Empirische Studien

Wie kann Qualität (Wartbarkeit, Testbarkeit, . . .) gemessen werden?

- Bewertung der Qualität durch Entwickler
- Anwendung verschiedener Metriken
- Bestimmung der Korrelation
- Kombination der Metriken (orthogonal)

Empirische Studien

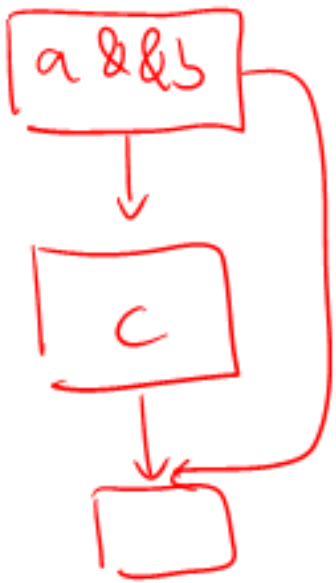
Maintainability Index (Coleman/Oman, 1994):

$$MI_1 = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot V(g') - 16.2 \cdot \ln(LOC)$$
$$MI_2 = MI_1 + 50 \cdot \sin \sqrt{2.46 \cdot perCM}$$

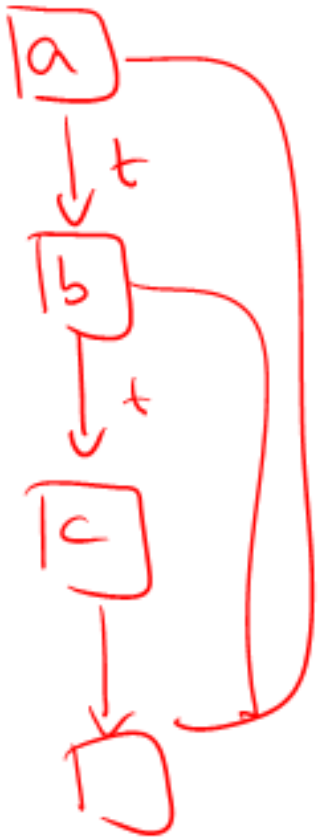
- V = average Halstead Volume per module
- $V(g')$ = average extended cyclomatic complexity per module
- LOC = average LOC per module
- $perCM$ = average percent of lines of comment per module

- MI_2 nur bei sinnvoller Kommentierung
- $MI < 65 \Rightarrow$ schlechte / $MI \geq 85 \Rightarrow$ gute Wartbarkeit

```
if (a&&b)
  c;
```




```
if(a&b)
```



Empirische Studien

Maintainability model (Muthanna u. a. 2000):

$$SMI = 125 - 3.989 \cdot FAN - 0.954 \cdot DF - 1.123 \cdot MC$$

- *FAN*: average number of external calls from the module
- *DF*: total number of outgoing and incoming data flow for the module
- *MC*: average McCabe for the module

Empirische Studien – OO

Wartbarkeit korreliert mit (Dagpinar und Jahnke 2003)

- TNOS - total number of statements
- NIM - number of instance methods
- FOUT - fan out, number of classes directly used

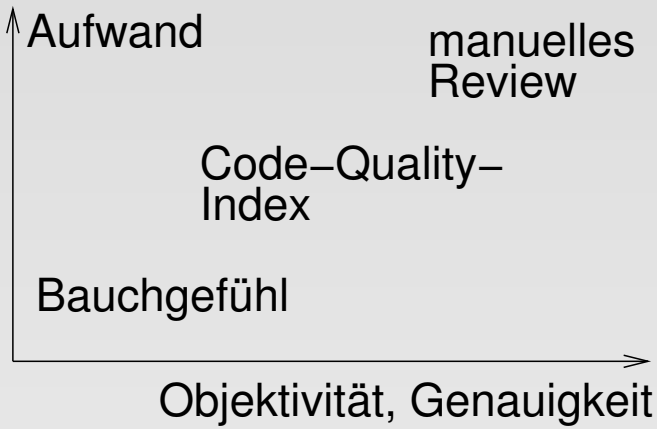
- **nicht** Vererbungshierarchie
- **nicht** Kohäsion
- **nicht** indirekte Kopplung
- **nicht** Kopplung über used-by Beziehungen

