Verifikasjon og validering

19. oktober 2006 - INF3120

Nils Christian Haugen & Stein Grimstad

Hvem er vi?

- Nils Christian Haugen
  - Chief Scientist i Objectnet
  - Utdannelse fra NTNU
  - E-post: nch@objectnet.no
- Stein Grimstad
  - Forsker ved Simula Research Laboratory
  - Seniorrådgiver i Objectnet
  - Utdannelse fra Universitet i Oslo
  - E-post: steingr@simula.no

Agenda

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
- (pause)
- Testing

Mål med forelesningen

- Kjenne til og forstå viktige begreper
- Kjenne til og forstå viktige teknikker
- Forstå at verifikasjon og validering er viktig
- Ha et utgangspunkt for å planlegge og gjennomføre verifikasjon og valideringsaktiviteter i prosjektoppgaven (dog ikke vektlagt tema i prosjektoppgaven)

Pensum

- Se web
- Vi vil ikke gjennomgå hele pensum
  - Selvstudium
- Materialet som presenteres er
  - Litt fra Sommerville
  - Litt eget
- I tillegg anbefales det å lese:
  - Myers : "The art of software testing" ($$$$$)
  - Beck: "Test-driven Development: By Example"
  - Mugridge & Cunningham: "Fit for Developing Software"
  - http://www.junit.org/
  - http://fit.c2.com/

Agenda: Validering og verifikasjon

- Introduksjon
- Validering og verifikasjon (V & V)
  - Definisjoner
  - Motivasjon
  - Feil
- Prototyping
- Inspeksjon
- Testing

V & V: Definisjoner

- Validation: “Are we building the right product?”
- Verification: “Are we building the product right?”

(Sommerville)

- Hvem definerer hva som er riktig?
- Arbeidet med å utbedre avvik som avdekkes er ikke en del av validerings- og verifikasjonsaktivitetene

V & V: Motivasjon

- Mennesker gjør feil
  - Det vi lager er ikke det vi burde laget
  - Det vi lager har defekter
- Sikre riktig kvalitet i produktet/tjenesten som lages
  - Ikke nødvendigvis ”best mulig” kvalitet: Godt nok
- Jo senere en feil oppdages, desto mer alvorlig er det
  - Typisk regner man en faktor i kostnad på 10 mellom de forskjellige fasene
  - Kan være forretningskritisk (i ytterste konsekvens kan menneskeliv gå tapt)

V & V: Eksempel på feil

V & V: Hvorfor finnes feil selv etter lansering?

- Det som er laget er feil (ikke det kunden vil ha)
- Ikke alle feil prioriteres rettet
  - Kategori 1: Kritiske feil som MÅ rettes (lansering holdes igjen til feilen er rettet)
  - Kategori 2: Kritiske feil som bør rettes (betydelig reduksjon av kvaliteten)
  - Kategori 3: Ikke-kritiske feil (kosmetiske feil)
- Kan ikke verifisere alle kombinasjoner av input til et software system
- "Confirmation bias"
  - Tendens til å teste bekreftelser, i motsetning til å prøve å falsifisere.
  - NB: Dette peker på en vesentlig forskjell i holdning mellom det å utvikle og det å teste. En god utvikler er "konstruktiv", mens en god tester er "destruktiv". Mange organisasjoner velger derfor å skille rollene, dvs. å ha egne testere.
- Tendens til å tro at sist funnet feil er "siste feil".

V & V: Tre viktige teknikker for V & V

- Prototyping
  - Lage et utkast til hvordan det endelige systemet kan se ut
  - Benyttes tidlig i utviklingsprosessen
  - Testes mot sluttbruker
- Inspeksjoner
  - Gjennomgang av leveranser
  - Benyttes i alle faser av utviklingsprosessen
  - Utføres hovedsakelig av mennesker, men kan være programmer
- Testing
  - Sjekke hvor godt systemet oppfyller kravene
  - Bør gjøres gjennomgående i utviklingsprosessen
  - Resultatet av kjøringer sammenlignes med forhåndsdefinert fasit

Agenda: Prototyping

- Introduksjon
- Validering og verifikasjon
- Prototyping
  - Formål
  - Verktøy
  - Eksempler
- Inspeksjon
- Testing

