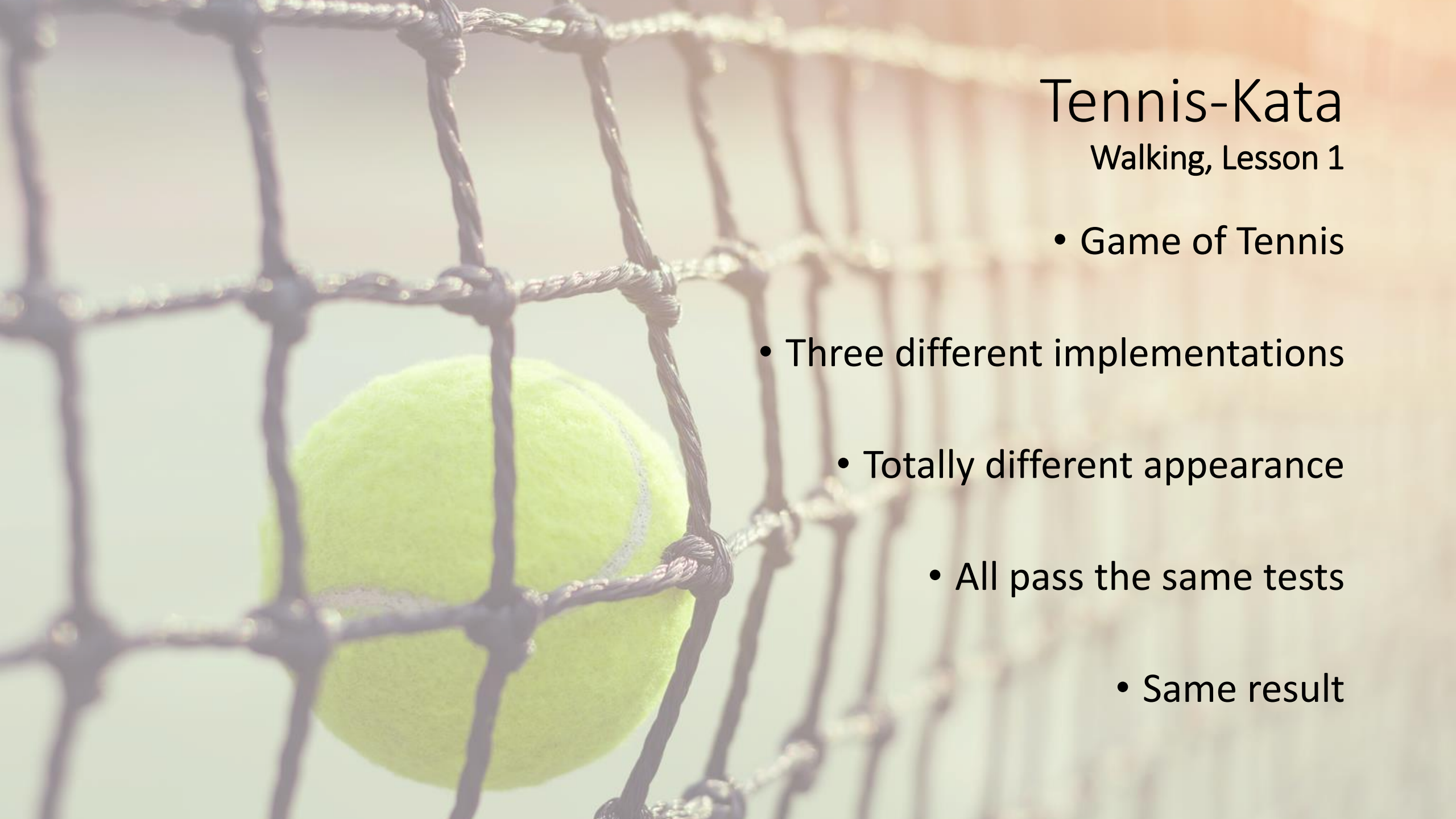
Markus Doggweiler, Alcor – Flying, April 22

Definition of Refactoring

“to restructure software by applying a series of changes without changing its observable behavior.”

