Föreläsning 10: Dokumentering, testning

Dokumentation

Varför dokumenterar man kod? Måste man alltid dokumentera kod? Om man vet att koden man skriver ska köras mer än en gång av någon annan än en själv behövs någon form av dokumentation. I stora system finns flera typer av dokument som ska skrivas

Designdokument

De egenutvecklade komponenterna beskrivs i detalj. Beroende på projektutvecklings-metod beskrivs de mer eller mindre detaljerat i förväg. UML-diagram som klasscheman, tillståndsdiagram och sekvensdiagram ger bra översikt över systemet. Dessa diagram gås igenom i nästa kurs – programutvecklingsteknik.

Även kodkommentarer ingår i designdokumentation. Det finns två typer av kodkommentarer. Dels block-kommentarer som kommenterar enskilda block, funktioner och klasser samt enskilda kommentarer vid snåriga kodsnuttar.

Kodkommentarer i Python

I python skriver man funktionskommentarer (och klasskommentarer) direkt efter deklarationen. Man skriver kommentaren som en vanlig flerradssträng (inleds och avslutas med tre dubbelfnuttar ”””)

```
def print_schedule(data):
    """
    The method prints the schedule on stdout (standard out)

    data - a dictionary containing the schedule
    """
```

Med modulen *pydoc* kan man sedan generera dokumentation, t.ex. i html-format.

Kodtestning I python

Det finns en modul i python - *doctest* som kan tolka tester i kodkommentaren. Man kan klistra in en testkörning av funktionen och den kommer att utföras. Om man kommenterar t.ex. *search_data* från lab2. Först behöver vi sätta upp parametern, sedan anropar vi funktionen och noterar vad vi får för resultat.

```
#############################################################
##
## search_data
##
def search_data(what, data):
    """
    Prints the required schedule if found

    >>> data = {'_18_nov': ['Fre', '15:00-17:00', '18', 'nov', 'Tilpro']}
    >>> search_data("18", data)
    Fre 15:00-17:00 18 nov Tilpro
    """

    # Här börjar koden

    found = False
    for key in data:
        if (what in key):
            found = True
            list = data[key];
            for item in list:
                print item,
            print "";
    if (found == False):
        print "Nothing happens", what

def _test():
    import doctest
    doctest.testmod()
```

Kör vi metoden test får vi inget svar, det är normalt eftersom testet fungerar. Ändrar vi testet att söka efter 99 istället för 18 får vi svaret:

```
Failed example:
    search_data('99', data)
Expected:
    Fre 15,00-17,00 18 nov Tilpro
Got:
    Nothing happens 99
```

Det är mycket värdefullt dokumentationen använda ett anrop för att beskriva användningen.

Unit test

Det behövs många tester för att testa stora system och alla dessa går inte att skriva i kodkommentarer för då blir inte kommentaren läslig. Istället skriver man unittest. Det finns ett paket *unittest* för det. För att sätta upp förutsättningarna för testet använder man *setUp* (man tar ner dem med *TearDown*).

Ett viktigt skäl till testning är att göra **regressionstest**, d.v.s. at kontrollera att den befintliga koden inte ändrats om man tillför eller ändrar någon annanstans. Genom att automatisera testningen med kod kan man med en knapptryckning göra tusentals regressionstest vilket inte är ovanligt.

Det är lätt att skriva program som fungerar när användaren gjort som tänkt. Mycket mer utmanande att skriva program där användaren gör på annat sätt.

```
import unittest
from test import test_support

class MyTestCase1(unittest.TestCase):

    # Only use setUp() and tearDown() if necessary

    def setUp(self):
        ... code to execute in preparation for tests ...

    def tearDown(self):
        ... code to execute to clean up after tests ...

    def test_feature_one(self):
        # Test feature one.
        ... testing code ...

    def test_feature_two(self):
        # Test feature two.
        ... testing code ...

    ... more test methods ...

class MyTestCase2(unittest.TestCase):
    ... same structure as MyTestCase1 ...

... more test classes ...

def test_main():
    test_support.run_unittest(MyTestCase1,
                              MyTestCase2,
                              ... list other tests ...
                             )

if __name__ == '__main__':
    test_main()
```

Testdokument

Alla testfall måste dokumenteras. Testfallen är starkt kopplade till kraven och funktionsbeskrivningarna. Ett testfall ska ha noggranna instruktioner hur testet genomfördes så att det går att återupprepa. Testfallet ska dessutom beskriva förväntat resultat är och vad utfallet blev. Det är viktigt att hålla reda på vilken version av koden man testar och ha med det i dokumentationen. Man brukar tala om en testmatris.

Id	Beskrivning	Förväntat utfall	Utfall

Odokumenterade tester är värdelösa!

Får man en felrapport på något som man anser sig ha testat så måste man kunna återskapa (reproducera) testet.

Vilken annan typ av dokumentation skrivs?

Om man hårdrar all dokumentation kan det delas in i två kategorier, antingen är det någon form av beslutsunderlag eller så är det någon sorts beskrivning. Ibland är det bådadera.

Beslutsunderlag
Dokument utgör ofta beslutsunderlag till om projektet ska få fortsätta eller t.o.m. om det överhuvudtaget ska få starta.

Beskrivning
System som inte är dokumenterade går varken att använda eller underhålla. Vid slutlig leverans måste systemet vara fullständigt dokumenterat. Det är inte bara slutanvändaren som behöver dokumentation. Även inom projektet behöver olika komponenter, kod, delsystem och utvecklingsverktyg dokumenteras. Tänk på att den kod du skriver kanske behöver underhållas i flera decennier framöver.

Vanliga dokument i projekt

Säkerhetsaspekter
Dessa dokument är inte så vanliga ännu men de kan komma att bli allt vanligare i framtiden. Det man tittar på är främst systemets interaktion med omvärlden, t.ex. login, get/post meddelanden, ftp m.m. För lite mer hårdvarunära system (typiskt skrivna i C/C++) kan det vara värt att titta på ”buffer underrun/overflow ”problem eller ”stack walking”.

Systembeskrivning
De komponenter som ingår i systemet beskrivs här ofta med enkla skisser. Komponenterna kan också vara ren hårdvara. Om dokumentet används som beslutsunderlag innehåller det ofta kostnad- krav- och riskkalkyler.

För underhåll ska även manualer för olika komponenter som t.ex. databaser ingå.

Leveransdokument
Ett stort projekt har ofta flera delleveranser. Det kan vara såväl interna leveranser inom projektet som mer eller mindre färdiga produkter till slutkund. Vid leverans måste man deklarera vad det är man levererar. Dokumentet innehåller en lista över alla filer som ingår (t.ex. zipfil, REDAME, installationsprogram) samt på vilket sätt det levereras (CD-skiva, ftp m.m.).

> **Kända begränsningar**
> Testfall som systemet inte klarar av ännu beskrivs som kända begräsningar av systemet.
> **Kända fel**
> Buggar som inte rättats ännu beskrivs som kända fel så att man inte får nya felrapporter.

Användarmanualer
Vid leverans till slutkund ska beskrivningar på hur man installerar, använder, administrerar och underhåller systemet följa med.

Kvalitetsdokument
Kvalitetsdokument som ISO9000 förekommer ibland. De utgör en utvärdering av projektprocessen med kvalitetstänkande som testning, tillräckligt antal resurser m.m.