (\quad 6 > 12 \\
& \;\land\; y' = 6 - 12 \\
& \;\land\; x' = 12 \;)
\end{aligned}
$$

FALSE

Behavior:

$$[x = 12,\ y = 18] \;\rightarrow\; [x = 12,\ y = 6] \;\rightarrow\; [x = 6,\ y = 6]$$

For Euclid's Algorithm

$$Init: \quad (x = M) \land (y = N)$$

$$\begin{array}{lll} Next: & & (\quad x > y \\ & & \;\;\land\; x' = x - y \\ & & \;\;\land\; y' = y \quad) \\ & \lor & (\quad y > x \\ & & \;\;\land\; y' = y - x \\ & & \;\;\land\; x' = x \quad) \end{array}$$

Behavior:

$$[x = 12,\; y = 18] \;\rightarrow\; [x = 12,\; y = 6] \;\rightarrow\; [x = 6,\; y = 6]$$

For Euclid's Algorithm

$$Init: \quad (x = M) \wedge (y = N)$$

$$\begin{aligned} Next: \quad & (\ \boxed{6 > 6} \quad \textbf{FALSE} \\ & \ \wedge\ x' = 6 - 6 \\ & \ \wedge\ y' = 6\ ) \\ \vee\ & (\ \boxed{6 > 6} \quad \textbf{FALSE} \\ & \ \wedge\ y' = 6 - 6 \\ & \ \wedge\ x' = 6\ ) \end{aligned}$$

Behavior:

$$[x = 12,\ y = 18] \ \rightarrow\ [x = 12,\ y = 6] \ \rightarrow\ [x = 6,\ y = 6]$$

For Euclid's Algorithm

$Init: \quad (x = M) \land (y = N)$

$$\begin{aligned} Next: \quad & (\ 6 > 6 \\ & \ \land\ x' = 6 - 6 \\ & \ \land\ y' = 6\ ) \\ \lor\ & (\ 6 > 6 \\ & \ \land\ y' = 6 - 6 \\ & \ \land\ x' = 6\ ) \end{aligned}$$

FALSE

FALSE

Behavior:

$$[x = 12,\ y = 18] \ \rightarrow\ [x = 12,\ y = 6] \ \rightarrow\ [x = 6,\ y = 6]$$

For Euclid's Algorithm

$Init: \quad (x = M) \wedge (y = N)$

$$
\begin{array}{lll}
Next: & & (\quad 6 > 6 \\
& & \ \wedge \ x' = 6 - 6 \\
& & \ \wedge \ y' = 6 \quad) \\
& \vee & (\quad 6 > 6 \\
& & \ \wedge \ y' = 6 - 6 \\
& & \ \wedge \ x' = 6 \quad)
\end{array}
$$

FALSE

FALSE

NO NEXT STATE

Behavior:

$$[x = 12,\ y = 18] \ \rightarrow \ [x = 12,\ y = 6] \ \rightarrow \ [x = 6,\ y = 6]$$

Euclid’s Algorithm

.

Euclid's Algorithm

For any values of x and y, there are unique values of x' and y' that make $Next$ true.

.

Euclid's Algorithm

For any values of x and y, there are unique values of x' and y' that make $Next$ true.

Euclid's algorithm is deterministic.

Euclid's Algorithm

For any values of x and y, there are unique values of x' and y' that make $Next$ true.

Euclid's algorithm is deterministic.

To Model Nondeterminism

Euclid's Algorithm

For any values of x and y, there are unique values of x' and y' that make $Next$ true.

Euclid's algorithm is deterministic.

To Model Nondeterminism

Allow multiple next states for a current state.

Euclid's Algorithm

For any values of x and y, there are unique values of x' and y' that make $Next$ true.

Euclid's algorithm is deterministic.

To Model Nondeterminism

Allow multiple next states for a current state.

Multiple assignments of values to primed variables that make $Next$ true for a single assignment of values to unprimed variables.

What About Formal Specs?

What About Formal Specs?

Need them only to apply tools.

What About Formal Specs?

Need them only to apply tools.

Require a formal language.

The Language: TLA$^+$

The Language: TLA$^+$

This

$$Init: \quad (x = M) \land (y = N)$$

$$\begin{aligned} Next: \quad & \ (\quad x > y \\ & \qquad \land\ x' = x - y \\ & \qquad \land\ y' = y \quad) \\ & \lor\ (\quad y > x \\ & \qquad \land\ y' = y - x \\ & \qquad \land\ x' = x \quad) \end{aligned}$$

The Language: TLA+

becomes this

$$Init \triangleq (x = M) \land (y = N)$$

$$\begin{aligned} Next \triangleq \quad & (\quad x > y \\ & \;\land\; x' = x - y \\ & \;\land\; y' = y \;) \\ \lor\; & (\quad y > x \\ & \;\land\; y' = y - x \\ & \;\land\; x' = x \;) \end{aligned}$$

The Language: TLA$^+$

plus declarations

CONSTANTS M, N

VARIABLES x, y

$$Init \triangleq (x = M) \land (y = N)$$

$$\begin{aligned} Next \triangleq \quad & (\quad x > y \\ & \quad \land\ x' = x - y \\ & \quad \land\ y' = y \quad) \\ \lor\ & (\quad y > x \\ & \quad \land\ y' = y - x \\ & \quad \land\ x' = x \quad) \end{aligned}$$

The Language: TLA$^+$

plus some boilerplate.