Empirische Studien – OO

Testbarkeit korreliert vor allem mit (Bruntink und van Deursen 2004):

- LOCC - lines of code per class
- FOUT - fan out, number of classes directly used
- RFC - response for class

Bewertungsschema von Simon u. a. (2006)



- statisch ermittelter, objektiver Code-Quality-Index
- 52 Qualitätsindikatoren (Typen von Bad Smells)
- Auswirkungen der Qualitätsindikatoren auf Qualitätseigenschaften:

  Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit, Austauschbarkeit, Zeitverhalten, Verbrauchsverhalten
- Häufigkeitsverteilung für mehr als 120 industrielle Systeme geschrieben in C++ und Java → „Industriestandard “
- Quality-Benchmark-Level

Quality-Benchmark-Level

1. Rudimentary: Code ist übersetz- und linkbar (nicht notwendigerweise ausführbar)
2. Basic:
   - wirtschaftliche Anpassbarkeit gegeben
   - Analysierbarkeit und Stabilität besonders gewichtig
3. Extended:
   - gute technische Qualität
   - zusätzlich hohe Gewichtung von Zeitverhalten und Verbrauchsverhalten
4. Advanced:
   - hervorragende technische Qualtität
   - zusätzlich hohe Gewichtung von Prüfbarkeit und Modifizierbarkeit
5. Complete:
   - perfekte technische Qualtität
   - zusätzlich hohe Gewichtung von Austauschbarkeit

Bei jedem Ebenenwechsel: geringere Toleranz für Verstöße

Weitere Anwendungen

- Hinweise auf Designfehler
- Klonerkennung (Mayrand)
- Mustersuche (Reduzierung des Suchraums)
- Suche nach Aspekten (FIN)
- ...

Tools

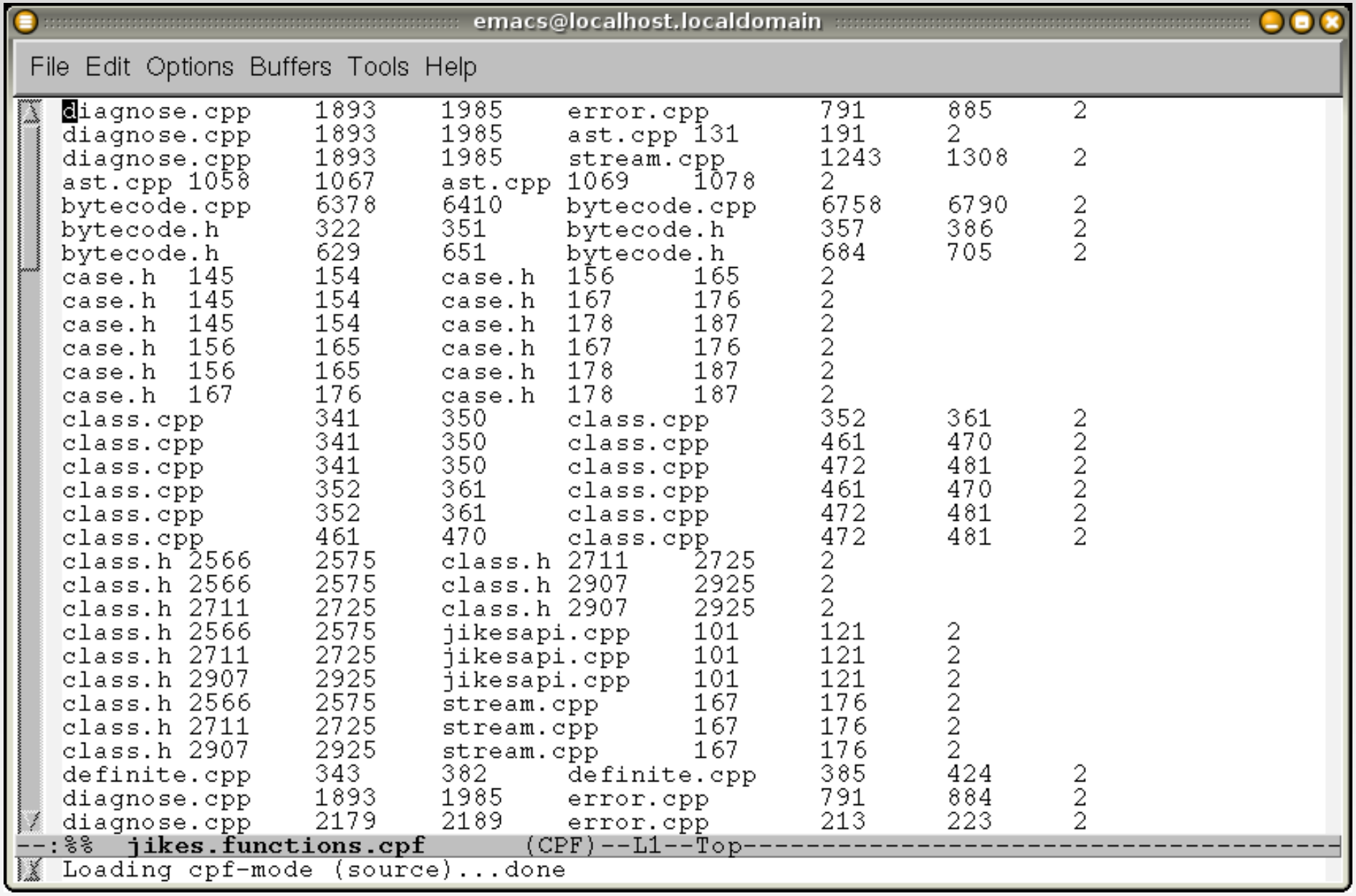
Freie Tools:

