

# Codemetriken - Statisch

Statische Codeanalyse

Robert Wimmer

6. April 2019



# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools

# LOC

## I Lines of Code (LOC)

- Lines of Code (LOC)
- Source Lines of Code (SLOC)
- Comment Lines of Code (CLOC)
- Non-Comment Lines of Code (NCLOC)
- Logical Lines of Code (LLOC)

```
int main(int argc, char** argv)
{
    // The Absolute Truth!!!
    if (argc > 1)
    {
        return 42;
    }
    return 0;
}
```


```
LOC:    9
SLOC:   8
CLOC:   1
NCLOC:  4
LLOC:   3
```

# LOC

## I Lines of Code (LOC)

- Keine Aussage über Qualität des Source Codes
- Verwendete Bibliotheken?
- Abhängig von der Sprache
- Funktionalität vs. SLOC?
  
- Sinnvoll: maximale Modulgröße definieren
- Verhältnis: Kommentar zu Code

# LOC Vergleich Größenordnung



- Space Shuttle	400k
- Mars Curiosity Rover	4.000k
- Linux Kernel 5.0	26.203k
- Windows XP	40.000k
- Mac OS X 10.4	86.000k

# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools

# McCabe – Zyklomatische Komplexität

## ■ Zyklomatische Komplexität / Cyclomatic Complexity Number (CCN)

- Komplexität eines SW-Moduls
- Verständlichkeit
- **Vorgriff:** Berechnet # Testfälle für Branch Coverage
- Findet Hotspots
- Richtwert von McCabe: 10

# McCabe – Berechnung CCN

I CCN = 1

- Anzahl der Binärverzweigungen (if, and, **or**, for, while, case) + 1

```
int main(int argc, char** argv)
{
    return 0;
}
```



# McCabe – Berechnung CCN

I CCN = 2

```
int main(int argc, char** argv)
{
    if (argc > 1)
    {
        return 42;
    }
    return 0;
}
```

# McCabe – Berechnung CCN

```
void bubblesort(int *array, int length)
{
    int i, j;
    for (i = 0; i < length - 1; ++i)
    {
        for (j = 0; j < length - i - 1; ++j)
        {
            if (array[j] > array[j + 1])
            {
                int tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
            }
        }
    }
}
```

- [Creative Commons Attribution-ShareAlike License](#)
- [https://en.wikibooks.org/wiki/Algorithm\\_Implementation/Sorting/Bubble\\_sort#C\\_2](https://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Bubble_sort#C_2)

# McCabe – Berechnung CCN

```
char* monatsName(int monat) {  
    switch(monat) {  
        case 1: return "Januar";  
        case 2: return "Februar";  
        case 3: return "Maerz";  
        case 4: return "April";  
        case 5: return "Mai";  
        case 6: return "Juni";  
        case 7: return "Juli";  
        case 8: return "August";  
        case 9: return "September";  
        case 10: return "Oktober";  
        case 11: return "November";  
        case 12: return "Dezember";  
        default:  
            return "(unbekannter Monat)";  
    }  
}
```