Martin Fowler



Tennis-Kata

Walking, Lesson 1

- Game of Tennis
- Three different implementations
- Totally different appearance
- All pass the same tests
- Same result

```
public interface TennisGame {
    void wonPoint(String playerName);
    String getScore();
}
```

```
public String getScore() {
    String score = "";
    int tempScore=0;
    if (m_score1==m_score2)
    {
        switch (m_score1)
        {
            case 0:
                score = "Love-All";
                break;
            case 1:
                score = "Fifteen-All";
                break;
            case 2:
                score = "Thirty-All";
                break;
            default:
                score = "Deuce";
                break;

        }
    }
    else if (m_score1>=4 || m_score2>=4)
    {
        int minusResult = m_score1-m_score2;
        if (minusResult==1) score ="Advantage player1";
        else if (minusResult ==-1) score ="Advantage player2";
        else if (minusResult>=2) score = "Win for player1";
        else score ="Win for player2";
    }
    else
    {
        for (int i=1; i<3; i++)
        {
            if (i==1) tempScore = m_score1;
            else { score+="-"; tempScore = m_score2;}
            switch(tempScore)
            {
                case 0:
                    score+="Love";
                    break;
                case 1:
                    score+="Fifteen";
                    break;
                case 2:
                    score+="Thirty";
                    break;
                case 3:
                    score+="Forty";
                    break;
            }
        }
    }
    return score;
}
```

```
public String getScore(){
    String score = "";
    if (P1point == P2point && P1point < 4)
    {
        if (P1point==0)
            score = "Love";
        if (P1point==1)
            score = "Fifteen";
        if (P1point==2)
            score = "Thirty";
        score += "-All";
    }
    if (P1point==P2point && P1point>=3)
        score = "Deuce";

    if (P1point > 0 && P2point==0)
    {
        if (P1point==1)
            P1res = "Fifteen";
        if (P1point==2)
            P1res = "Thirty";
        if (P1point==3)
            P1res = "Forty";

        P2res = "Love";
        score = P1res + "-" + P2res;
    }
    if (P2point > 0 && P1point==0)
    {
        if (P2point==1)
            P2res = "Fifteen";
        if (P2point==2)
            P2res = "Thirty";
        if (P2point==3)
            P2res = "Forty";

        P1res = "Love";
        score = P1res + "-" + P2res;
    }

    if (P1point>P2point && P1point < 4)
    {
        if (P1point==2)
            P1res="Thirty";
        if (P1point==3)
            P1res="Forty";
        if (P2point==1)
            P2res="Fifteen";
        if (P2point==2)
            P2res="Thirty";
        score = P1res + "-" + P2res;
    }
    if (P2point>P1point && P2point < 4)
    {
        if (P2point==2)
            P2res="Thirty";
        if (P2point==3)
            P2res="Forty";
        if (P1point==1)
            P1res="Fifteen";
        if (P1point==2)
            P1res="Thirty";
        score = P1res + "-" + P2res;
    }
    if (P1point > P2point && P2point >= 3)
```

```
public String getScore() {
    String s;
    if (p1 < 4 && p2 < 4 && !(p1 + p2 == 6)) {
        String[] p = new String[]{"Love", "Fifteen", "Thirty", "Forty"};
        s = p[p1];
        return (p1 == p2) ? s + "-All" : s + "-" + p[p2];
    } else {
        if (p1 == p2)
            return "Deuce";
        s = p1 > p2 ? p1N : p2N;
        return ((p1-p2)*(p1-p2) == 1) ? "Advantage " + s : "Win for " + s;
    }
}
```

And more...