Tool	Sprachen	Metriken
clc (Perl)	C/C++	LOC, Comments, #Statements
cccc	C++, Java	LOC, McCabe, OO, ...
Metrics	C	LOC, Comments, Halstead, McCabe
MAS-C4	C	LOC, Comments, Halstead, McCabe, Nesting, Fan-In/-Out, Data Flow, ...
JMT	Java	OO-Metriken
Eclipse Plugins	Java	LOC, McCabe, OO, ...

Kommerzielle: z.B. McCabeQA, CMT, TAU/Logiscope, SDMetrics, CodeCheck, Krakatau Metrics, RSM, Together, ...

Techniken zur Klonerkennung

2. Klonerkennung
   - Hintergrund
   - Definitionen und Beispiele
   - Ansätze zur Klonerkennung
   - Verfahren nach Baker
   - Verfahren nach Baxter
   - Verfahren nach Mayrand et al.
   - Vergleich von Techniken zur Klonerkennung


emacs@localhost.localdomain
File Edit Options Buffers Tools Help
diagnose.cpp 1893 1985 error.cpp 791 885 2
diagnose.cpp 1893 1985 ast.cpp 131 191 2
diagnose.cpp 1893 1985 stream.cpp 1243 1308 2
ast.cpp 1058 1067 ast.cpp 1069 1078 2
bytecode.cpp 6378 6410 bytecode.cpp 6758 6790 2
bytecode.h 322 351 bytecode.h 357 386 2
bytecode.h 629 651 bytecode.h 684 705 2
case.h 145 154 case.h 156 165 2
case.h 145 154 case.h 167 176 2
case.h 145 154 case.h 178 187 2
case.h 156 165 case.h 167 176 2
case.h 156 165 case.h 178 187 2
case.h 167 176 case.h 178 187 2
class.cpp 341 350 class.cpp 352 361 2
class.cpp 341 350 class.cpp 461 470 2
class.cpp 341 350 class.cpp 472 481 2
class.cpp 352 361 class.cpp 461 470 2
class.cpp 352 361 class.cpp 472 481 2
class.cpp 461 470 class.cpp 472 481 2
class.h 2566 2575 class.h 2711 2725 2
class.h 2566 2575 class.h 2907 2925 2
class.h 2711 2725 class.h 2907 2925 2
class.h 2566 2575 jikesapi.cpp 101 121 2
class.h 2711 2725 jikesapi.cpp 101 121 2
class.h 2907 2925 jikesapi.cpp 101 121 2
class.h 2566 2575 stream.cpp 167 176 2
class.h 2711 2725 stream.cpp 167 176 2
class.h 2907 2925 stream.cpp 167 176 2
definite.cpp 343 382 definite.cpp 385 424 2
diagnose.cpp 1893 1985 error.cpp 791 884 2
diagnose.cpp 2179 2189 error.cpp 213 223 2
--:%% jikes.functions.cpf (CPF)--L1--Top
Loading cpf-mode (source)...done

