Tampereen yliopisto
Tampere University

Rajapinnat: Sopimussuunnittelu

Tampereen yliopisto
Tampere University

Hyvä rajapinta?

- ***Täydellinen***
- Kaunis
- Söpö

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

Rajapinnan tehtävä?

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

Tampereen yliopisto
Tampere University

Mistä rajapinnat tulevat?

Ohjelman suunnittelun päävaiheet: Komponenttien

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

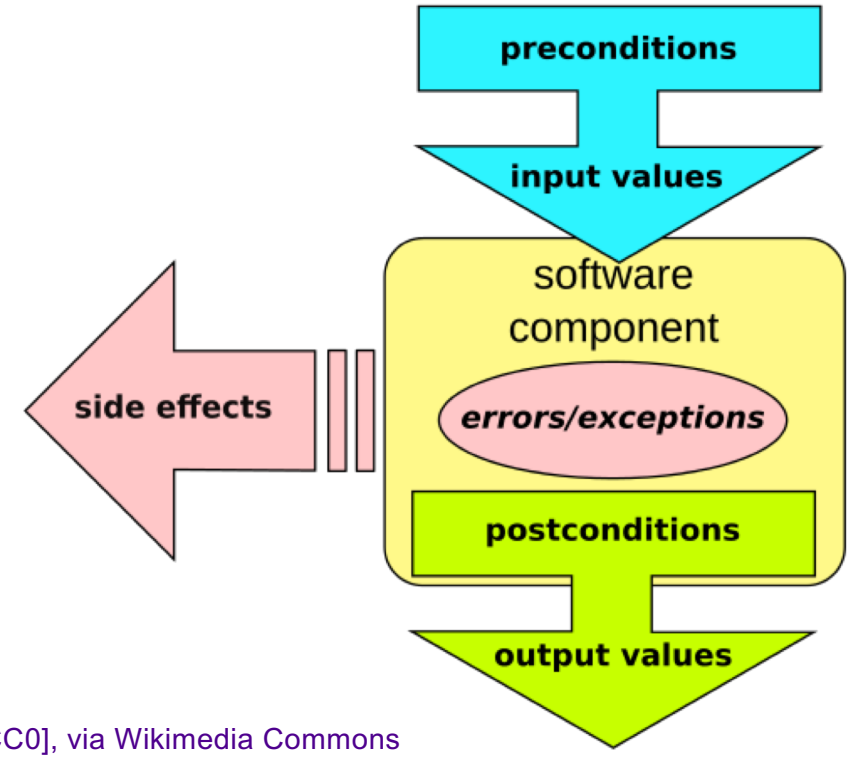
Tampereen yliopisto
Tampere University

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 suunnitteluproses-sista



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

Tampereen yliopisto
Tampere University

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. $\{j<30 \land a<10 \land b<20\}$

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

Pysyväisväittämä

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

```
{LUOKKAINVARIANTTI ∧ P} o.palvelu() {LUOKKAINVARIANTTI ∧ 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)]];
```

Tampereen yliopisto
Tampere University

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 `log2(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 `== last - first`) comparisons.

Tampereen yliopisto
Tampere University

Sopimussuunnittelu: dokumentointi

Tampereen yliopisto
Tampere University

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 $O(1)$, worst case $O($`size()`$)$.

Mitä tiedät operaation [] käyttäytymisestä dokumentaation perusteella?