# McCabe – Berechnung des CCN

CCN = 22 ...

```
val = (rd_param->rawsamples[i]>>16)&0xffff;
if ((val & 0xf000) == 0x3000) {
    val &= ~0xf000;
    if (val & 0x800)
        val |= 0xf000;
}
//Imaginarteil
rd_param->q_samp_uncorr[i] = val;
val = rd_param->rawsamples[i]&0xffff;
if ((val & 0xf000) == 0xb000) {
    val &= ~0xf000;
    if (val & 0x800)
        val |= 0xf000;
}
//Realteil
rd_param->i_samp_uncorr[i] = val + 2;
}
//revert sign extension made by transfer algorithm
//1 part
std::valarray<int32_t> i_corr_array {rd_param->i_samp_uncorr, rd_param->num_samples};
for (int i = 15; i > 10; i--)
{
    std::valarray<int32_t> is_array {1 << i, rd_param->num_samples};
    std::valarray<bool> comp = (i_corr_array>=is_array);

    int anz_elements2correct = std::count( &comp[0], &comp[ comp.size() ], true);
    i_corr_array[comp] -= std::valarray<int32_t>{1 << (i+1), anz_elements2correct};
    std::valarray<int32_t> is_array2 {-(1 << i), rd_param->num_samples};
    std::valarray<bool> comp2 = (i_corr_array<is_array2);
    anz_elements2correct = std::count( &comp2[0], &comp2[ comp2.size() ], true);
    i_corr_array[comp2] += std::valarray<int32_t>{1 << (i+1), anz_elements2correct};
}

// Q Part
std::valarray<int32_t> q_corr_array {rd_param->q_samp_uncorr, rd_param->num_samples};
for (int i = 15; i > 10; i--)
{
    std::valarray<int32_t> is_array {1 << i, rd_param->num_samples};
    std::valarray<bool> comp = (q_corr_array>=is_array);

    int anz_elements2correct = std::count( &comp[0], &comp[ comp.size() ], true);
    q_corr_array[comp] -= std::valarray<int32_t>{1 << (i+1), anz_elements2correct};
    std::valarray<int32_t> is_array2 {-(1 << i), rd_param->num_samples};
    std::valarray<bool> comp2 = (q_corr_array<is_array2);
    anz_elements2correct = std::count( &comp2[0], &comp2[ comp2.size() ], true);
    q_corr_array[comp2] += std::valarray<int32_t>{1 << (i+1), anz_elements2correct};
}

for (int i=0; i < (int)rd_param->num_samples;i++)
{
    //zero exception avoiding
    if (i_corr_array[i] == 0) i_corr_array[i]=1;
    if (q_corr_array[i] == 0) q_corr_array[i]=1;
    //Get indices of frames
    if (i_corr_array[i] == -1364 && q_corr_array[i] == 1365) (*frame_indices).append(i);
    //make exponent correction
    if (this_exp < 0)
    {
        i_corr_array[i] = i_corr_array[i] << (-this_exp);
        q_corr_array[i] = q_corr_array[i] << (-this_exp);
    }
    else if (this_exp > 0)
    {
        i_corr_array[i] = i_corr_array[i] >> (this_exp);
        q_corr_array[i] = q_corr_array[i] >> (this_exp);
    }
    //make absolute of i and q data
    (*i_samp)[i] = (double)pow(i_corr_array[i]/4096.0, 2);
    (*q_samp)[i] = (double)pow(q_corr_array[i]/4096.0, 2);
    (*betr)[i] = (double)10*log10(sqrt((*i_samp)[i]*(*q_samp)[i]));
}
//when receive status is negative the devices fifo is empty: this is an underflow, caused
//by too many read calls (may appear basically both the framerate is to high and the averaging loops
//setting is to high: this causes too many readings)
if (bladeSt->receivedatastatus == -1)
{
    ++bladeSt->rxTimeoutCounter;
}
else
{
    bladeSt->rxTimeoutCounter=0;
}
if (bladeSt->receivedatastatus < 0)
{
    ++bladeSt->rxTimeoutCounter;
}
}
else
{
    generateRandomRXData(betr, frame_indices);
    res = 0;
    bladeSt->receivedatastatus = -1;
}
return res;
}
```

# McCabe – Berechnung CCN

## I Funktionen mit $CCN > 100$ oder gar $CCN > 1000$ !

- Problem
  - Schwer verständlich
  - Fehleranfällig
  - Hoher Testaufwand
- Lösung
  - Umstrukturieren / Refactoring

# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools

# Kognitive Komplexität

## I Problem McCabe: Keine Aussage über Lesbarkeit

- Niedrige zyklomatische Komplexität
  - kein Hotspot laut McCabe
  - dennoch eingeschränkte Lesbarkeit

```
int sumOfPrimes(int max) {                // +1
    int total = 0;
    OUT: for (int i = 1; i <= max; ++i) { // +1
        for (int j = 2; j < i; ++j) {      // +1
            if (i % j == 0) {              // +1
                continue OUT;
            }
        }
        total += i;
    }
    return total;
}                                           // Cyclomatic Complexity 4
```

# Kognitive Komplexität

## I Beispiel Kognitive Komplexität

- U.a. Verschachtelungstiefe stärker mit einberechnet

```
int sumOfPrimes(int max) {  
    int total = 0;  
    OUT: for (int i = 1; i <= max; ++i) { // +1  
        for (int j = 2; j < i; ++j) {      // +2  
            if (i % j == 0) {              // +3  
                continue OUT;             // +1  
            }  
        }  
        total += i;  
    }  
    return total;  
} // Cognitive Complexity 7
```