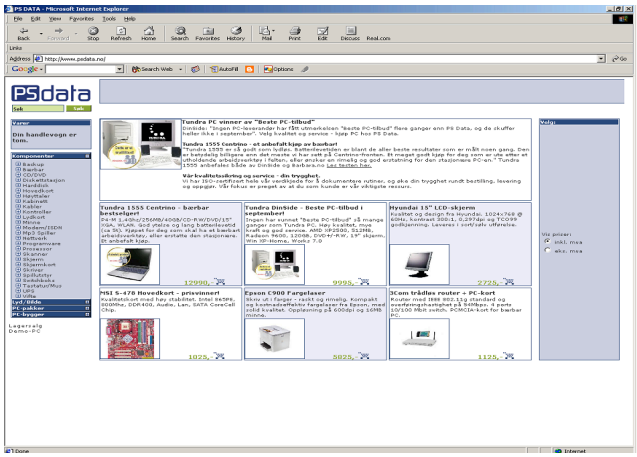
Prototyping: Formål

- Lager utkast til hvordan systemet kan se ut
  - Tegninger (storyboarding)
  - Skjermbilder uten funksjonalitet
- Brukes til validering av systemet i krav/analyse-fasen
  - Enkelt for brukere å forholde seg til (i forhold til modeller og dokumenter)
  - Avdekker effektivt manglende/feil krav og forretningsregler
- Ulemper:
  - Gode prototyper kan gi falske forventninger i forhold til hvor langt prosjektet har kommet
  - Brukere kan bli opphengt i irrelevante detaljer

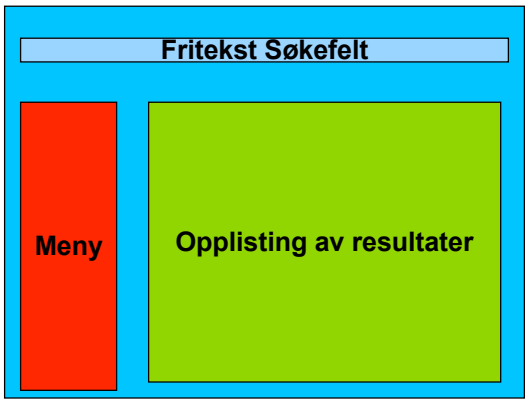
Prototyping: Verktøy

- Penn og papir
- Whiteboard
- Tegneprogrammer
- PowerPoint
- HTML-editorer
- GUI-byggere
- Modellbaserte prototypingsverktøy
- Programmeringsspråk

Prototyping: Eksempler (1)



Prototyping: Eksempler (2)



Agenda: Inspeksjon

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
  - Dokumentgjennomgang
  - Kodegjennomgang
  - Automatisert statisk kodeanalyse
  - Matematisk verifikasjon
- Testing

Inspeksjon: Dokumentgjennomgang

- Gjennomgang av dokumenter med formål å finne feil og mangler
- Forskjellige teknikker kan benyttes
  - Skrive dokumentet sammen i par (kontinuerlig inspeksjon)
  - Forskjellige typer gjennomgangsmøter (dokumentet presenteres i møtet, distribueres på forhånd, forskjellige typer fasilitatorer...)
  - Aspektbasert inspeksjon
- En typisk fremgangsmåte
  - dokumentet distribueres for gjennomlesning
  - arrangerer gjennomgangsmøte
- Hvem bør gjennomgå dokumentet?
  - De som skal ha systemet
  - De som skal bruke dokumentet
  - De som har vært delaktige i utformingen av dokumentet
  - Eksperter (rollen / forretningsområdet)
  - Personer du av ulike årsaker ønsker en godkjennelse fra

Inspeksjon: Kodegjennomgang

- Gjennomgang av kildekoden for å finne feil og mangler
  - Funksjonelle feil (avvik fra design og kravspesifikasjon)
  - Avvik fra kodestandard og retningslinjer (for eksempel navngiving av klasser)
  - Tekniske feil (for eksempel feil i en hashmap-funksjon)
  - Testbarhet (for eksempel dokumentasjonsmangler, muligheten for isolasjon, etc)
- Forskjellige teknikker kan benyttes
  - Parprogrammering (kontinuerlig kodegjennomgang)
  - Distribusjon til en eller flere for gjennomlesning
  - Koden gjennomgås av en review-gruppe i et møte
  - Aspektbasert inspeksjon
- Hvem bør gjennomgå kode?
  - Sjefsprogrammerer/designere
  - Juniorprogrammerere (opplæring)
  - Testere
  - Tekniske eksperter
  - Arkitekter

Inspeksjon: Automatisert statisk kodeanalyse

- Programmer som analyserer kode og avdekker avvik fra spesifiserte krav
  - Sjekker at alle klassenavn begynner med store bokstaver
  - Sjekker at metoder ikke er for lange
  - Sjekker at input-paramtere valideres
  - Sjekke at alle metoder er dokumentert
  - Ikke-eksekverbar kode
  - Variabler som aldri brukes
  - Minnehåndtering
- Integreres i utviklingsmiljøet, versjonshåndteringssystemet eller byggesystemet
  - For eksempel kan kode hvor verktøyene rapporterer feil i nektes innsjekking
- Raskt å kjøre (billig) i motsetning til inspeksjon
  - Men mange ting fanges ikke opp her....
- Eksempel på verktøy:
  - Kompilatorer, IntelliJ, Eclipse, jDepend, jMetric