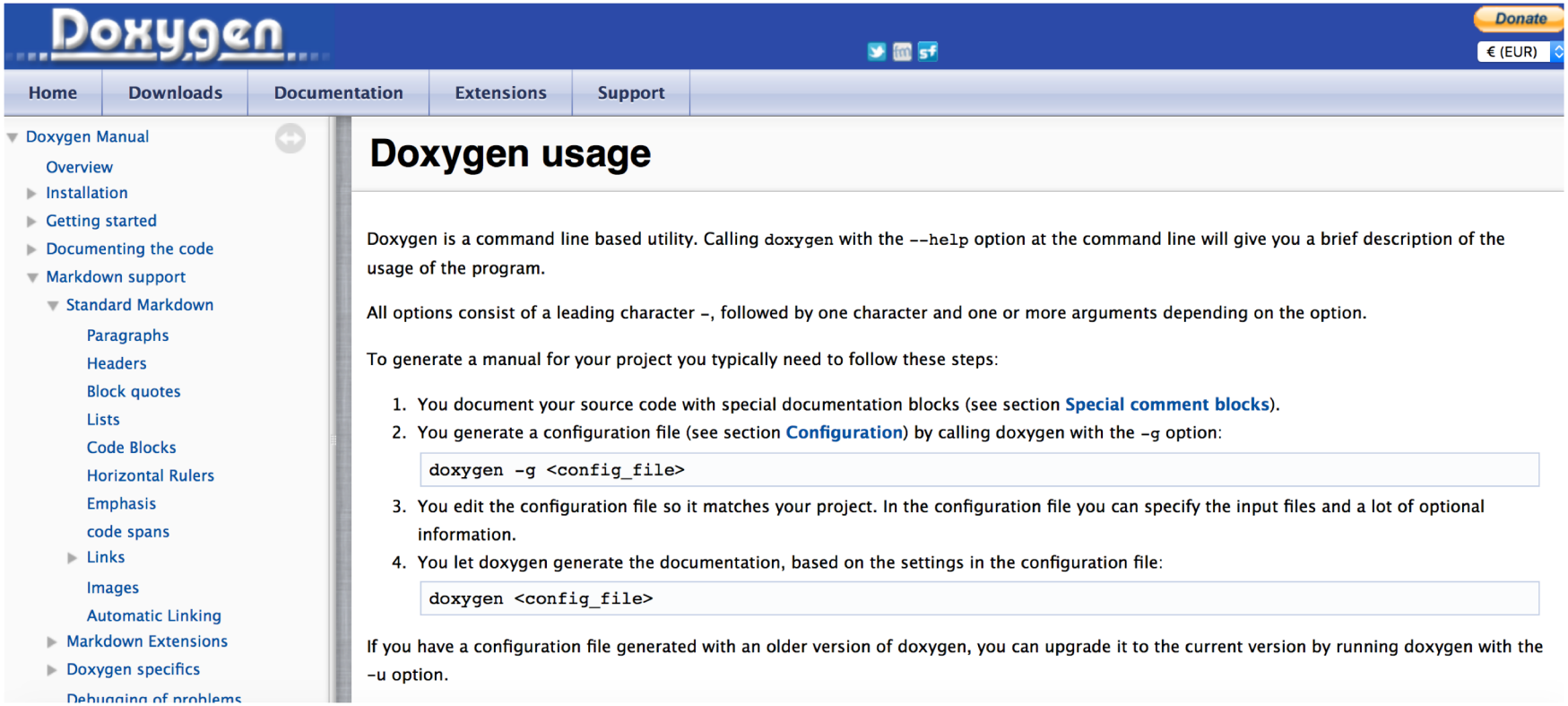
Tampereen yliopisto
Tampere University

Toteutuksen piilotus

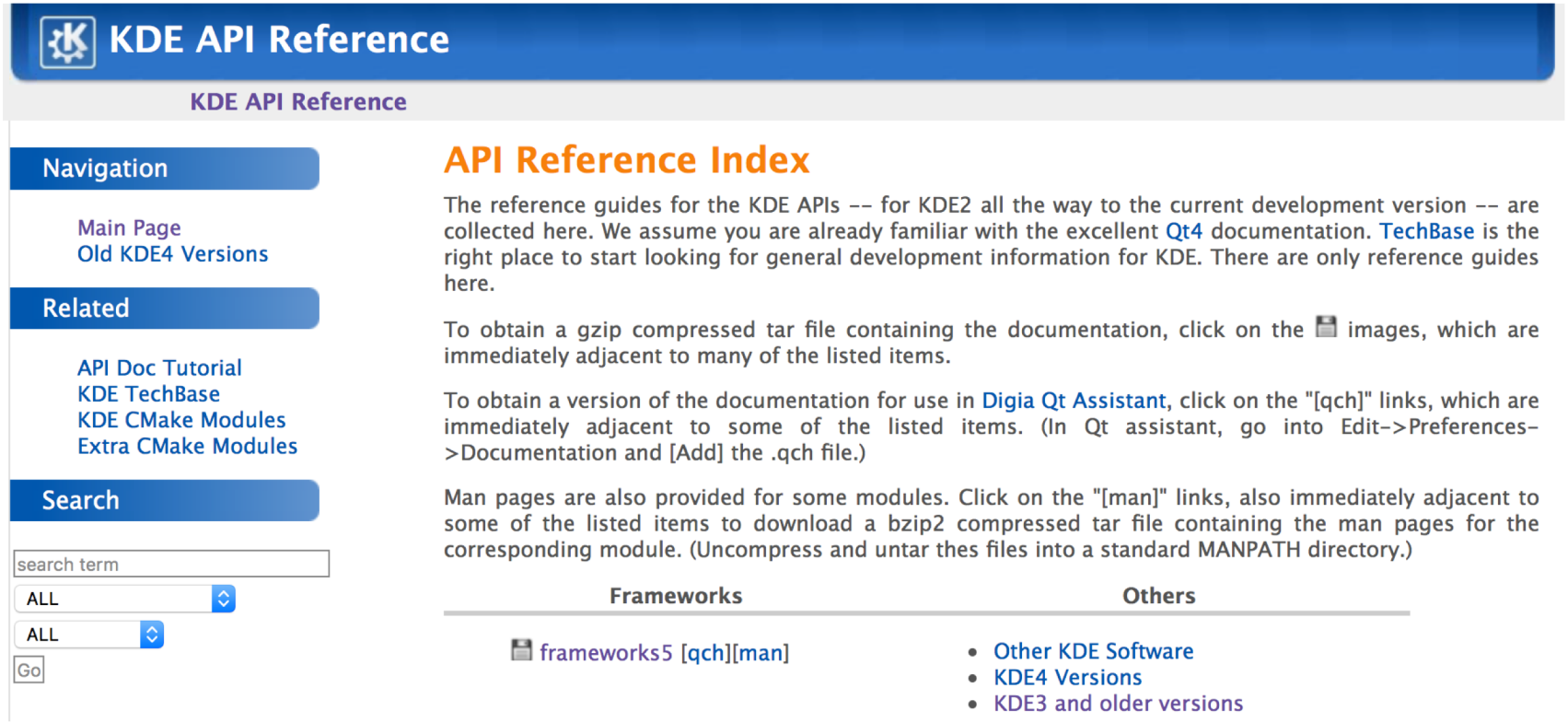
- Esi- ja jälkiehdot *dokumentoidaan* käyttäjän ymmärtämässä muodossa
- Jos luokka **testaa** ehdot/invariantin, testi tietysti kirjoitetaan sisäisen toteutuksen perusteella