MODULE *Euclid*

EXTENDS $Integers$

CONSTANTS M, N

VARIABLES x, y

$$Init \triangleq (x = M) \land (y = N)$$

$$\begin{aligned} Next \triangleq \quad & (\quad x > y \\ & \;\land\; x' = x - y \\ & \;\land\; y' = y \quad) \\ \lor \; & (\quad y > x \\ & \;\land\; y' = y - x \\ & \;\land\; x' = x \quad) \end{aligned}$$

The Language: TLA$^+$

You type

```
------------------ MODULE Euclid --------------------
  EXTENDS Integers

  CONSTANTS M, N

  VARIABLES x, y

  Init == (x = M) /\ (y = N)

  Next ==     (   x > y
               /\ x' = x - y
               /\ y' = y   )
            \/ (   y > x
               /\ y' = y - x
               /\ x' = x )
======================================================
```

You can model check TLA^+ specs.

You can model check TLA$^+$ specs.

– Checks all executions of a small model.

You can model check TLA$^+$ specs.

– Checks all executions of a small model.

– Extremely effective and fairly easy.

You can model check TLA$^+$ specs.

You can write formal correctness proofs
and check them mechanically.

You can model check TLA$^+$ specs.

You can write formal correctness proofs
and check them mechanically.

– Hard work.

You can model check TLA^{+} specs.

You can write formal correctness proofs
and check them mechanically.

But math works only for toy examples.

You can model check TLA$^+$ specs.

You can write formal correctness proofs
and check them mechanically.

But math works only for toy examples.

To model real systems, you need a real language
with types, procedures, objects, etc.

You can model check TLA$^+$ specs.

You can write formal correctness proofs
and check them mechanically.

~~But math works only for toy examples.~~

~~To model real systems, you need a real language
with types, procedures, objects, etc.~~

Wrong

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems.

Chris Newcombe
Amazon Engineer

[W]e have used TLA^+ on 10 large complex real-world systems. In every case TLA^+ has added significant value,

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production,

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness. Executive management are now proactively encouraging teams to write TLA$^+$ specs for new features and other significant design changes.

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness. Executive management are now proactively encouraging teams to write TLA$^+$ specs for new features and other significant design changes. In annual planning, managers are now allocating engineering time to use TLA$^+$.

Chris Newcombe
Amazon Engineer

[W]e have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness. Executive management are now proactively encouraging teams to write TLA$^+$ specs for new features and other significant design changes. In annual planning, managers are now allocating engineering time to use TLA$^+$.

Chris Newcombe
Amazon Engineer
November, 2013

The XBox 360 Memory System

The XBox 360 Memory System

Writing a TLA$^+$ spec caught a bug that would not otherwise have been found.

The XBox 360 Memory System

Writing a TLA$^+$ spec caught a bug that would not otherwise have been found.

That bug would have caused every XBox 360 to crash after 4 hours of use.

You can learn about TLA$^+$ on the web.

You can learn about TLA$^+$ on the web.

Today, I’ll talk about informal specs, starting with an example.

TLATEX — the TLA$^+$ pretty-printer

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The naive output:

$$\begin{array}{l} Foo \;\Rightarrow\; \land\, a = b \\ \qquad\quad \land\, ccc = d \end{array}$$

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The naive output:

$$
\begin{array}{l}
Foo \;\Rightarrow\; \land\; a = b \\
\qquad\quad\;\; \land\; ccc = d
\end{array}
$$

The user probably wanted these aligned.

TLATEX — the TLA$^+$ pretty-printer

The input:

```
Foo => /\ a   = b
       /\ ccc = d
```

The right output:

$$\begin{array}{lll} Foo \;\Rightarrow & \land\; a & = b \\ & \land\; ccc & = d \end{array}$$

The user probably wanted these aligned.

TLATEX — the TLA$^+$ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

TLATEX — the TLA$^+$ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The naive output:

$$\wedge\ aaa + bb = c$$
$$\wedge\ iii = jj * k$$

TLATEX — the TLA$^+$ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The naive output:

$$\land\ aaa + bb = c$$
$$\land\ iii = jj * k$$

The user probably didn't wanted these aligned.

TLATEX — the TLA$^+$ pretty-printer

The input:

```
/\ aaa + bb = c
/\ iii = jj * k
```

The right output:

$$\begin{aligned}&\land\ aaa + bb = c\\&\land\ iii = jj * k\end{aligned}$$

The user probably didn't wanted these aligned.

