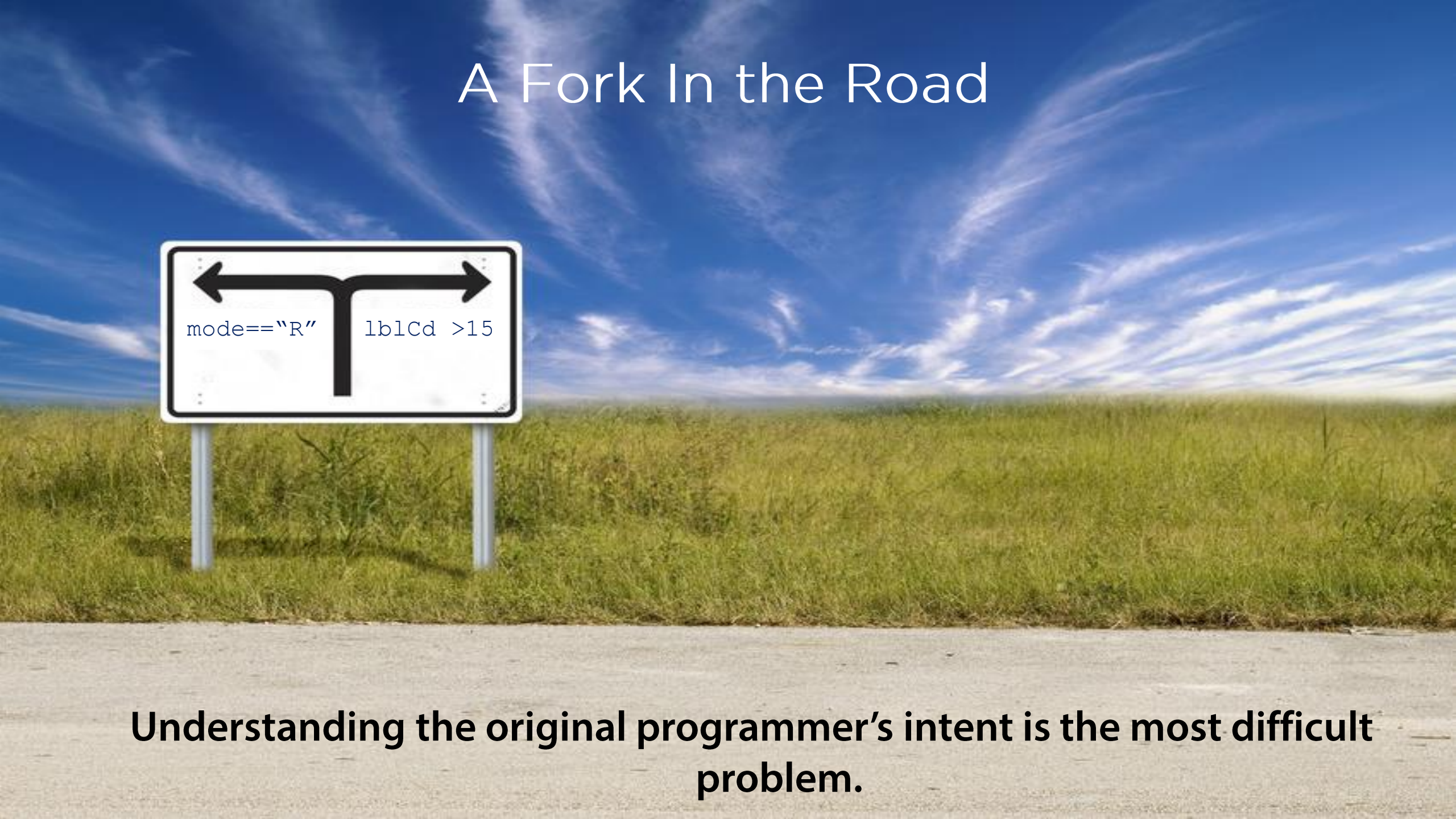
Conditionals

Cory House
PRINCIPAL CONSULTANT

@housecor www.cleancsharp.net

A Fork In the Road



Understanding the original programmer's intent is the most difficult problem.