# Kognitive Komplexität

## Grundidee

### I Drei Grundregeln

- Strukturen ignorieren, die viele Statements einfach lesbar darstellen (simple switch's, ...)
- Jedes break im linearen Codefluss erhöht die Komplexität um 1
- Jede Verschachtelung von Codefluss unterbrechenden Strukturen erhöht die Komplexität

### I Zusätzliche vier Typen, die die Komplexität erhöht

- Verschachtelung – Kontrollflussstrukturen ineinandergelagert (for, if, ...)
- Strukturell – bewertet zusätzlich Kontrollflussstrukturen die verschachtelt sind
- Fundamental – nicht verschachtelte Statements
- Hybrid – Kontrollflussstrukturen, die zwar nicht verschachtelt sind, aber die Verschachtelungstiefe erhöht (nebeneinanderliegende for-Schleifen, ...)

# Kognitive Komplexität

## Beispiel

### I Auszug aus Dokument

#### - COGNITIVE COMPLEXITY

A new way of measuring understandability

I <https://www.sonarsource.com/docs/CognitiveComplexity.pdf>

## Appendix C: Examples

From `org.sonar.java.resolve.JavaSymbol.java` in the SonarJava analyzer:

```
@Nullable
private MethodJavaSymbol overriddenSymbolFrom(ClassJavaType classType) {
    if (classType.isUnknown()) { // +1
        return Symbols.unknownMethodSymbol;
    }

    boolean unknownFound = false;
    List<JavaSymbol> symbols = classType.getSymbol().members().lookup(name);
    for (JavaSymbol overrideSymbol : symbols) { // +1
        if (overrideSymbol.isKind(JavaSymbol.MTH) // +2 (nesting = 1)
            && !overrideSymbol.isStatic()) { // +1

            MethodJavaSymbol methodJavaSymbol = (MethodJavaSymbol)overrideSymbol;
            if (canOverride(methodJavaSymbol)) { // +3 (nesting = 2)
                Boolean overriding = checkOverridingParameters(methodJavaSymbol,
                                                                classType);
                if (overriding == null) { // +4 (nesting = 3)
                    if (!unknownFound) { // +5 (nesting = 4)
                        unknownFound = true;
                    }
                } else if (overriding) { // +1
                    return methodJavaSymbol;
                }
            }
        }
    }

    if (unknownFound) { // +1
        return Symbols.unknownMethodSymbol;
    }

    return null;
} // total complexity = 19
```