Tampereen yliopisto
Tampere University

Dokumentointi: Doxygen



Dokumentointi: Doxygen



Tampereen yliopisto
Tampere University

Dokumentointi: Doxygen

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

Tampereen yliopisto
Tampere University

Dokumentointi: Doxygen

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

Tampereen yliopisto
Tampere University

Sopimussuunnittelun käyttö

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

Tampereen yliopisto
Tampere University

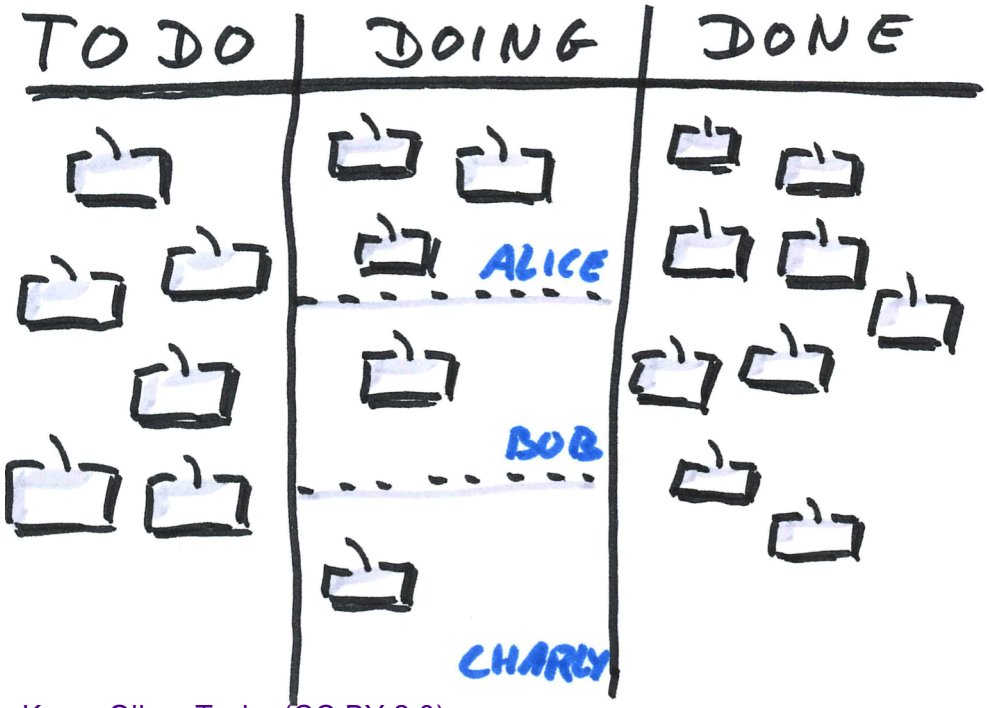
Muita sopimuksia ja käytäntöjä

Tampereen yliopisto
Tampere University

Työnjako

Yksikertaiset menetelmät parhaita

- Kanban-taulu
- `trello.com`



Kuva: Oliver Tacke (CC BY 2.0)