Game 2

- Very looooong
- Lots of redundant hardcoded stuff (e.g. scores)
- Lots of if's

```
if (P1point == P2point && P1point < 4)
{
    if (P1point==0)
        score = "Love";
    if (P1point==1)
        score = "Fifteen";
    if (P1point==2)
        score = "Thirty";
    score += "-All";
}
if (P1point==P2point && P1point>=3)
    score = "Deuce";

if (P1point > 0 && P2point==0)
{
    if (P1point==1)
        P1res = "Fifteen";
    if (P1point==2)
        P1res = "Thirty";
    if (P1point==3)
        P1res = "Forty";

    P2res = "Love";
    score = P1res + "-" + P2res;
}
...
```

Game 3

- Very short
- Cryptic
- Lots of oneliners (calculations, ternaries, etc.)

```
public String getScore() {
    String s;
    if (p1 < 4 && p2 < 4 && !(p1 + p2 == 6)) {
        String[] p = new String[]{"Love", "Fifteen", "Thirty", "Forty"};
        s = p[p1];
        return (p1 == p2) ? s + "-All" : s + "-" + p[p2];
    } else {
        if (p1 == p2)
            return "Deuce";
        s = p1 > p2 ? p1N : p2N;
        return ((p1-p2)*(p1-p2) == 1) ? "Advantage " + s : "Win for " + s;
    }
}
```

Game 1

- A mix of the other two
- Lots of ifs
- Lots of hardcoded scores (there's more near the end 😍)

```
if (m_score1==m_score2)
{
    switch (m_score1)
    {
        case 0:
            score = "Love-All";
            break;
        case 1:
            score = "Fifteen-All";
            break;
        case 2:
            score = "Thirty-All";
            break;
        default:
            score = "Deuce";
            break;

    }
}
else if (m_score1 >= 4 || m_score2 >= 4)
{
    int minusResult = m_score1-m_score2;
    if (minusResult==1) score ="Advantage player1";
    else if (minusResult ==-1) score ="Advantage player2";
    else if (minusResult >= 2) score = "Win for player1";
    else score ="Win for player2";
}
else
{
```

Original implementations very different

What about (refactored) results?