# Inhaltsverzeichnis

## I Statische Codeanalyse

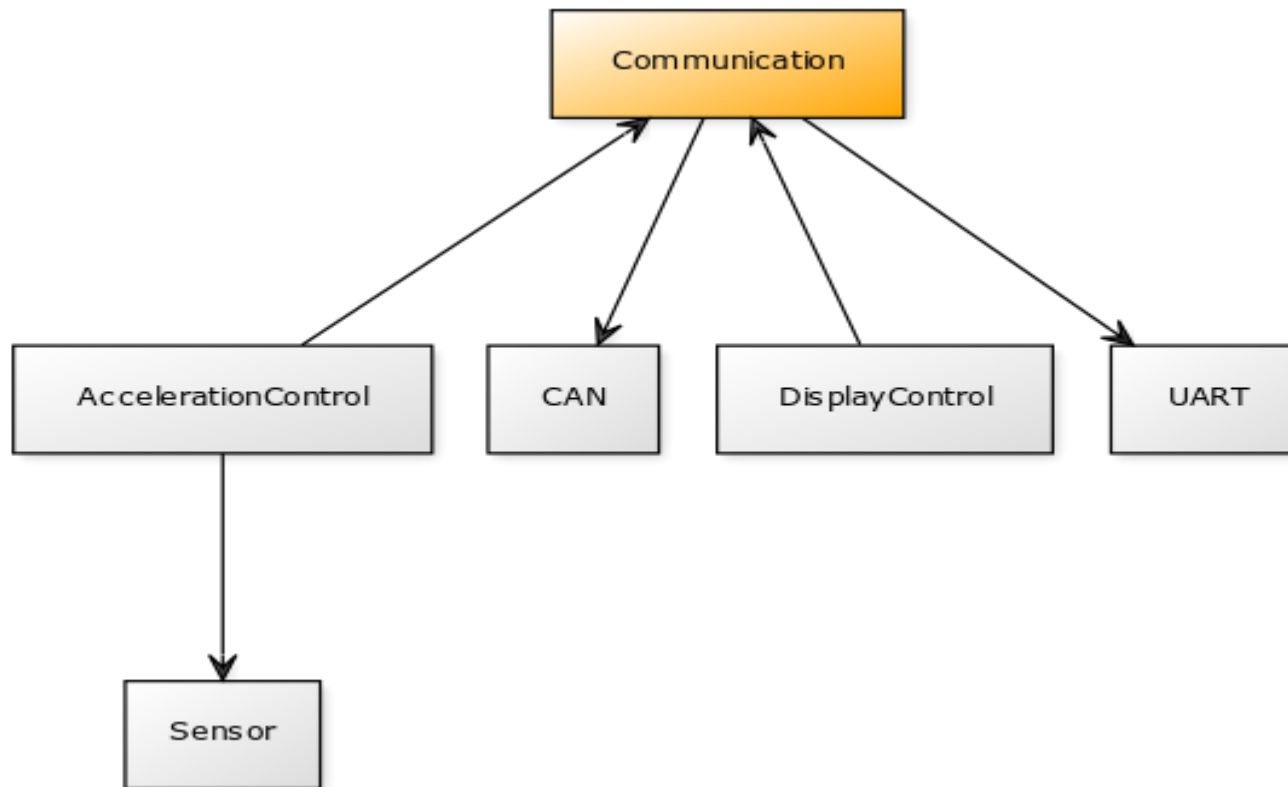
- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools

# Kopplung von Modulen

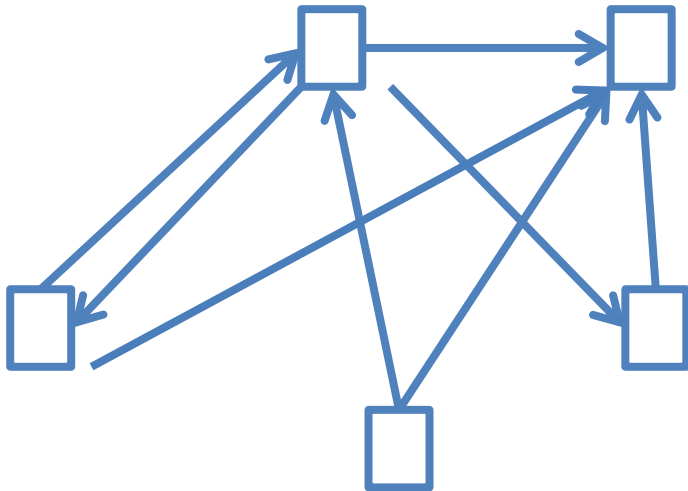
- Kopplung: ein Maß für die Abhängigkeit von SW-Modulen