There is no precise definition of correct alignment.

There is no precise definition of correct alignment.

We can’t specify mathematically what the user wants.

There is no precise definition of correct alignment.

We can’t specify mathematically what the user wants.

If we can’t specify correctness, specification is useless.

There is no precise definition of correct alignment.

We can’t specify mathematically what the user wants.

~~If we can’t specify correctness, specification is useless.~~

Wrong.

There is no precise definition of correct alignment.

We can't specify mathematically what the user wants.

~~If we can't specify correctness, specification is useless.~~

Wrong.

Not knowing what a program should do means
we have to think even harder.

There is no precise definition of correct alignment.

We can't specify mathematically what the user wants.

~~If we can't specify correctness, specification is useless.~~

Wrong.

Not knowing what a program should do means
we have to think even harder.

Which means that a spec is even more important.

My Spec

My Spec

6 rules plus definitions (in comments).

My Spec

6 rules plus definitions (in comments).

Example:

A left-comment token is LeftComment aligned with its covering token.

My Spec

6 rules plus definitions (in comments).

Example:

A left-comment token is LeftComment aligned with its covering token.

Defined term.

Why did I write this spec?

Why did I write this spec?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

Why did I write this spec?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules

Why did I write this spec?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

Why did I write this spec?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

The few bugs in implementing the rules were easy to catch.

Why did I write this spec?

It was a lot easier to understand and debug 6 rules than 850 lines of code.

I did a lot of debugging of the rules, aided by debugging code to report what rules were being used.

The few bugs in implementing the rules were easy to catch.

Had I just written code, it would have taken me much longer and not produced formatting as good.

Why not a formal spec?

Why not a formal spec?

Getting it right not that important.

Why not a formal spec?

Getting it right not that important.

It didn't have to work on all corner cases.

Why not a formal spec?

Getting it right not that important.

It didn't have to work on all corner cases.

There were no tools that could help me.

What is Typical About This Spec

What is Typical About This Spec

The spec is at a higher-level than the code.

What is Typical About This Spec

The spec is at a higher-level than the code.

It could have been implemented in any language.

What is Typical About This Spec

The spec is at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the spec.

What is Typical About This Spec

The spec is at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the spec.

No method or tool for writing better code would have made the spec unnecessary.

What is Typical About This Spec

The spec is at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the spec.

No method or tool for writing better code would have made the spec unnecessary.

It says nothing about how to write code.

What is Typical About This Spec

The spec is at a higher-level than the code.

It could have been implemented in any language.

No method or tool for writing better code would have helped to write the spec.

No method or tool for writing better code would have made the spec unnecessary.

It says nothing about how to write code.

You write a spec to help you think about the problem before you think about the code.

What is Not Typical About This Spec

What is Not Typical About This Spec

It’s quite subtle.

What is Not Typical About This Spec

It's quite subtle.

95% of the code most people write requires less thought; specs that are shorter and simpler suffice.

What is Not Typical About This Spec

It's quite subtle.

95% of the code most people write requires less thought; specs that are shorter and simpler suffice.

It's a set of rules.

What is Not Typical About This Spec

It's quite subtle.

95% of the code most people write requires less thought; specs that are shorter and simpler suffice.

It's a set of rules.

A set of rules/requirements/axioms is usually a bad spec.

What is Not Typical About This Spec

It's quite subtle.

95% of the code most people write requires less thought; specs that are shorter and simpler suffice.

It's a set of rules.

A set of rules/requirements/axioms is usually a bad spec.

It's hard to understand the consequences of a set of rules.

Specifying How to Compute a Function

Specifying How to Compute a Function

Specifying what the pretty-printer should do was hard.

Specifying How to Compute a Function

Specifying what the pretty-printer should do was hard.

Implementing the spec was easy.

Specifying How to Compute a Function

Specifying what the pretty-printer should do was hard.

Implementing the spec was easy.

Specifying what a sorting program should do is easy.

Specifying How to Compute a Function

Specifying what the pretty-printer should do was hard.

Implementing the spec was easy.

Specifying what a sorting program should do is easy.

Figuring out how to implement it efficiently is hard
(if no one has shown you).

Specifying How to Compute a Function

Specifying what the pretty-printer should do was hard.

Implementing the spec was easy.

Specifying what a sorting program should do is easy.

Figuring out how to implement it efficiently is hard
(if no one has shown you).

It requires thinking, which means writing a specification.

An example: Quicksort

An example: Quicksort

A divide-and-conquer algorithm for sorting an array $A[0], \ldots, A[N-1]$.

An example: Quicksort

A divide-and-conquer algorithm for sorting an array $A[0], \ldots, A[N-1]$.

For simplicity, assume the $A[i]$ are numbers.

It uses a procedure $Partition(lo, hi)$.

It uses a procedure $Partition(lo, hi)$.

It chooses $pivot$ in $lo \ldots (hi - 1)$, permutes $A[lo], \ldots, A[hi]$ to make $A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$, and returns $pivot$.

