

# Rajapinnat: Sopimussuunnittelu

# Hyvä rajapinta?

- *Täydellinen*
- Kaunis
- Söpö



Kuva: clement127 (CC BY-NC-ND 2.0)

# Rajapinnan tehtävä?



Kuva: Clement127 (CC BY-NC-ND 2.0)

# Mistä rajapinnat tulevat?

Ohjelman suunnittelun päävaiheet: Komponenttien

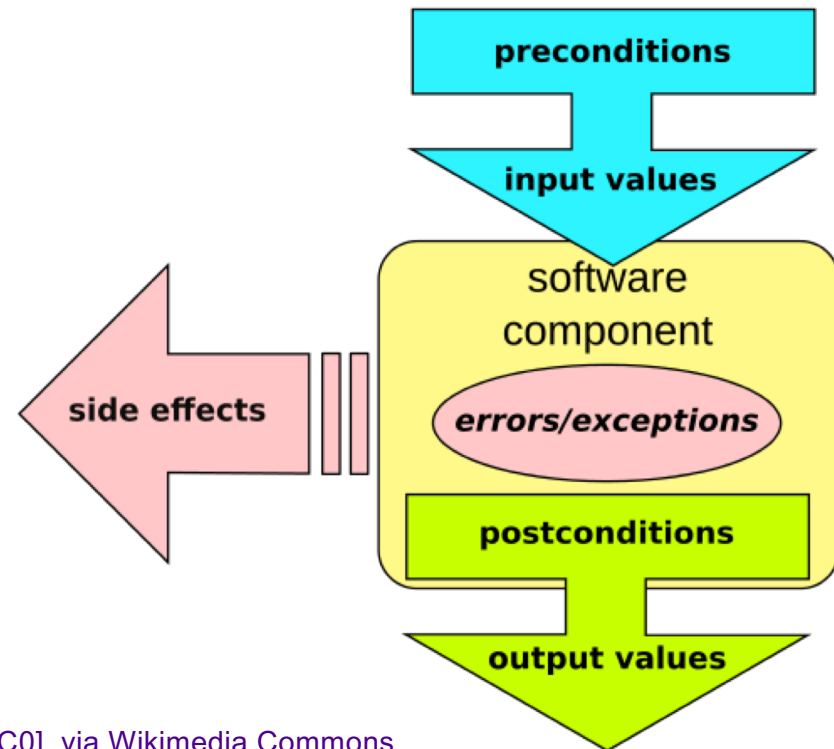
- tunnistaminen
- vastuualueiden määrittely
- välisten suhteiden määrittely
- **Rajapintojen määrittely**

# Rajapintasuunnittelu

- Miten rajapintaa on lupa **käyttää**?
- Mitä rajapinnan toiminnot **lupaavat** tehdä?
- Millaisia **virheitä** toiminnoissa voi tulla?
- Miten rajapintaa tulisi **testata**?

# Sopimussuunnittelu (Design by Contract)

- Selkeä metafora suunnitteluprosessista



Kuva: Fabuio (Own work) [CC0], via Wikimedia Commons

# Sopimussuunnittelu

Tapa määritellä rajapinta **sopimuksena**

- Asiakas (kutsuja) ja toimittaja (toteuttaja)
- Velvoite ja hyöty

Rajapinnan ja käyttäjän välinen sopimus

- Kutsujan velvoite: miten rajapintaa **saa** kutsua? (esiehto)
- Toteuttajan velvoite: mitä rajapinta **lupaa** tehdä? (jälkiehto)
- virhetapausten määrittely

`{P} o.palvelu() {Q}`

# Esiehto (precondition)

- Oltava voimassa **ennen** palvelua
- Toteutuminen **kutsujan** vastuulla
- “Milloin/miten palvelua saa kutsua?” tai “mitä se odottaa?”
- Esim.  $\{a < 10 \wedge b < 20\}$



# Jälkiehto (postcondition)

- Oltava voimassa palvelun **jälkeen**
- Toteutuminen **toteuttajan** vastuulla
- “Mitä palvelu lupaa tehdä?” tai “mitä se takaa?”
- Jos jälkiehto ei toteudu, päädytään poikkeustilanteeseen
- Esim.  $\{j < 30 \wedge a < 10 \wedge b < 20\}$