error.cpp-2

File Edit Options Buffers Tools C++ Help

```
// This procedure uses a  quick sort algorithm to sort the ERRORS
// by the left_line_no and left_column_no fields.
//
void ParseError::SortMessages()
{
     int lower,
         upper,
         lostack[32],
         histack[32];

     int top,
         i,
         j;
     ParseErrorInfo pivot, temp;

     top = 0;
     lostack[top] = 0;
     histack[top] = errors.Length() - 1;

     while (top >= 0)
     {
         lower = lostack[top];
         upper = histack[top];
         top--;

         while (upper > lower)
         {
             /*********************************************************
             /* The array is most-likely almost sorted. Therefore,
             /* we use the middle element as the pivot element.
             *********************************************************/
             i = (lower + upper) / 2;
             pivot = errors[i];
             errors[i] = errors[lower];

             /*********************************************************
             /* Split the array section indicated by LOWER and UPPER
             /* using ARRAY(LOWER) as the pivot.
             *********************************************************/
             i = lower;
             for (j = lower + 1; j <= upper; j++)
                 if ((errors[j].left_token < pivot.left_token) ||
                 /*****************************************************
                 /* When two error messages start in the same location
                 /* and one is nested inside the other, the outer one
                 /* is placed first so that it can be printed last.
                 /* Recall that its right-span location is reached
                 /* after the inner one has been completely processed.
                 *****************************************************/
                     (errors[j].left_token == pivot.left_token &&
                      errors[j].right_token > pivot.right_token) ||
                 /*****************************************************
                 /* When two error messages are at the same location
                 /* span, check the NUM field to keep the sort stable.
                 /* When the location spans only a single symbol,
                 /* the one with the lowest "num" is placed first.
                 *****************************************************/
                     (errors[j].left_token  == pivot.left_token  &&
                      errors[j].right_token == pivot.right_token &&
                      pivot.left_token == pivot.right_token     &&
                      errors[j].num < pivot.num)
                 /*****************************************************/
```

```
--:%%  diagnose.cpp-1      (C++ Abbrev)--L1911--85%----------
```

```
//
// This procedure uses a  quick sort algorithm to sort the ER
// by the left_line_no and left_column_no fields.
//
void SemanticError::SortMessages()
{
     int lower,
         upper,
         lostack[32],
         histack[32];

     int top,
         i,
         j;
     ErrorInfo pivot,
               temp;

     top = 0;
     lostack[top] = 0;
     histack[top] = error.Length() - 1;

     while (top >= 0)
     {
         lower = lostack[top];
         upper = histack[top];
         top--;

         while (upper > lower)
         {
             //
             // The array is most-likely almost sorted. There
             //we use the middle element as the pivot elemen
             //
             i = (lower + upper) / 2;
             pivot = error[i];
             error[i] = error[lower];

             //
             // Split the array section indicated by LOWER an
             // using ARRAY(LOWER) as the pivot.
             //
             i = lower;
             for (j = lower + 1; j <= upper; j++)
                 if ((error[j].left_token < pivot.left_token)
                 //
                 // When two error messages start in the same
                 // and one is nested inside the other, the o
                 // is placed first so that it can be printed
                 // Recall that its right-span location is re
                 // after the inner one has been completely p
                 //
                     (error[j].left_token == pivot.left_token
                      error[j].right_token > pivot.right_toke
                 //
                 // When two error messages are at the same l
                 // span, check the NUM field to keep the sor
                 // When the location spans only a single sym
                 // the one with the lowest "num" is placed f
                 //
                     (error[j].left_token  == pivot.left_toke
                      error[j].right_token == pivot.right_tok
                      pivot.left_token == pivot.right_token
```

```
--:%%  error.cpp-2        (C++ Abbrev)--L818--27%----------
```