It uses a procedure $Partition(lo,\, hi)$.

It chooses $pivot$ in $lo \ldots (hi - 1)$, permutes $A[lo], \ldots, A[hi]$ to make $A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$, and returns $pivot$.

For this example, we don't care how this procedure is implemented.

Let's specify Quicksort in pseudo-code.

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
$QS(lo, p)$;
$QS(p + 1, hi)$; } }

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
$QS(lo, p)$;
$QS(p + 1, hi)$; } }

main { $QS(0, N - 1)$; }

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
$QS(lo, p)$;
$QS(p + 1, hi)$; } }

main { $QS(0, N - 1)$; }

procedure $Partition(lo, hi)$ {
  Pick $pivot$ in $lo \ldots (hi - 1)$;
  Permute $A[lo], \ldots, A[hi]$ to make
    $A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;
  return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
  $QS(lo, p)$;
  $QS(p + 1, hi)$; } }

main { $QS(0, N - 1)$; }

Informal: no formal syntax, no declarations, ...

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
$QS(lo, p)$;
$QS(p + 1, hi)$; } }

main { $QS(0, N - 1)$; }

Informal: no formal syntax, no declarations, ...

Easy to understand.

procedure $Partition(lo, hi)$ {

Pick $pivot$ in $lo \ldots (hi - 1)$;

Permute $A[lo], \ldots, A[hi]$ to make
$A[lo], \ldots, A[pivot] \leq A[pivot + 1], \ldots, A[hi]$;

return $pivot$; }

procedure $QS(lo, hi)$ { **if** $(lo < hi)$ { $p := Partition(lo, hi)$;
$QS(lo, p)$;
$QS(p + 1, hi)$; } }

main { $QS(0, N - 1)$; }

But is it really Quicksort?

It’s the way Quicksort is almost always described.

It’s the way Quicksort is almost always described.

But recursion is not a fundamental part of Quicksort.

It’s the way Quicksort is almost always described.

But recursion is not a fundamental part of Quicksort.

It’s just one way of implementing divide-and-conquer.

It's the way Quicksort is almost always described.

But recursion is not a fundamental part of Quicksort.

It's just one way of implementing divide-and-conquer.

It's probably not the best way for parallel execution.

Problem: Write a non-recursive version of Quicksort.

Problem: Write a non-recursive version of Quicksort.

Almost no one can do it in 10 minutes.

Problem: Write a non-recursive version of Quicksort.

Almost no one can do it in 10 minutes.

They try to “compile” the recursive version.

Solution:

Solution:

Maintain a set U of index ranges on which $Partition$ needs to be called.

Solution:

Maintain a set U of index ranges on which $Partition$ needs to be called.

Initially, U equals $\{\langle 0, N-1 \rangle\}$

Solution:

Maintain a set U of index ranges on which $Partition$ needs to be called.

Initially, U equals $\{\langle 0, N-1 \rangle\}$

We could write it in pseudo-code,
but it's better to simply write $Init$ and $Next$ directly.

$$
\begin{aligned}
Init\text{:}\quad & A = \text{any array of numbers of length } N \\
\wedge\ & U = \{\langle 0, N-1 \rangle\}
\end{aligned}
$$

$$\begin{aligned} Init: \quad & A = \text{any array of numbers of length } N \\ & \land\ U = \{\langle 0, N-1 \rangle\} \end{aligned}$$

Before writing $Next$, let's make a definition:

$$Init: \quad A = \text{any array of numbers of length } N$$
$$\land \; U = \{\langle 0, N-1 \rangle\}$$

Before writing $Next$, let's make a definition:

$Partitions(B, pivot, lo, hi) \triangleq$
the set of arrays obtained from B by permuting
$B[lo], \ldots, B[hi]$ such that ...

$Init$: A = any array of numbers of length N
$\wedge$ $U = \{\langle 0, N-1 \rangle\}$

Before writing $Next$, let's make a definition:

$Partitions(B, pivot, lo, hi) \triangleq$
the set of arrays obtained from B by permuting $B[lo], \ldots, B[hi]$ such that ...

$Next$:

A relation between old values of A, U and new values A', U'.

$Next$:

$Next$:

$$U \neq \{\}$$ Stop if $U = \{\}$

$\wedge$

$Next$:

$$U \neq \{\}$$

$\land$ Pick any $\langle b, t \rangle$ in U :

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN

ELSE

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

ELSE

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

$A' =$ Any element of $Partitions(A, p, b, t)$

ELSE

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

$A' =$ Any element of $Partitions(A, p, b, t)$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed and $\langle b, p \rangle$ and $\langle p+1, t \rangle$ added

ELSE

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\land \ \text{Pick any } \langle b, t \rangle \text{ in } U: \\
\qquad \text{IF } b \neq t \\
\qquad\quad \text{THEN Pick any } p \text{ in } b \,..\, (t-1): \\
\qquad\qquad\quad A' = \text{Any element of } Partitions(A, p, b, t) \\
\qquad\qquad \land \ U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\qquad\qquad\qquad\quad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
\qquad\quad \text{ELSE} \quad A' = A
\end{array}
$$

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

$A' =$ Any element of $Partitions(A, p, b, t)$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed and $\langle b, p \rangle$ and $\langle p+1, t \rangle$ added

ELSE $A' = A$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed

Why can (almost) no one find this version of Quicksort?