# Velvoitteet (vastuut)

## Kutsuja

- Pitää huolta, että esiehto toteutuu
- Esiehtoja voidaan tarkastaa testausvaiheessa, ei enää valmiissa ohjelmistossa

## Toteuttaja

- Pitää huolta, että suorituksen jälkeen jälkiehto voimassa
- Ei enää tarkastuksia kutsujan velvoitteista
- Jos palvelua ei voida toteuttaa (jälkiehto ei täyty) ⇒ virhetilanne, ilmaistava

# (Luokka)invariantti

Pysyväisväittäjä

- Testaa, että olio on käyttökelpoinen tai “järjissään”
- Oltava voimassa kutsujen välissä

$\{\text{LUOKKAINVARIANTTI} \wedge P\} o.\text{palvelu}() \{\text{LUOKKAINVARIANTTI} \wedge Q\}$

- Hyödyllinen toteuttajalle, ei kutsujalle

## Esimerkki

```
class Paivays
{
public
    asetaPaiva( int paiva );
private
    int p_;
    int kk_;
    int v_;
};
```

- Invariantti?
- Esiehto?
- Jälkiehto?

# C++20: Contracts

- C++20 mahdollistaa esi- ja jälkiehtojen asettamisen sekä invariantin tarkistamisen koodissa

```
double sqrt(double x) [[expects: x >= 0]];  
void sort(vector<emp>& v) [[ensures audit:  
is_sorted(v)]];
```

# C++20: Contracts

```
int push(queue& q, int val)
    [[ expects: !q.full() ]]
    [[ ensures: !q.empty() ]]
    {
        ...
        [[assert: q.is_ok() ]]
        ...
    }
```

# Ehtojen määrittely

## 25.4.3.4 binary\_search

[binary.search]

```
template<class ForwardIterator, class T>
    bool binary_search(ForwardIterator first, ForwardIterator last,
                      const T& value);
```

```
template<class ForwardIterator, class T, class Compare>
    bool binary_search(ForwardIterator first, ForwardIterator last,
                      const T& value, Compare comp);
```

- 1 *Requires:* The elements  $e$  of  $[first, last)$  are partitioned with respect to the expressions  $e < value$  and  $!(value < e)$  or  $comp(e, value)$  and  $!comp(value, e)$ . Also, for all elements  $e$  of  $[first, last)$ ,  $e < value$  implies  $!(value < e)$  or  $comp(e, value)$  implies  $!comp(value, e)$ .
- 2 *Returns:* true if there is an iterator  $i$  in the range  $[first, last)$  that satisfies the corresponding conditions:  $!(*i < value) \ \&\& \ !(value < *i)$  or  $comp(*i, value) == false \ \&\& \ comp(value, *i) == false$ .
- 3 *Complexity:* At most  $\log_2(last - first) + \mathcal{O}(1)$  comparisons.

# Ehtojen määrittely

## 25.4.1.1 sort

[sort]

```
template<class RandomAccessIterator>
    void sort(RandomAccessIterator first, RandomAccessIterator last);
```

```
template<class RandomAccessIterator, class Compare>
    void sort(RandomAccessIterator first, RandomAccessIterator last,
              Compare comp);
```

- 1 *Effects:* Sorts the elements in the range `[first,last)`.
- 2 *Requires:* `RandomAccessIterator` shall satisfy the requirements of `ValueSwappable` (17.6.3.2). The type of `*first` shall satisfy the requirements of `MoveConstructible` (Table 20) and of `MoveAssignable` (Table 22).
- 3 *Complexity:*  $\mathcal{O}(N \log(N))$  (where  $N == \text{last} - \text{first}$ ) comparisons.