Agenda

Clear intent

Bite-size logic

Use the right tool

Sometimes code isn’t the answer

Compare Booleans Implicitly

```
if (loggedIn == true)
```

```
if (loggedIn)
```

Assign Booleans Implicitly

```
bool goingToChipotleForLunch;
if (cashInWallet > 6.00)
{
  goingToChipotleForLunch = true;
}
else
{
  goingToChipotleForLunch = false;
}
```

```
bool goingToChipotleForLunch = cashInWallet > 6.00;
```

Be Positive
(Don’t not be anti-negative)

```
if (!isNotLoggedIn)
```

```
if (loggedIn)
```

Ternary is Beautiful

```
int registrationFee;
if (isSpeaker)
{
  registrationFee = 0;
}
else
{
  registrationFee = 50;
}
```

```
int registrationFee = isSpeaker ? 0 : 50;
```

Be Strongly Typed, Not “Stringly” Typed

```
if (employeeType == "manager")
```

```
if (employee.type == EmployeeType.Manager)
```

Magic Numbers

Which would you rather read?
Sally went to the #12 dealer to buy a #19 #515.
Sally went to the Ferrari dealer to buy an Enzo.

Magic Numbers

```
if (age > 21)
{
  // body here
}
```

```
const int legalDrinkingAge = 21;

if (age > legalDrinkingAge)
{
  // body here
}
```

```
if (status == 2)
{
  // body here
}
```

```
if (status == Status.active)
{
  // body here
}
```

Complex Conditionals

```
if (car.Year > 1980
   && (car.Make == "Ford" || car.Make == "Chevrolet")
   && car.Odometer < 100000
   && car.Vin.StartsWith("V2") || car.Vin.StartsWith("IA3"))
{
   // Do lots of things here
}
```

1. **Intermediate variables**
2. **Encapsulate via function**

Intermediate Variables

```
if (employee.Age > 55
    && employee.YearsEmployed > 10
    && employee.IsRetired == true)
{
    // body here
}
```

← What question is this asking?

```
bool eligibleForPension = employee.Age > 55
    && employee.YearsEmployed > 10
    && employee.IsRetired == true;
```

Encapsulate Complex Conditionals

```
//Check for valid file extensions, confirm is admin or active
if ( (fileExt == ".mp4"
    || fileExt == ".mpg"
    || fileExt == ".avi")
    && (isAdmin == 1 || isActiveFile))
```

```
private bool ValidFileRequest(string fileExtension, bool isActiveFile, bool isAdmin)
{
  return (fileExt == ".mp4"
    || fileExt == ".mpg"
    || fileExt == ".avi")
    && (isAdmin == 1 || isActiveFile))
}
```

Encapsulate Complex Conditionals

```
//Check for valid file extensions, confirm is admin or active
if ( (fileExt == ".mp4"
    || fileExt == ".mpg"
    || fileExt == ".avi")
    && (isAdmin == 1 || isActiveFile))
```

```
private bool ValidFileRequest(string fileExtension, bool isActiveFile, bool isAdmin)
{
  var validFileExtensions = new List<string>() { "mp4", "mpg", "avi" };

  bool validFileType = validFileExtensions.Contains(fileExtension);
  bool userIsAllowedToViewFile = isActiveFile || isAdmin;

  return validFileType && userIsAllowedToViewFile;
}
```

Favor Polymorphism Over Switch

```
public void LoginUser(User user)
{
  switch (user.Status)
  {
    case Status.Active:
      // active user logic
      break;
    case Status.Inactive:
      // inactive user logic
      break;
    case Status.Locked:
      // locked user logic
      break;
  }
}
```

```
public void LoginUser(User user)
{
  user.Login();
}


public abstract class User
{
  public string FirstName;
  public string LastName;
  public int Status;
  public int AccountBalance;

  public abstract void Login();
}
```

Favor Polymorphism over Switch

```
ublic class ActiveUser : User

public override void Login()
{
  //Active user logic here
}
```