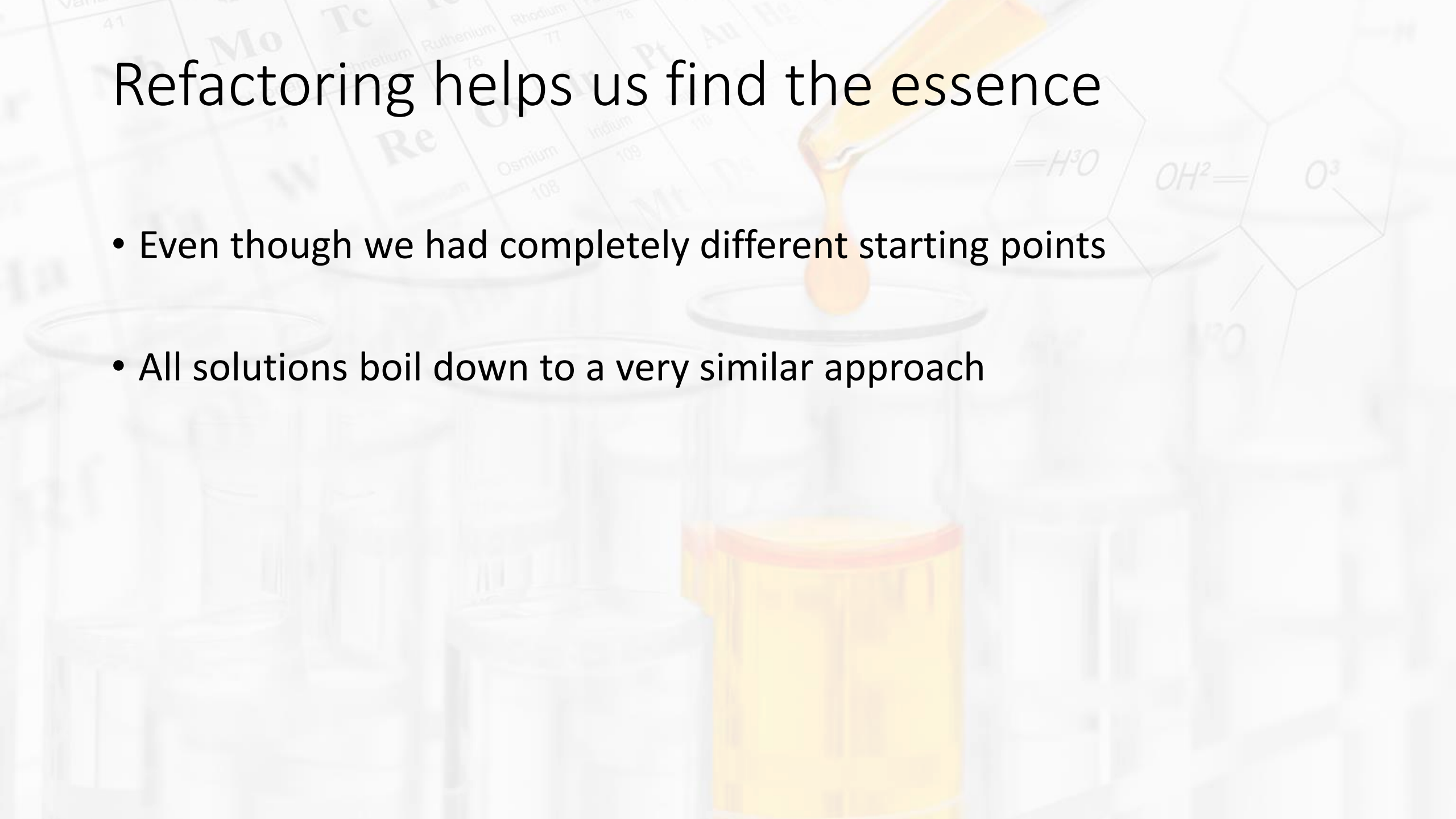
```
public String getScore() {
    if (scoreIsEqual()) {
        return getEqualScore();
    }

    if (scoreIsRegular()) {
        return getRegularScore();
    }

    if (scoreIsAdvantage()) {
        return getAdvantageScore();
    }

    return getWinningScore();
}
```

```
public String getScore() {
    if (scoreIsRegular()) {
        return getRegularScore();
    }

    if (scoreIsEqual()) {
        return DEUCE;
    }

    if (isScoreAdvantage()) {
        return String.format(
                ADVANTAGE_TEMPLATE,
                getLeadingPlayer()
        );
    }

    return String.format(WIN_TEMPLATE, getLeadingPlayer());
}
```

```
public String getScore() {
    if (isEqualScore()) {
        return getEqualScore();
    }

    if (isRegularScore()) {
        return SCORES[player1Points] + "-" + SCORES[player2Points];
    }

    if (hasWinner()) {
        return String.format(WIN_TEMPLATE, getPlayerInFront());
    }

    return String.format(ADVANTAGE_TEMPLATE, getPlayerInFront());
}
```