# Sopimussuunnittelu: dokumentointi

# Dokumentointi

```
mapped_type& operator[] (const key_type& k);  
mapped_type& operator[] (key_type&& k);
```

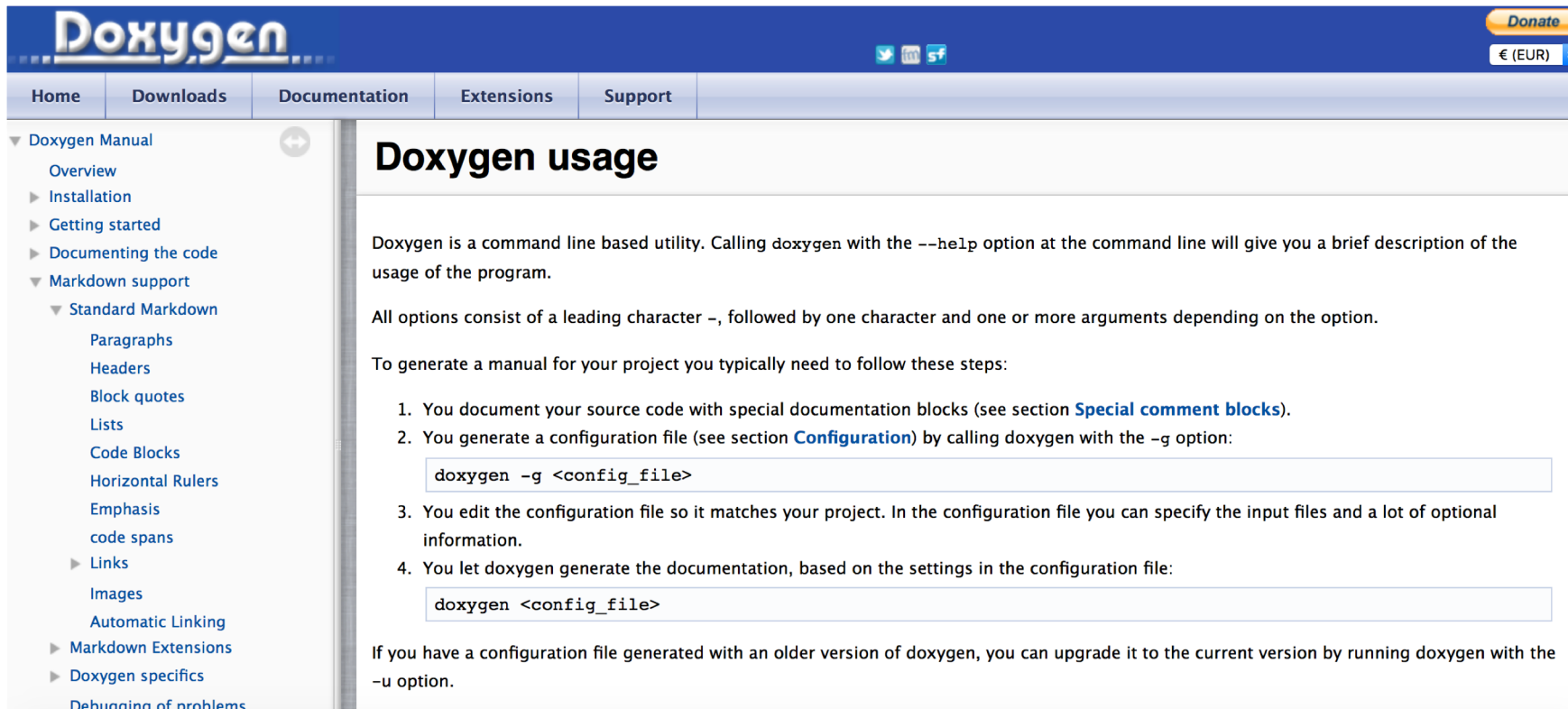
- 1 *Requires:* `mapped_type` shall be `DefaultInsertable` into `*this`. For the first operator, `key_type` shall be `CopyInsertable` into `*this`. For the second operator, `key_type` shall be `MoveConstructible`.
- 2 *Effects:* If the `unordered_map` does not already contain an element whose key is equivalent to `k`, the first operator inserts the value `value_type(k, mapped_type())` and the second operator inserts the value `value_type(std::move(k), mapped_type())`.
- 3 *Returns:* A reference to `x.second`, where `x` is the (unique) element whose key is equivalent to `k`.
- 4 *Complexity:* Average case  $\mathcal{O}(1)$ , worst case  $\mathcal{O}(\text{size}())$ .

Mitä tiedät operaation `[]` käyttäytymisestä dokumentaation perusteella?

# Toteutuksen piilotus

- Esi- ja jälkiehdot *dokumentoidaan* käyttäjän ymmärtämässä muodossa
- Jos luokka **testaa** ehdot/invariantin, testi tietysti kirjoitetaan sisäisen toteutuksen perusteella

# Dokumentointi: Doxygen



The screenshot shows the Doxygen website's documentation page. The top navigation bar includes 'Home', 'Downloads', 'Documentation', 'Extensions', and 'Support'. A sidebar on the left lists the 'Doxygen Manual' with sub-sections like 'Overview', 'Installation', 'Getting started', 'Documenting the code', and 'Markdown support'. The main content area is titled 'Doxygen usage' and contains the following text:

Doxygen is a command line based utility. Calling `doxygen` with the `--help` option at the command line will give you a brief description of the usage of the program.