ast.cpp-2

File Edit Options Buffers Tools C++ Help

```
//
// This procedure uses a  quick sort algorithm to sort the ERRORS
// by the left_line_no and left_column_no fields.
//
void ParseError::SortMessages()
{
    int lower,
        upper,
        lostack[32],
        histack[32];

    int top,
        i,
        j;
    ParseErrorInfo pivot, temp;

    top = 0;
    lostack[top] = 0;
    histack[top] = errors.Length() - 1;

    while (top >= 0)
    {
        lower = lostack[top];
        upper = histack[top];
        top--;

        while (upper > lower)
        {
            /*****************************************************
            /* The array is most-likely almost sorted. Therefore,
            /* we use the middle element as the pivot element.
            /*****************************************************
            i = (lower + upper) / 2;
            pivot = errors[i];
            errors[i] = errors[lower];

            /*****************************************************
            /* Split the array section indicated by LOWER and UPP
            /* using ARRAY(LOWER) as the pivot.
            /*****************************************************
            i = lower;
            for (j = lower + 1; j <= upper; j++)
                if ((errors[j].left_token < pivot.left_token) ||
                /*************************************************
                /* When two error messages start in the same loca
                /* and one is nested inside the other, the outer
                /* is placed first so that it can be printed last
                /* Recall that its right-span location is reached
                /* after the inner one has been completely proces
                /*************************************************
                    (errors[j].left_token == pivot.left_token &&
                     errors[j].right_token > pivot.right_token) |
                /*************************************************
                /* When two error messages are at the same locati
                /* span, check the NUM field to keep the sort sta
                /* When the location spans only a single symbol,
                /* the one with the lowest "num" is placed first.
                /*************************************************
                    (errors[j].left_token  == pivot.left_token  &
                     errors[j].right_token == pivot.right_token &
                     pivot.left_token == pivot.right_token     &&
                     errors[j].num < pivot.num)
```

--:%% diagnose.cpp-1 (C++ Abbrev)--L1920--85%----------

```
// This procedure uses a quick sort algorithm to sort the cases i
// statement. Element 0 is not sorted, because it is the default
// may be NULL).
//
void AstSwitchStatement::SortCases()
{
    int lower;
    int upper;
    int lostack[32];
    int histack[32];
    int top = 0;
    int i;
    int j;
    CaseElement pivot;
    CaseElement temp;

    lostack[top] = 1;
    histack[top] = num_cases;

    while (top >= 0)
    {
        lower = lostack[top];
        upper = histack[top];
        top--;

        while (upper > lower)
        {
            //
            // The array is most-likely almost sorted. Therefore,
            // we use the middle element as the pivot element.
            //
            i = (lower + upper) >> 1;
            pivot = *cases[i];
            *cases[i] = *cases[lower];

            //
            // Split the array section indicated by LOWER and UPP
            // using ARRAY(LOWER) as the pivot.
            //
            i = lower;
            for (j = lower + 1; j <= upper; j++)
                if (*cases[j] < pivot)
                {
                    temp = *cases[++i];
                    *cases[i] = *cases[j];
                    *cases[j] = temp;
                }
            *cases[lower] = *cases[i];
            *cases[i] = pivot;

            top++;
            if ((i - lower) < (upper - i))
            {
                lostack[top] = i + 1;
                histack[top] = upper;
                upper = i - 1;
            }
            else
            {
                histack[top] = i - 1;
                lostack[top] = lower;
                lower = i + 1;
```