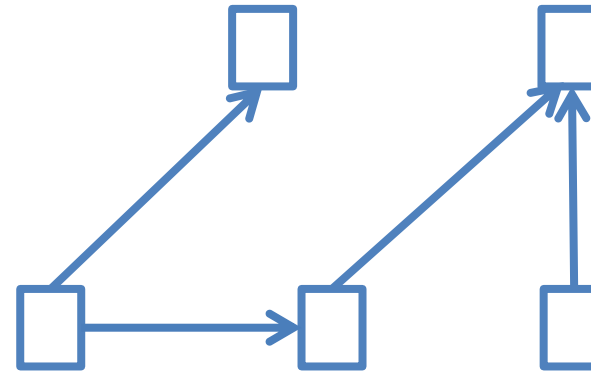
# Kopplung von Modulen

## Coupling Between Objects (CBO)

### starke Kopplung

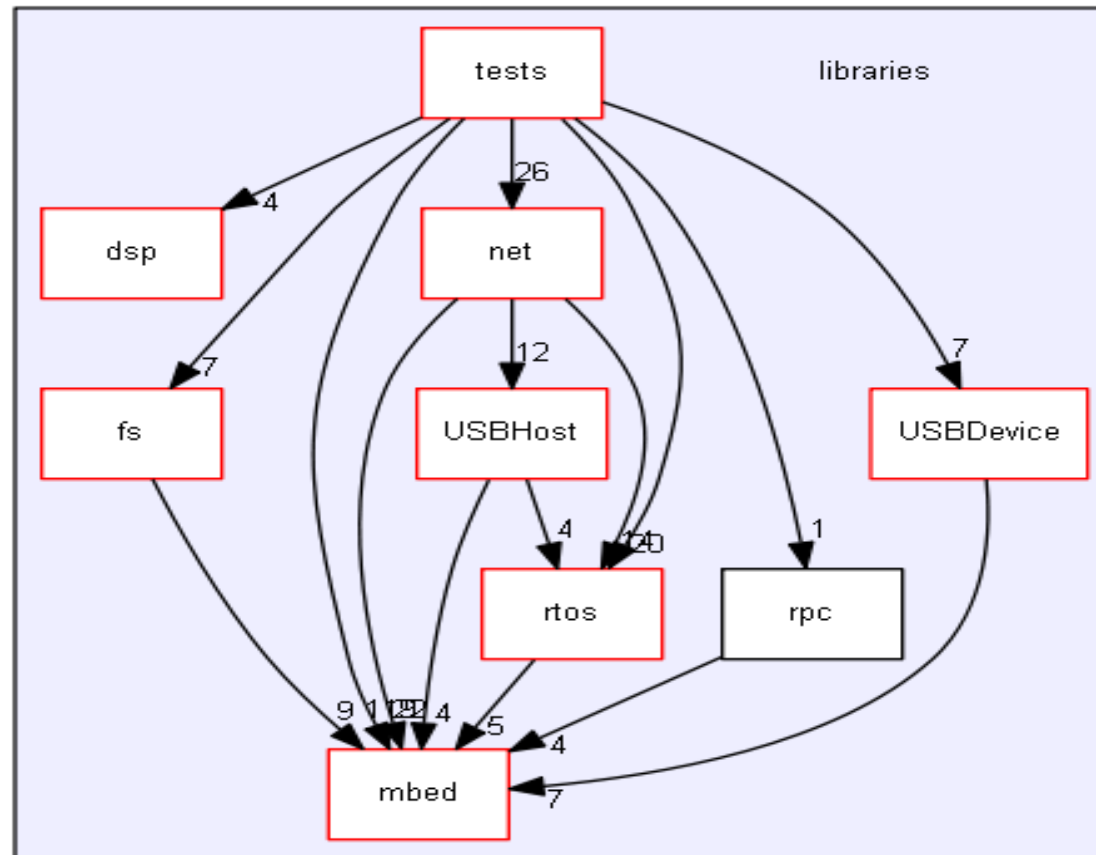


### schwache Kopplung



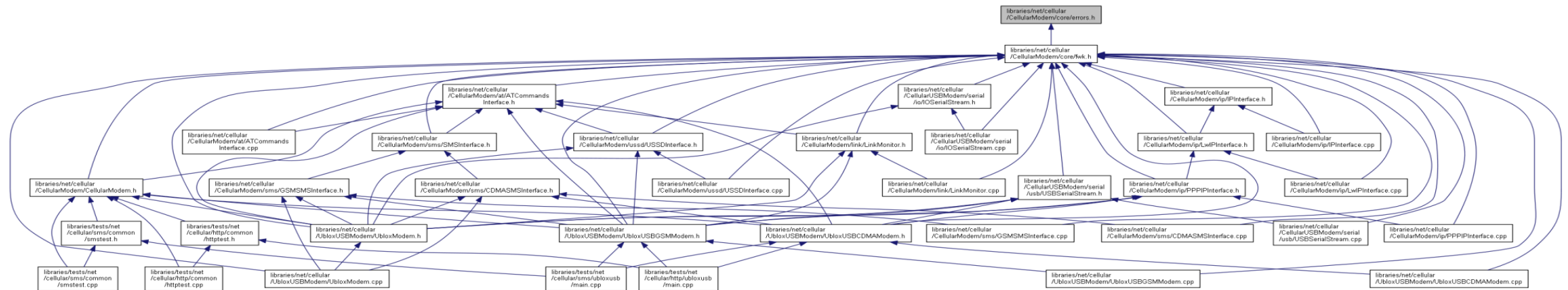
# Kopplung von Modulen

## ■ Beispiel: mbed Abhängigkeiten auf Bibliothek-Ebene



# Kopplung von Modulen

## Unübersichtliches Beispiel: Abhängigkeiten auf File-Ebene



# Kopplung von Modulen

## I Offensichtliche Abhängigkeiten

- #includes, #define, #if, #elseif
- Vererbung
- Hardware: Prozessor und Peripherie