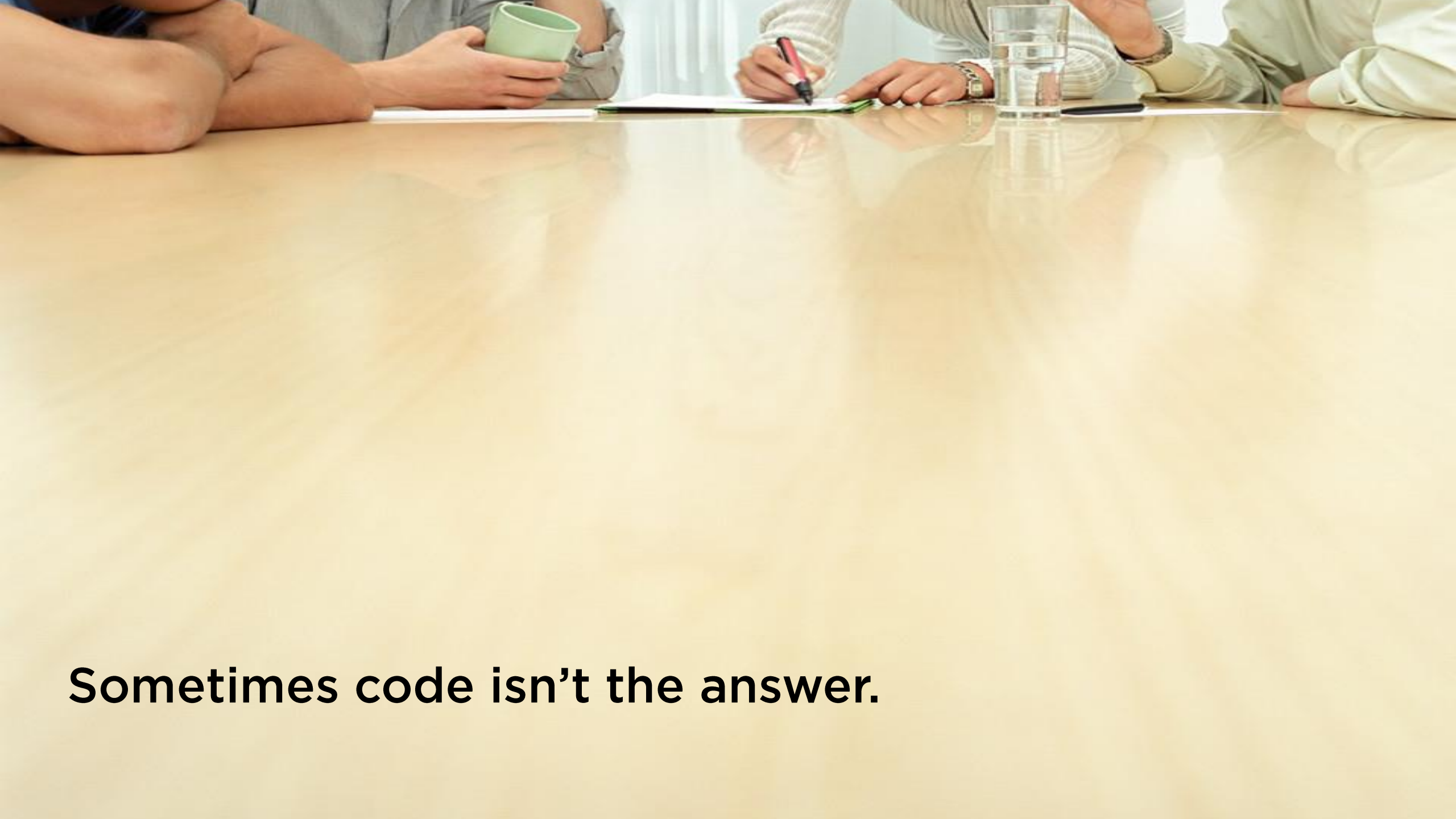
```
public class LockedUser : User
{
  public override void Login()
  {
    //Locked user logic here
  }
}
```

```
ublic class InactiveUser : User

public override void Login()
{
  //Inactive user logic here
}
```

Be Declarative

```
st<User> matchingUsers = new List<User>();

reach (var user in users)

f (user.AccountBalance < minAccountBalance
  && user.Status == Status.Active)

 matchingUsers.Add(user);


turn matchingUsers;
```

```
turn users
Where(u => u.AccountBalance < minAccountBalance)
Where(u => u.Status == Status.Active);
```



Sometimes code isn’t the answer.

Table Driven Methods

```
f (age < 20)
  return 345.60m;
lse if (age < 30)
  return 419.50m;
lse if (age < 40)
  return 476.38m;
lse if (age < 50)
  return 516.25m;
```

InsuranceRate table

InsuranceRateId	MaximumAge	Rate
1	20	346.60
2	30	420.50
3	40	476.38
4	50	516.25

```
return Repository.GetInsuranceRate(age);
```

Examples

- Insurance rates
- Pricing structures
- Complex and dynamic business rules

Table Driven Methods

Great for dynamic logic

Avoids hard coding

Write less code

Avoids complex data structures

Make changes without a code deployment

Summary

Strive for clear intent

Assign and compare booleans implicitly

Prefer positive conditionals

Prefer ternaries over if/else

Be strongly typed via constants, enums

Avoid magic numbers

Avoid complex conditionals

- Declare variables, extract methods

Prefer declarative over iterative

Consider using the DB

Next up: Methods