--:%% ast.cpp-2 (C++ Abbrev)--L158-- 5%----------

stream.cpp-2

File Edit Options Buffers Tools C++ Help

```
//
// This procedure uses a  quick sort algorithm to sort the ERRORS
// by the left_line_no and left_column_no fields.
//
void ParseError::SortMessages()
{
    int lower,
        upper,
        lostack[32],
        histack[32];

    int top,
        i,
        j;
    ParseErrorInfo pivot, temp;

    top = 0;
    lostack[top] = 0;
    histack[top] = errors.Length() - 1;

    while (top >= 0)
    {
        lower = lostack[top];
        upper = histack[top];
        top--;

        while (upper > lower)
        {
            /*****************************************************
            /* The array is most-likely almost sorted. Therefore,
            /* we use the middle element as the pivot element.
            /*****************************************************
            i = (lower + upper) / 2;
            pivot = errors[i];
            errors[i] = errors[lower];

            /*****************************************************
            /* Split the array section indicated by LOWER and UPP
            /* using ARRAY(LOWER) as the pivot.
            /*****************************************************
            i = lower;
            for (j = lower + 1; j <= upper; j++)
                if ((errors[j].left_token < pivot.left_token) ||
                /*************************************************
                /* When two error messages start in the same loca
                /* and one is nested inside the other, the outer
                /* is placed first so that it can be printed last
                /* Recall that its right-span location is reached
                /* after the inner one has been completely proces
                /*************************************************
                    (errors[j].left_token == pivot.left_token &&
                     errors[j].right_token > pivot.right_token) |
                /*************************************************
                /* When two error messages are at the same locati
                /* span, check the NUM field to keep the sort sta
                /* When the location spans only a single symbol,
                /* the one with the lowest "num" is placed first.
                /*************************************************
                    (errors[j].left_token  == pivot.left_token  &
                     errors[j].right_token == pivot.right_token &
                     pivot.left_token == pivot.right_token     &&
                     errors[j].num < pivot.num)
```

--:%% diagnose.cpp-1 (C++ Abbrev)--L1920--85%----------

```
//
// This procedure uses a  quick sort algorithm to sort the stream
// by their locations.
//
void LexStream::SortMessages()
{
    int lower,
        upper,
        lostack[32],
        histack[32];

    int top,
        i,
        j;
    StreamError pivot,
                temp;

    top = 0;
    lostack[top] = 0;
    histack[top] = bad_tokens.Length() - 1;

    while (top >= 0)
    {
        lower = lostack[top];
        upper = histack[top];
        top--;

        while (upper > lower)
        {
            //
            // The array is most-likely almost sorted. Therefore
            // we use the middle element as the pivot element.
            //
            i = (lower + upper) / 2;
            pivot = bad_tokens[i];
            bad_tokens[i] = bad_tokens[lower];

            //
            // Split the array section indicated by LOWER and UP
            // using ARRAY(LOWER) as the pivot.
            //
            i = lower;
            for (j = lower + 1; j <= upper; j++)
            {
                if (bad_tokens[j].start_location < pivot.start_l
                {
                    temp = bad_tokens[++i];
                    bad_tokens[i] = bad_tokens[j];
                    bad_tokens[j] = temp;
                }
            }
            bad_tokens[lower] = bad_tokens[i];
            bad_tokens[i] = pivot;

            top++;
            if ((i - lower) < (upper - i))
            {
                lostack[top] = i + 1;
                histack[top] = upper;
                upper = i - 1;
            }
            else
```

--:%% stream.cpp-2 (C++ Abbrev)--L1270--92%----------

Klonerkennung

- Lernziele
  - Varianten der Klonerkennung (Erkennung duplizierten Codes)
  - Bezug zu Abstraktionsebenen von Programmdarstellungen
- Kontext
  - Beseitigung von Redundanz auf Codierungsebene
  - erleichtert nachfolgende Reengineering-Aktivitäten