## I Versteckte Abhängigkeiten

- extern declarations
- Speicherzugriff über absolute Adressen
- compilerspezifische Konstrukte
- Abhängigkeit von Tools
  - SVN-Konstanten
  - CRC-Berechnung
  - Code-Generatoren



# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

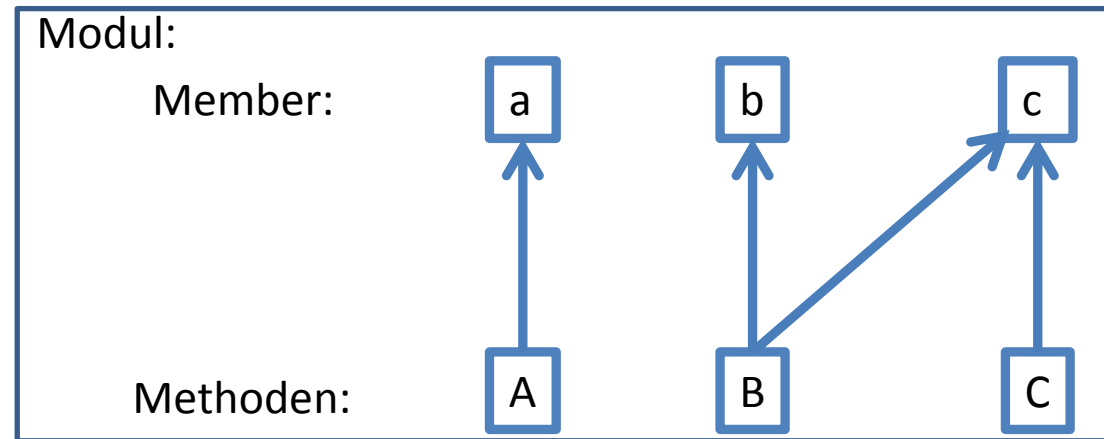
## I Tools

# Kohäsion

- I Kohäsion / Bindekraft
- I Starke Kohäsion
  - eine Aufgabe pro Modul
- I Schwache Kohäsion
  - Zuständigkeit der Aufgaben unklar
  - Code-Duplizierung
- I Design Guidelines
  - Single Responsibility Principle (SRP)
  - Don't Repeat Yourself (DRY)