Refactoring helps us find the essence

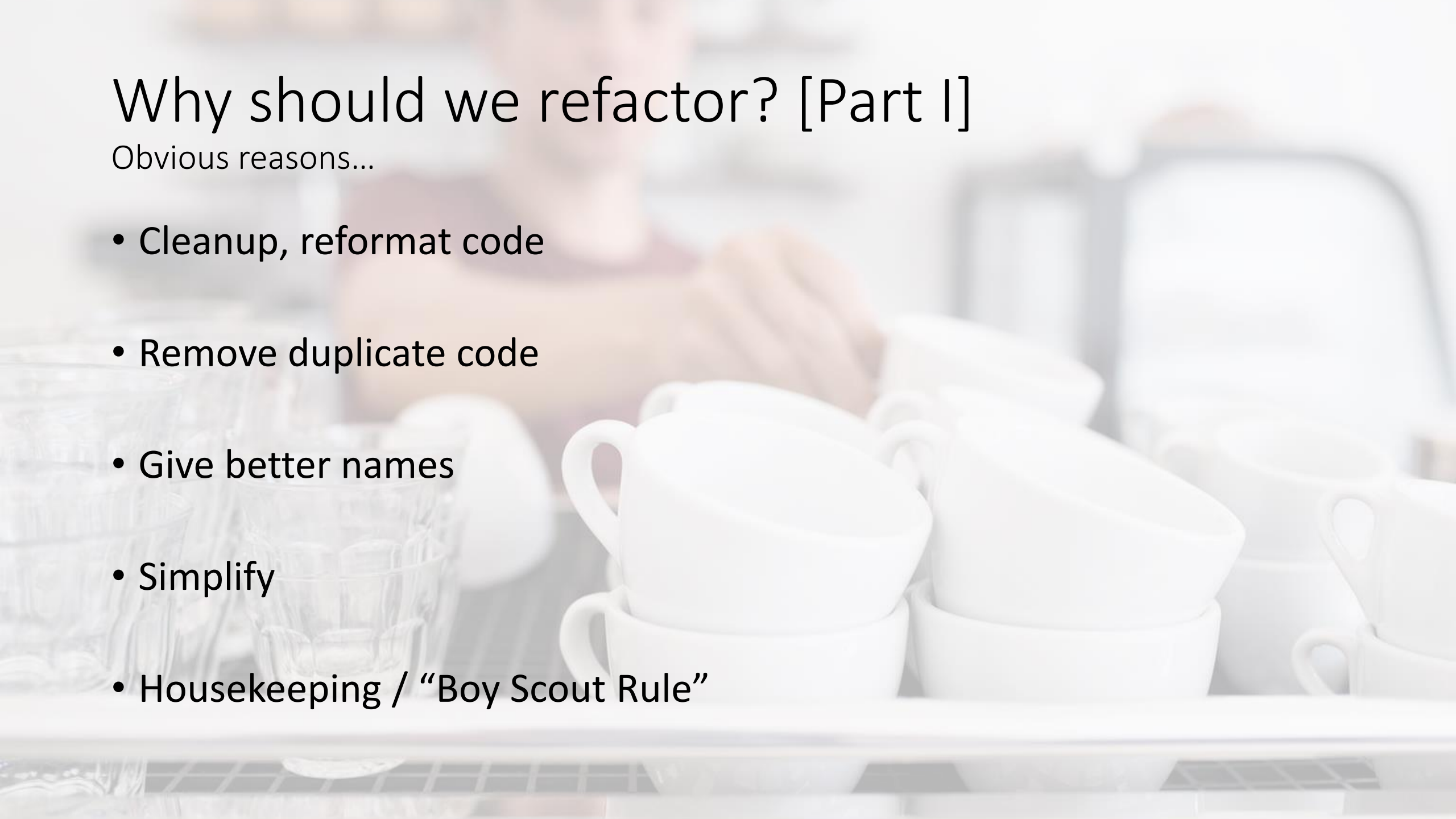
- Even though we had completely different starting points
- All solutions boil down to a very similar approach



Why should we refactor? [Part I]

Obvious reasons…

- Cleanup, reformat code
- Remove duplicate code
- Give better names
- Simplify
- Housekeeping / “Boy Scout Rule”



Why should we refactor? [Part II]

Hidden benefits…

Why should we refactor? [Part II]

Hidden benefits…

- When we **implement**, we deal with (technical) **details**
  - “We are in the hole”
  - Dealing with “dirty” details

- When we **refactor**, we deal with **higher level concerns**
  - “We get out of the hole, and have a look from a higher level”

Why should we refactor? [Part III]

Because we can!