Why can (almost) no one find this version of Quicksort?

Their minds are stuck in code.

Why can (almost) no one find this version of Quicksort?

Their minds are stuck in code.

They can't think at a higher level.

$Next$:

$U \neq \{\}$

$\wedge$ Pick any $\langle b, t \rangle$ in U :

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

$A' =$ Any element of $Partitions(A, p, b, t)$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed and $\langle b, p \rangle$ and $\langle p+1, t \rangle$ added

ELSE $A' = A$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed

Easy to write this as a formula.

$Next$:

$$
\begin{array}{l}
U \neq \{\} \\
\land \text{ Pick any } \langle b, t \rangle \text{ in } U: \\
\quad \text{IF } b \neq t \\
\quad \text{THEN Pick any } p \text{ in } b \,..\, (t-1): \\
\qquad\qquad A' = \text{Any element of } Partitions(A, p, b, t) \\
\qquad\quad \land\ U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\qquad\qquad\qquad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
\quad \text{ELSE} \qquad A' = A \\
\qquad\quad \land\ U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{array}
$$

Pick an arbitrary value

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\wedge \;\; \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } \; b \neq t \\
\qquad\quad \text{THEN } \; \text{Pick any } p \text{ in } b \,..\, (t-1) : \\
\qquad\qquad\qquad A' = \text{Any element of } Partitions(A, p, b, t) \\
\qquad\qquad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\qquad\qquad\qquad\qquad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
\qquad\quad \text{ELSE } \qquad A' = A \\
\qquad\qquad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{array}
$$

Pick an arbitrary value is existential quantification.

$Next$:

$U \neq \{\}$

$\wedge$ $\exists \langle b, t \rangle \in U$:

IF $b \neq t$

THEN Pick any p in $b \,..\, (t-1)$:

$A' =$ Any element of $Partitions(A, p, b, t)$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed and $\langle b, p \rangle$ and $\langle p+1, t \rangle$ added

ELSE $A' = A$

$\wedge$ $U' = U$ with $\langle b, t \rangle$ removed

Pick an arbitrary value is existential quantification.

$Next$:

$$
\begin{aligned}
& U \neq \{\} \\
\land\ & \exists \langle b, t \rangle \in U : \\
& \quad \text{IF } b \neq t \\
& \quad \quad \text{THEN } \exists p \in b \,..\, (t-1) : \\
& \quad \quad \quad \quad A' = \text{Any element of } Partitions(A, p, b, t) \\
& \quad \quad \quad \land\ U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
& \quad \quad \quad \quad \quad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
& \quad \quad \text{ELSE} \quad A' = A \\
& \quad \quad \quad \land\ U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{aligned}
$$

Pick an arbitrary value is existential quantification.

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\wedge \;\; \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } \; b \neq t \\
\qquad\quad \text{THEN } \; \exists\, p \in b \,..\, (t-1) : \\
\qquad\qquad\qquad A' = \text{Any element of } Partitions(A, p, b, t) \\
\qquad\qquad\quad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\qquad\qquad\qquad\qquad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
\qquad\quad \text{ELSE } \qquad A' = A \\
\qquad\qquad\quad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{array}
$$

Or sometimes

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\land \ \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } b \neq t \\
\qquad\quad \text{THEN } \exists p \in b \,..\, (t-1) : \\
\qquad\qquad\qquad A' \in Partitions(A, p, b, t) \\
\qquad\qquad\quad \land \ U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\qquad\qquad\qquad\qquad \langle b, p \rangle \text{ and } \langle p+1, t \rangle \text{ added} \\
\qquad\quad \text{ELSE } \qquad A' = A \\
\qquad\qquad\quad \land \ U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{array}
$$

Or sometimes even simpler.

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\wedge \;\; \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } \; b \neq t \\
\qquad\quad \text{THEN } \; \exists\, p \in b \,..\, (t-1) : \\
\qquad\qquad\qquad A' \in Partitions(A, p, b, t) \\
\qquad\qquad\quad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed and} \\
\\
\qquad\quad \text{ELSE } \qquad A' = A \\
\qquad\qquad\quad \wedge \;\; U' = U \text{ with } \langle b, t \rangle \text{ removed}
\end{array}
$$

And so on.