All options consist of a leading character `-`, followed by one character and one or more arguments depending on the option.

To generate a manual for your project you typically need to follow these steps:

1. You document your source code with special documentation blocks (see section [Special comment blocks](#)).
2. You generate a configuration file (see section [Configuration](#)) by calling `doxygen` with the `-g` option:

```
doxygen -g <config_file>
```
3. You edit the configuration file so it matches your project. In the configuration file you can specify the input files and a lot of optional information.
4. You let `doxygen` generate the documentation, based on the settings in the configuration file:

```
doxygen <config_file>
```

If you have a configuration file generated with an older version of `doxygen`, you can upgrade it to the current version by running `doxygen` with the `-u` option.

# Dokumentointi: Doxygen

## KDE API Reference

KDE API Reference

### Navigation

- [Main Page](#)
- [Old KDE4 Versions](#)


### Related

- [API Doc Tutorial](#)
- [KDE TechBase](#)
- [KDE CMake Modules](#)
- [Extra CMake Modules](#)

### Search


## API Reference Index

The reference guides for the KDE APIs -- for KDE2 all the way to the current development version -- are collected here. We assume you are already familiar with the excellent [Qt4](#) documentation. [TechBase](#) is the right place to start looking for general development information for KDE. There are only reference guides here.

To obtain a gzip compressed tar file containing the documentation, click on the  images, which are immediately adjacent to many of the listed items.

To obtain a version of the documentation for use in [Digia Qt Assistant](#), click on the "[qch]" links, which are immediately adjacent to some of the listed items. (In Qt assistant, go into Edit->Preferences->Documentation and [Add] the .qch file.)

Man pages are also provided for some modules. Click on the "[man]" links, also immediately adjacent to some of the listed items to download a bzip2 compressed tar file containing the man pages for the corresponding module. (Uncompress and untar these files into a standard MANPATH directory.)

Frameworks	Others
 <a href="#">frameworks5 [qch][man]</a>	<ul style="list-style-type: none"><li><a href="#">Other KDE Software</a></li><li><a href="#">KDE4 Versions</a></li><li><a href="#">KDE3 and older versions</a></li></ul>

# Dokumentointi: Doxygen

```
/**  
... text ... */  
tai  
/*!  
... text ...  
*/
```

# Dokumentointi: Doxygen

```
\pre { esiehto }  
\post { jälkiehto }  
\throw <poikkeusolio> { poikkeus }  
\invariant { invariantti }
```

# Sopimussuunnittelun käyttö

- Luentotason esimerkeissä usein varsin suoraviivaista.
- Käytännössä:
  - Skaalautuu huonosti
  - Matemaattisen tarkka määrittely haastavaa (ja usein liiallista)
  - Periytymisen haasteet