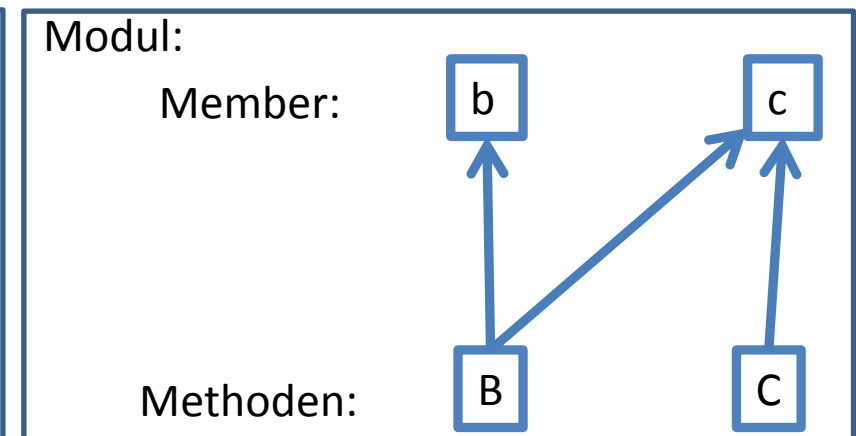
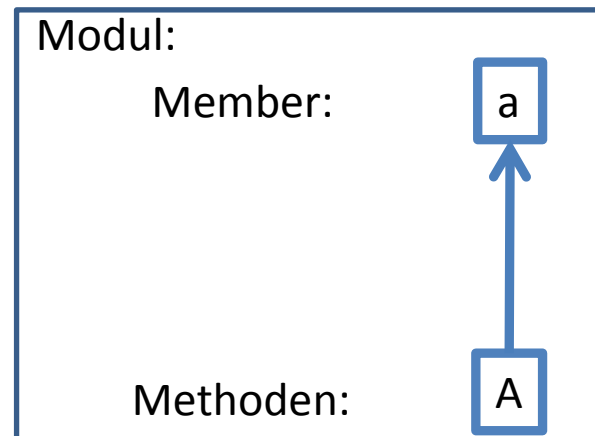
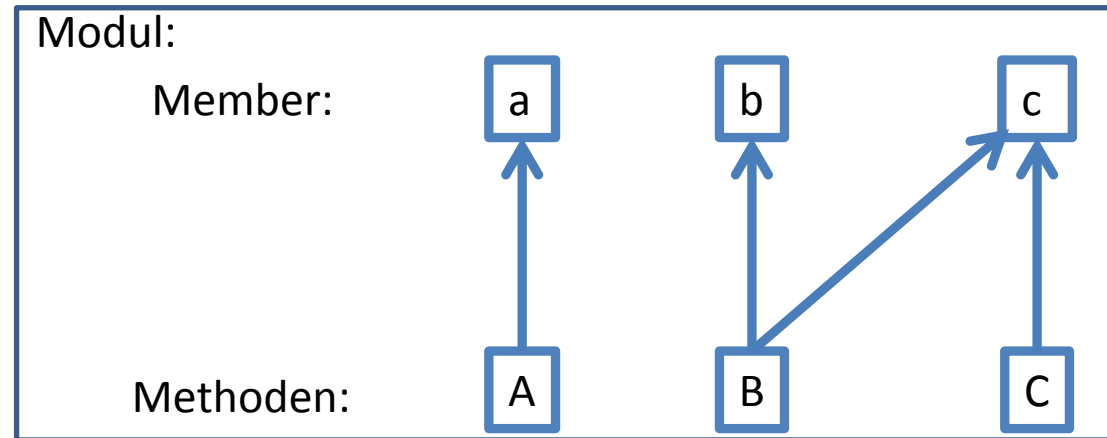
# Kohäsion

- Lack of Cohesion in Methods (LCOM)
- Kohäsion des Moduls messen
  - Methoden und benutzte Member
  - Anzahl Paare von Methoden mit gemeinsamen Members



# Kohäsion

- Grundidee
  - Insel-Methoden mit eigenen Attributen sind ein anderes Modul
- LCOM hoch
  - disjunkte Attributmengen
- LCOM niedrig
  - gleiche Attributmengen



# Aussagekraft von SLOC, CCN, Kohäsion

## I Beispiel + mögliche Interpretation

- SLOC stieg + CCN blieb gleich => neuer Code ca. gleich komplex
- SLOC stieg + CCN um das Doppelte gestiegen => ...
- SLOC reduziert + CCN reduziert => DRY reduziert, Refactoring
- CCN hoch, Kopplung hoch, Kohäsion niedrig => ...??
- CCN niedrig, Kopplung gering, Kohäsion hoch => ...??

# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools

# Anwendung von Metriken

- nur kombiniert aussagekräftig
- zeigen Trends in der Entwicklungsgeschichte an
- nie isoliert betrachten
- Interpretation nach Anwendungsfall
- Gefahr: „programmieren für die SW-Metrik“

# Inhaltsverzeichnis

## I Statische Codeanalyse

- Lines of Code
- McCabe Komplexität
- Kognitive Komplexität
- Kopplung von Modulen
- Kohäsion

## I Aussagekraft von Metriken

## I Tools



# Tools für die Übungen

- Hfcca.py : einfach zu installieren, sofort nutzbar; ersetzt durch
- Lizard
- CCCC: Tool zur Erzeugung von Statistiken, u.a. CCN; erzeugt HTML-Seite
- Compiler mit Flags für Dependency Generation
- JDepend / NDepend
- Sonarqube : viele Plugins, mächtig und vielseitig
- Doxygen
- LOC: sloccount, cloc (Linux)
- McCabe: pmccabe (Linux)
- ...