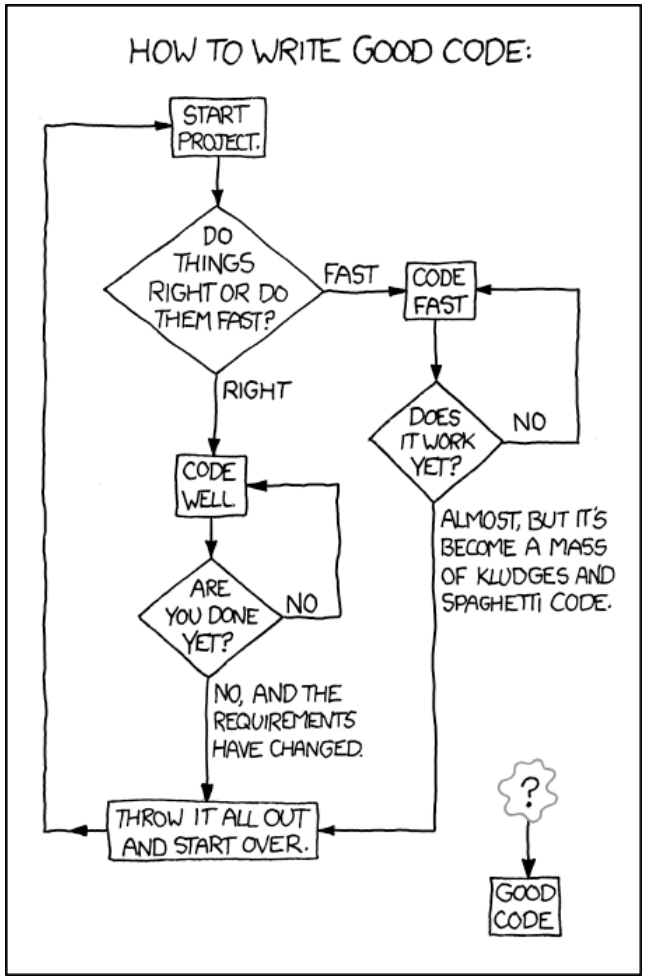
Tampereen yliopisto
Tampere University

Koodikonventiot

Parantavat koodin luettavuutta ja laatua

Sovittavia asioita

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



Tampereen yliopisto
Tampere University

Pariohjelmointi

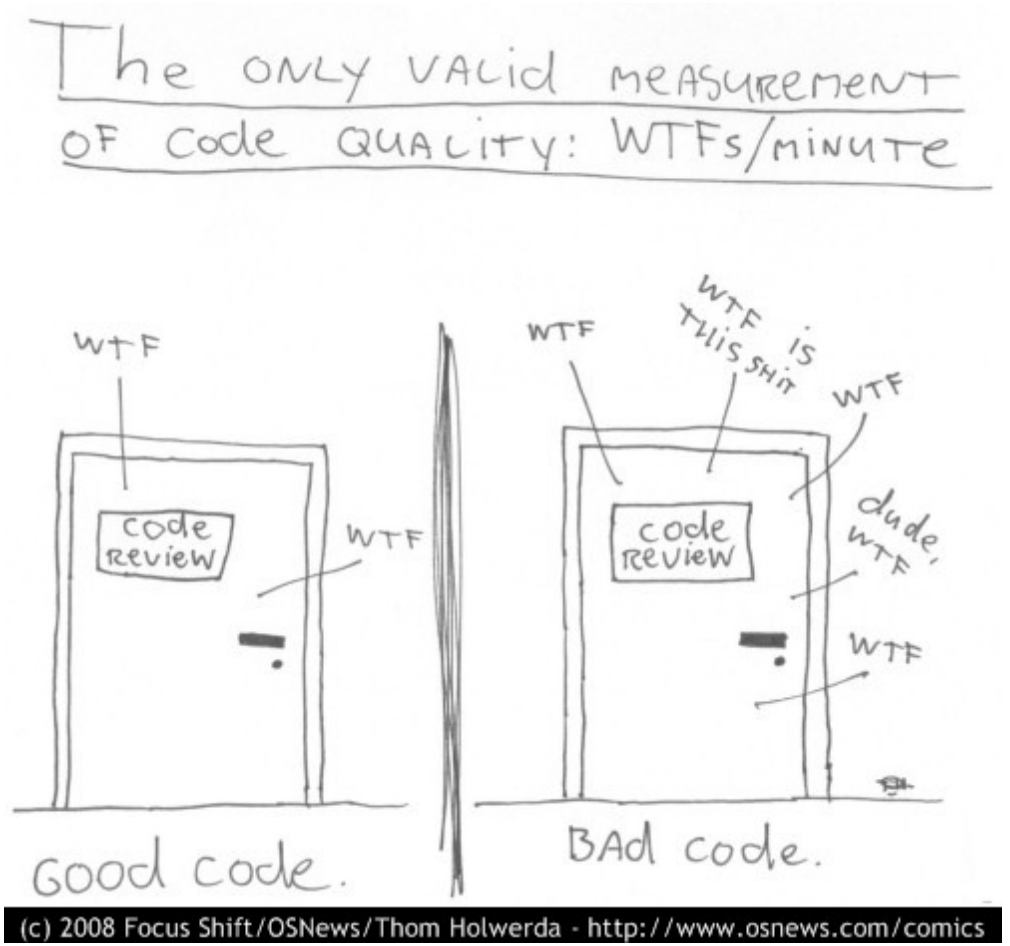
- Lähtöisin eXtreme Programmingista
- Kaksi ohjelmoijaa: Kontrolloija (controller) ja tarkkailija (observer)
  - Parempi laatu: vähemmän virheitä
  - Tiimityö ja kommunikaatio
  - Oppiminen

Tampereen yliopisto
Tampere University

Koodin katselmointi

Koodia luetaan ja mietitään:
- Tekeekö se sitä, mitä pitää,
- Vastaako se koodikonventioihin
- Sisältääkö se virheitä

Kuva: osnews.com/Thom Holwerda



Tampereen yliopisto
Tampere University

Sopimussuunnittelu: testaus

Tampereen yliopisto
Tampere University

Sopimussuunnittelu ja testaus

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

Tampereen yliopisto
Tampere University

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 ⇒ assert ei tee mitään

Invariantti

```
inline void JarjestettyTaulukko::Invariantti()
{
#ifndef NDEBUG
    // Invariantti: alkiot 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();
}
```

Tampereen yliopisto
Tampere University

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