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\land \ \exists \langle b, t \rangle \in U : \\
\quad \text{IF } b \neq t \\
\qquad \text{THEN } \exists p \in b \,..\, (t-1) : \\
\qquad\qquad A' \in Partitions(A, p, b, t) \\
\qquad\quad \land \ U' = (U \setminus \{\langle b, t \rangle\}) \cup \{\langle b, p \rangle, \langle p+1, t \rangle\} \\
\qquad \text{ELSE } \quad A' = A \\
\qquad\quad \land \ U' = U \setminus \{\langle b, t \rangle\}
\end{array}
$$

And so on.

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\land\ \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } b \neq t \\
\qquad\quad \text{THEN } \exists p \in b \,..\, (t-1) : \\
\qquad\qquad\qquad A' \in Partitions(A, p, b, t) \\
\qquad\qquad \land\ U' = (U \setminus \{\langle b, t \rangle\}) \cup \{\langle b, p \rangle, \langle p+1, t \rangle\} \\
\\
\qquad\quad \text{ELSE} \qquad A' = A \\
\qquad\qquad \land\ U' = U \setminus \{\langle b, t \rangle\}
\end{array}
$$

A TLA$^+$ formula.

$Next$:

$$
\begin{array}{l}
\quad U \neq \{\} \\
\land \;\; \exists \langle b, t \rangle \in U : \\
\qquad \text{IF } \; b \neq t \\
\qquad\quad \text{THEN } \; \exists\, p \in b \,..\, (t-1) : \\
\qquad\qquad\qquad A' \in Partitions(A, p, b, t) \\
\qquad\qquad\quad \land \;\; U' = (U \setminus \{\langle b, t \rangle\}) \cup \{\langle b, p \rangle, \langle p+1, t \rangle\} \\
\\
\qquad\quad \text{ELSE } \qquad A' = A \\
\qquad\qquad\quad \land \;\; U' = U \setminus \{\langle b, t \rangle\}
\end{array}
$$

If you prefer pseudo-code...

PlusCal

PlusCal

Like a toy programming language.

PlusCal

Like a toy programming language.

Algorithm appears in a comment in a TLA$^+$ module.

PlusCal

Like a toy programming language.

Algorithm appears in a comment in a TLA$^+$ module.

An expression can be any TLA$^+$ expression.

PlusCal

Like a toy programming language.

Algorithm appears in a comment in a TLA^+ module.

An expression can be any TLA^+ expression.

Constructs for nondeterminism.

PlusCal

Like a toy programming language.

Algorithm appears in a comment in a TLA$^+$ module.

An expression can be any TLA$^+$ expression.

Constructs for nondeterminism.

Compiled to an easy to understand TLA$^+$ spec.

PlusCal

Like a toy programming language.

Algorithm appears in a comment in a TLA^+ module.

An expression can be any TLA^+ expression.

Constructs for nondeterminism.

Compiled to an easy to understand TLA^+ spec.

Can apply TLA^+ tools.

Programs that Run Forever

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

– Operating systems.

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

– Operating systems.

– Distributed systems.

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

- Operating systems.
- Distributed systems.

Few people can get them right by just thinking (and writing).

Programs that Run Forever

I’ve been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

– Operating systems.

– Distributed systems.

Few people can get them right by just thinking (and writing).

I’m not one of them.

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

- Operating systems.
- Distributed systems.

Few people can get them right by just thinking (and writing).

I'm not one of them.

We need tools to check what we do.

Programs that Run Forever

I've been talking about programs that compute a function.

Programs that run forever usually involve concurrency:

– Operating systems.

– Distributed systems.

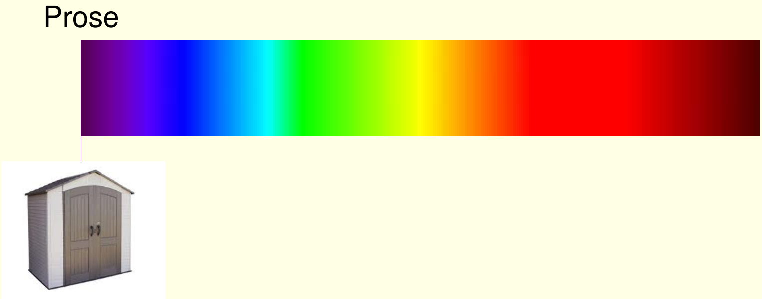
Few people can get them right by just thinking (and writing).

I'm not one of them.

We need tools to check what we do.

Use TLA$^+$/ PlusCal.

The Other 95%



Most code is really simple.

The Other 95%

```
/*************************************************************
* CLASS ResourceFileReader                                  *
*                                                           *
* A ResourceFileReader returns an object for reading a      *
* resource file, which is a file kept in the same           *
* directory as the tlatex.Token class.  The constructor     *
* takes a file name as argument.  The object's two public   *
* methods are                                               *
*                                                           *
*   getLine() : Returns the next line of the file as a      *
*               string.  Returns null after the last line.  *
*                                                           *
*   close()   : Closes the file.                            *
*************************************************************/
```

Why did I write that spec?

Why did I write that spec?

To be sure I knew **what** the code should do before writing it.

Why did I write that spec?