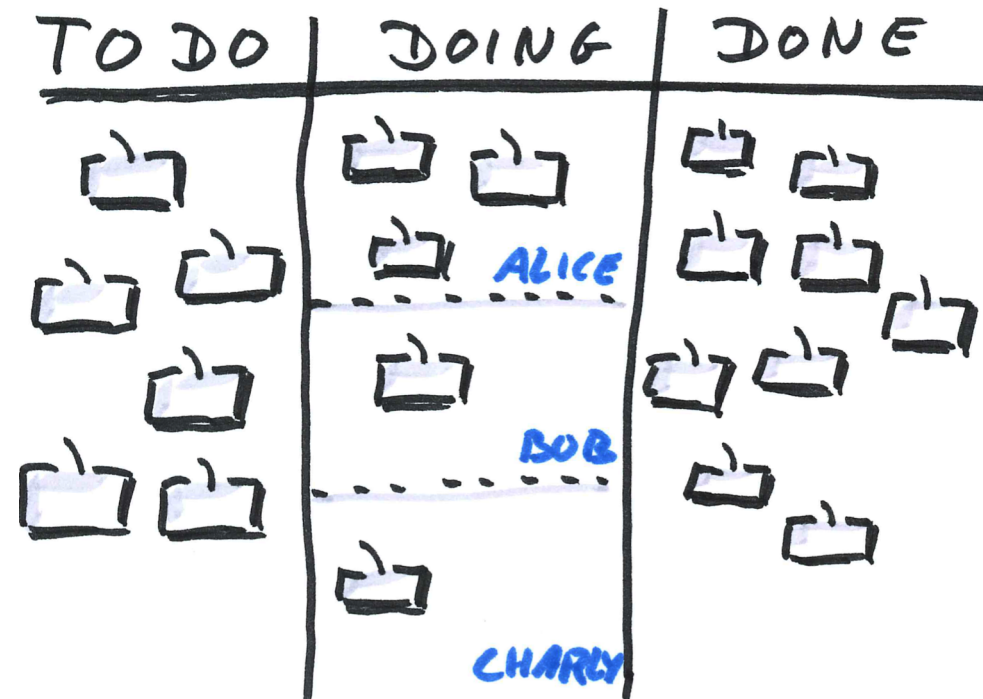


# Muita sopimuksia ja käytäntöjä

# Työnjako

Yksikertaiset menetelmät parhaita

- Kanban-taulu
- trello.com



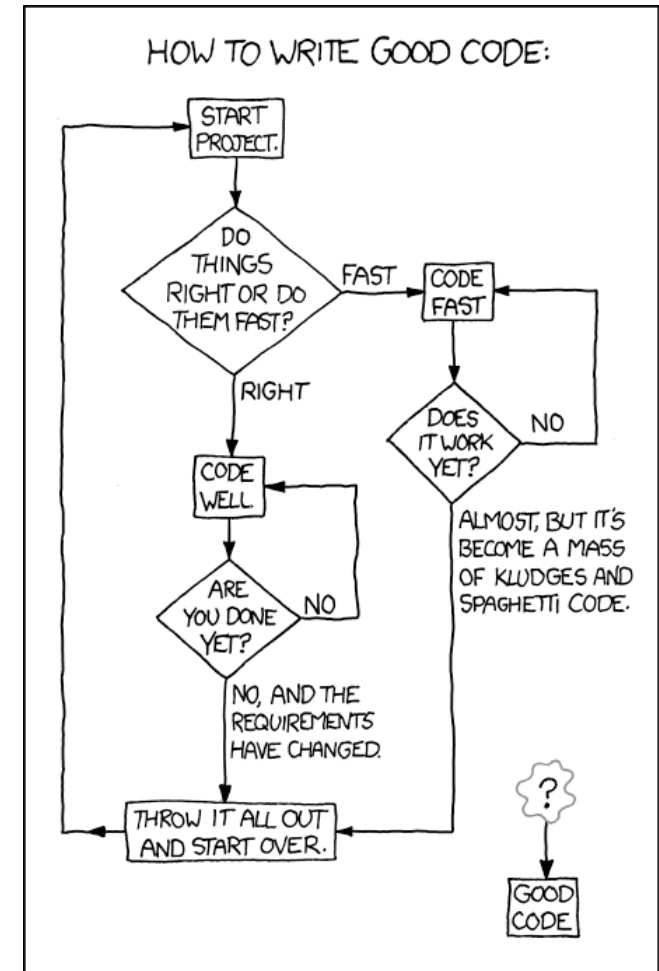
Kuva: Oliver Tacke (CC BY 2.0)

# Koodikonventiot

Parantavat koodin luettavuutta ja laatua

Sovittavia asioita

- Kommentointi (ml. Doxygen)
- Sisennys
- Rivin pituus
- Nimeäminen
- Koodauskäytännöt ja -periaatteet, peukkusäännöt
- Tyyliasiat



# Pariohjelmointi

- Lähtöisin eXtreme Programmingista
- Kaksi ohjelmoijaa: Kontrolloija (controller) ja tarkkailija (observer)
  - Parempi laatu: vähemmän virheitä
  - Tiimityö ja kommunikaatio
  - Oppiminen