Herbert can’t…

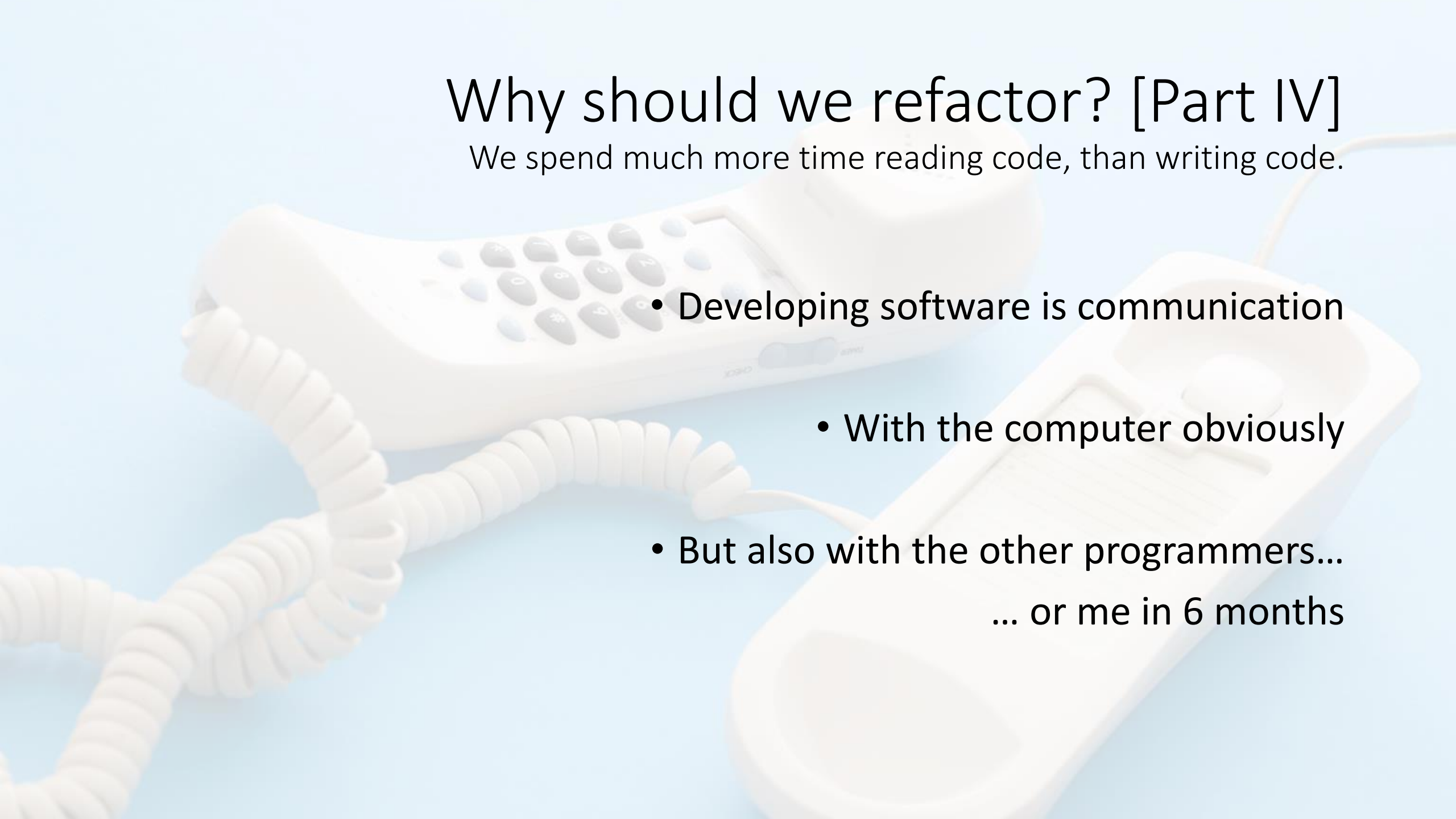
… and so can’t these guys:

Maybe that’s why it’s called “soft”-ware?

- A huge advantage we have
- In most other jobs this is not possible
- We can easily change working stuff

Why should we refactor? [Part IV]

We spend much more time reading code, than writing code.



- Developing software is communication
- With the computer obviously
- But also with the other programmers…

… or me in 6 months

Reason to refactor - Recap

- Because we can
- Find the essence
- (Re)think about problem/solution on different levels
- Improve communication

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

Martin Fowler

In other words:

When we refactor, we have the chance to be a good programmer.

Any questions?

Now…

…or later at markus.doggweiler@gmail.com