To be sure I knew **what** the code should do before writing it.

Without writing a spec, I only thought I knew what it should do.

Why did I write that spec?

To be sure I knew **what** the code should do before writing it.

Without writing a spec, I only thought I knew what it should do.

Later, I didn't have to read the code to know what it did.

Why did I write that spec?

To be sure I knew **what** the code should do before writing it.

Without writing a spec, I only thought I knew what it should do.

Later, I didn't have to read the code to know what it did.

General rule:
A spec of **what** the code does should say everything
that anyone needs to know to use the code.

Why did I write that spec?

To be sure I knew **what** the code should do before writing it.

Without writing a spec, I only thought I knew what it should do.

Later, I didn't have to read the code to know what it did.

How the code worked was too simple to require a spec.

What programmers should know about thinking.

What everyone should know about thinking.

What everyone should know about thinking.

Everyone thinks they think.

What everyone should know about thinking.

Everyone thinks they think.

If you don't write down your thoughts,
you're fooling yourself.

What programmers should know about thinking.

What programmers should know about thinking.

You should think before you code.

What programmers should know about thinking.

You should write before you code.

What programmers should know about thinking.

You should write before you code.

A spec is simply what you write before coding.

What code should you specify?

What code should you specify?

Any piece of code that someone else might want to use or modify.

What code should you specify?

Any piece of code that ~~someone else~~ you might want to use or modify in a month.

What code should you specify?

Any piece of code that ~~someone else~~ you might want to use or modify in a month.

It could be:

What code should you specify?

Any piece of code that ~~someone else~~ you might want to use or modify in a month.

It could be:

An entire program or system.

What code should you specify?

Any piece of code that ~~someone else~~ you might want to use or modify in a month.

It could be:

- An entire program or system.
- A class.

What code should you specify?

Any piece of code that ~~someone else~~ you might want to use or modify in a month.

It could be:

- An entire program or system.
- A class.
- A method.

What code should you specify?

you

Any piece of code that ~~someone else~~ might want to use or modify in a month.

It could be:

- An entire program or system.
- A class.
- A method.
- A tricky piece of code in a method.

What should you specify about the code?

What should you specify about the code?

What it does.

What should you specify about the code?

What it does.

Everything anyone needs to know to use it.

What should you specify about the code?

What it does.

Everything anyone needs to know to use it.

Maybe: how it does it.

What should you specify about the code?

What it does.

Everything anyone needs to know to use it.

Maybe: how it does it.

The algorithm / high-level design.

How should you think about / specify the code?

How should you think about / specify the code?

Above the code level.

How should you think about / specify the code?

Above the code level.

In terms of states and behaviors.

How should you think about / specify the code?

Above the code level.

In terms of states and behaviors.

Mathematically, as rigorously / formally as necessary.

How should you think about / specify the code?

Above the code level.

In terms of states and behaviors.

Mathematically, as rigorously / formally as necessary.

Perhaps with pseudo-code or PlusCal if specifying **how**.

How do you learn to write specs?

How do you learn to write specs?

By writing formal specs.

How do you learn to write formal specs?

How do you learn to write formal specs?

You learned to write programs by writing them, running them, and correcting your errors.

How do you learn to write formal specs?

You learned to write programs by writing them, running them, and correcting your errors.

You can learn to write formal specs by writing them, “running” them with a model checker, and correcting your errors.

How do you learn to write formal specs?

You learned to write programs by writing them, running them, and correcting your errors.

You can learn to write formal specs by writing them, “running” them with a model checker, and correcting your errors.

TLA$^+$ may not be the best language for your formal specification needs.

How do you learn to write formal specs?

You learned to write programs by writing them, running them, and correcting your errors.

You can learn to write formal specs by writing them, "running" them with a model checker, and correcting your errors.

TLA$^+$ may not be the best language for your formal specification needs.

But it's great for learning learning to think mathematically.

How do you connect the spec to the code?

How do you connect the spec to the code?

Comments connecting mathematical concepts and their implementation.

How do you connect the spec to the code?

Comments connecting mathematical concepts and their implementation.

Example:

Mathematical concept: graph

How do you connect the spec to the code?

Comments connecting mathematical concepts and their implementation.

Example:

Mathematical concept: graph

Implementation: array of node objects & array of link objects

What about coding?

What about coding?

Nothing I have said implies anything about how you should code.

What about coding?

Nothing I have said implies anything about how you should code.

You still have to think while you code.

What about coding?

Nothing I have said implies anything about how you should code.

You still have to think while you code.

What you write while coding is code.

What about coding?

Nothing I have said implies anything about how you should code.

You still have to think while you code.

What you write while coding is code.

I have nothing to say about how you should code.

What about coding?

Nothing I have said implies anything about how you should code.

You still have to think while you code.

What you write while coding is code.

I have nothing to say about how you should code.

Use any programming language you want.

What about coding?

Nothing I have said implies anything about how you should code.

You still have to think while you code.

What you write while coding is code.