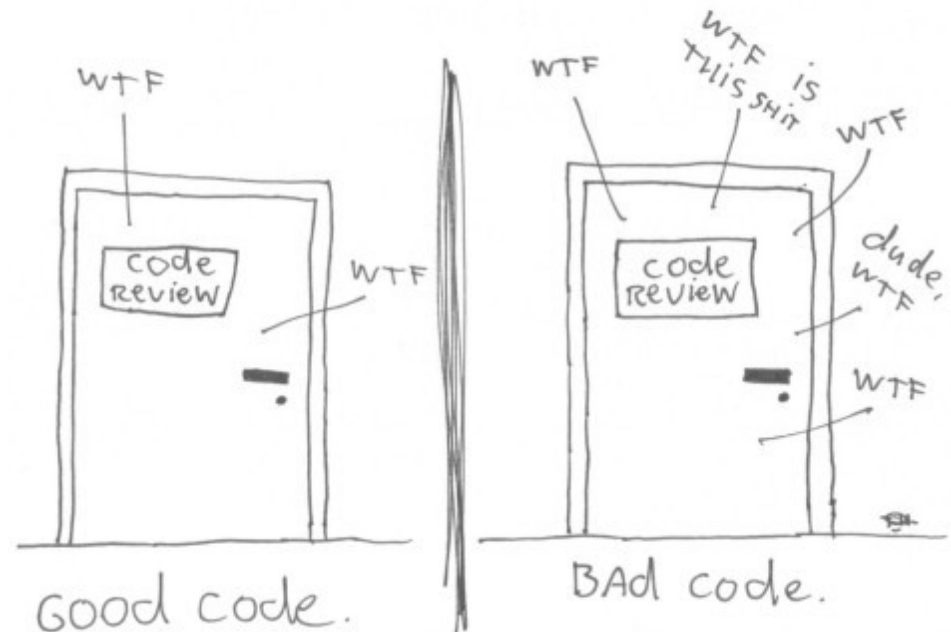
# Koodin katselmointi

Koodia luetaan ja mietitään:

- Tekeekö se sitä, mitä pitää,
- Vastaako se koodikonventioihin
- Sisältääkö se virheitä

Kuva: osnews.com/Thom Holwerda

The ONLY VALID MEASUREMENT  
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift/OSNews/Thom Holwerda · <http://www.osnews.com/comics>

# Sopimussuunnittelu: testaus

# Sopimussuunnittelu ja testaus

- Täydentää testausprosessia: yksikkötestaus, integraatiotestaus, järjestelmätestaus
- Testataan myös esiehtoja
- Integraatiotestaus ilmaiseksi
- Tukee debuggausta: sopimuksen rikkoutumiskohdat auttavat paljastamaan myös virhekohtat

## Ehtojen testaus: C++

C++:n `assert`-makro testaa ehdon ja kaataa ohjelman, jos ei päde

```
#include <cassert>
void f( int i )
{ assert( i >= 0 );}
```

Symboli `NDEBUG` määritelty  $\Rightarrow$  `assert` ei tee mitään



```
inline void JarjestettyTaulukko::Invariantti()  
{  
#ifndef NDEBUG  
    // Invariantti: alkioit ovat aina suuruusjärjestyksessä siten,  
    että  
    // indeksissä 1 on pienin alkio ja indeksissä KOKO suurin  
    for( int i = 1; i < KOKO; i++ )  
    {  
        if( alkio[ i ] > alkio[ i+1 ] )  
            throw JarjestettyTaulukko::InvarianttiRikottu();  
    }  
#endif  
}  
  
void JarjestettyTaulukko::etsiJaMuutaAlkio( Alkio const& etsittava,  
Alkio const& korvaava )  
{  
    Invariantti();  
    // Jäsenfunktion toteutus  
    Invariantti();  
}
```

## Invariantti

# Ehtojen testaus: Qt

Qt:lla omia funktioita/makroja

- `#include <QtGlobal>`
- `Q_ASSERT`-makrot
- Pois päältä `QT_NO_DEBUG`

```
void Q_ASSERT(bool test)  
void Q_ASSERT(bool test, const char* where, const char* what)
```

## Ehtojen testaus: Qt

```
int divide(int a, int b)
{
    Q_ASSERT(b != 0);
    return a / b;
}
```

⇒ ASSERT: "b == 0" in file div.cpp, line 7

## Ehtojen testaus: Qt

Vastaavasti:

```
    Q_ASSERT_X(b != 0, "divide", "division  
by zero");  
    return a / b;
```

⇒ ASSERT failure in divide: "division by zero", file div.cpp, line 7

# Testaaminen

"Beware of bugs in the above code, I have only proved it correct, not tried it."

Donald Knuth