I have nothing to say about how you should code.

Use any programming language you want.

Use any coding methodology you want:
test-driven development, agile programming, ...

You'll still have to test and debug.

You'll still have to test and debug.

Writing specs is an additional step.

You'll still have to test and debug.

Writing specs is an additional step.

It may save time by catching errors early,
when they're easier to correct.

You'll still have to test and debug.

Writing specs is an additional step.

It may save time by catching errors early,
when they're easier to correct.

It will improve your programming,
so you write better programs.

Why are programmers reluctant to write specs?

Why are programmers reluctant to write specs?

Writing is hard.

Why is writing hard?

Why is writing hard?

Writing requires thinking.

Why is writing hard?

Writing requires thinking.

Thinking is hard.

Why is writing hard?

Writing requires thinking.

Thinking is hard.

It’s easier to think your thinking.

Why is writing hard?

Writing requires thinking.

- Thinking is hard.
- It’s easier to think your thinking.

Writing is like running.

Why is writing hard?

Writing requires thinking.

- Thinking is hard.
- It's easier to think your thinking.

Writing is like running.

- The less you do it, the slower you are.

Why is writing hard?

- Writing requires thinking.
  - Thinking is hard.
  - It's easier to think your thinking.
- Writing is like running.
  - The less you do it, the slower you are.
  - You have to strengthen your writing muscles.

Why is writing hard?

- Writing requires thinking.
  - Thinking is hard.
  - It's easier to think your thinking.
- Writing is like running.
  - The less you do it, the slower you are.
  - You have to strengthen your writing muscles.
  - It takes practice.

Why is writing hard?

Writing requires thinking.

- Thinking is hard.
- It's easier to think your thinking.

Writing is like running.

- The less you do it, the slower you are.
- You have to strengthen your writing muscles.
- It takes practice.

It's easier to find an excuse not to.

What if the spec is wrong?

What if the spec is wrong?

Maybe you made a mistake.

What if the spec is wrong?

Maybe you made a mistake.

Maybe the requirements change,
or an enhancement is needed.

What if the spec is wrong?

Maybe you made a mistake.

Maybe the requirements change,
or an enhancement is needed.

The code will have to be changed,
maybe even before the program is finished.

What if the spec is wrong?

Maybe you made a mistake.

Maybe the requirements change,
or an enhancement is needed.

The code will have to be changed,
maybe even before the program is finished.

This eventually happens to all useful programs.

In an ideal world, a new spec would be written and the code completely rewritten.

In an ideal world, a new spec would be written
and the code completely rewritten.

In the real world, the code is patched
and maybe the spec is updated.

In an ideal world, a new spec would be written
and the code completely rewritten.

In the real world, the code is patched
and maybe the spec is updated.

If this is inevitable, why write specs?

Reason 1

Reason 1

Whoever has to modify the code will be grateful for every word or formula of spec you write.

Reason 1

Whoever has to modify the code will be grateful for every word or formula of spec you write.

And “whoever” may be you.

Reason 1

Whoever has to modify the code will be grateful for every word or formula of spec you write.

And “whoever” may be you.

That’s why you should update the spec when changing the code.

Reason 2

Reason 2

Every time code is patched, it becomes a little uglier, harder to understand, and harder to maintain.

Reason 2

Every time code is patched, it becomes a little uglier, harder to understand, and harder to maintain.

If you don’t start with a spec, every piece of the code you write is a patch.

Reason 2

Every time code is patched, it becomes a little uglier, harder to understand, and harder to maintain.

If you don't start with a spec, every piece of the code you write is a patch.

The program starts out ugly, hard to understand, and hard to maintain.

Reason 2

Every time code is patched, it becomes a little uglier, harder to understand, and harder to maintain.

If you don't start with a spec, every piece of the code you write is a patch.

The program starts out ugly, hard to understand, and hard to maintain.

"No battle was ever won according to plan

Reason 2

Every time code is patched, it becomes a little uglier, harder to understand, and harder to maintain.

If you don't start with a spec, every piece of the code you write is a patch.

The program starts out ugly, hard to understand, and hard to maintain.

> "No battle was ever won according to plan, but no battle was ever won without one."
>
> Dwight D. Eisenhower

Some people will tell you that writing specs is a waste of time.

Some people will tell you that writing specs is a waste of time.

In some situations it is.

Some people will tell you that writing specs is a waste of time.

In some situations it is. Sometimes there's no need to think about what you're doing.

Some people will tell you that writing specs is a waste of time.

In some situations it is. Sometimes there’s no need to think about what you’re doing.

But remember: when they’re telling you not to write a spec, they’re really telling you not to think.

Thinking doesn't guarantee that you won't make mistakes.

Thinking doesn’t guarantee that you won’t make mistakes.

Not thinking usually guarantees that you will.

To find out more about TLA$^+$, go to my home page and click on:

The TLA Web Page

To find out more about TLA$^+$, go to my home page and click on:

The TLA Web Page

Thank You