Inspeksjon: Matematisk verifikasjon

- Matematisk bevis for at programmene oppfører seg i henhold til kravspesifikasjon
  - Matematisk bevise konsekvensen av hver programmeringsinstruks
  - Fordrer at kravspesifikasjonen er uttrykt på et egnet format:
    - *void* Sort (A,n);
    - *Inndata: n tall A[0..n-1] vilkårlig rekkefølge*
    - *Utdata: A'[i-1] <= A'[i], 0< i < n*
- Problem
  - Programmeringsspråk er ikke nødvendigvis veldefinerte
  - Komplisert for avanserte strukturer
  - Bevisene er i seg selv kompliserte og sjansen for feil er stor
- Veldig, veldig dyrt (typisk $1000 per kodelinje)
  - Romferger
  - Atomreaktorer
  - T-banen i Paris
- Brukes lite i industrien
- For mer informasjon: Ta kurset IN217 (INF 4230)

Agenda: Testing

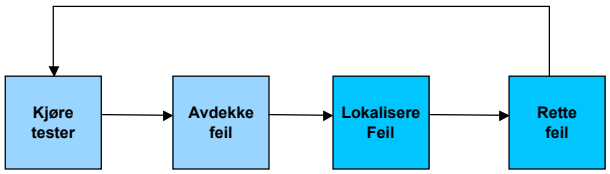
- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
- Testing
  - Innledning
  - Testing vs. debugging
  - Typer av test
  - Testplan
  - Testdata
  - Ekvivalensklasser
  - Gjenbruk
  - Introduksjon til Fit

Testing: Innledning

- En form for ”induktiv bevisføring”
  - Vise (sannsynliggjøre) at alle svaner er hvite ved å observere et representativt utvalg av alle svaner.
- Test er både verifikasjon og validering
- Testing kontrollerer ny funksjonalitet, samt at eksisterende funksjonalitet fortsatt virker (regresjonstesting)
- Effektiv testing gjøres ved:
  - Testbare krav
  - Testbare programvarestrukturer
  - Bruk av ekvivalensklasser
  - Automatiske tester
  - Gjenbruk

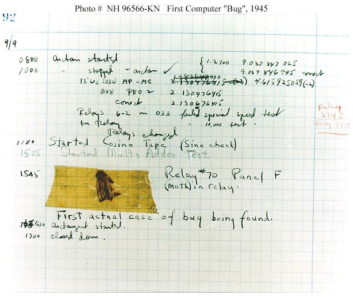
Testing: Testing vs. debugging

- Testing er ikke det samme som debugging
- Testing er å finne feil
- Debugging er å lokalisere og rette feil
- Begge deler gjøres tradisjonelt i test-fasen av prosjektet, men smidige metodikker gjør det hele tiden



Testing: Det første virkelige tilfelle av ”bug”

- 9. september 1945, kl. 3:45 p.m., fant forskere ved Harvard-universitetet årsaken til at Mark II Aiken Relay kalkulatoren oppførte seg merkelig
- En møll ble funnet fanget mellom punkter på relé #70, panel F
- Maskinen ble ”debugget” med en pinsett!
- Dokumentert i loggen som ”First actual case of bug being found.” med møllen tapet inn ved siden av



Testing: Typer av test (1)

- Enhetstest
  - Tester at objektet virker isolert
  - Simulerer omgivelsene til objektet
  - Bruker konstruerte testdata
  - Utføres av utviklere
  - Skrives ofte som automatiske tester
- Modultest
  - Tester at objektet virker sammen med de andre objektene i modulen
  - Simulerer som regel omgivelsene til modulen
  - Bruker konstruerte testdata
  - Utføres ofte av utviklere
  - Gjøres ofte manuelt, men kan med fordel automatiseres

Testing: Typer av test (2)

- Systemtest
  - Tester at hele systemet oppfører seg korrekt
  - Testdata konstrueres ofte
  - Utføres ofte av en egen testgruppe
  - Inkluderer også ikke-funksjonelle tester
    - Ytelsestest (tester ytelse = hastigheten på én transaksjon)
    - Stresstest (tester skalerbarhet = hastigheten på mange samtidige transaksjoner)
    - Recoverability-test (tester systemets håndtering av uforutsette avbrudd)
- Akseptansetest
  - Tester at systemet lar brukerne gjøre det de trenger
  - Tester med reelle data
  - Utføres gjerne i samarbeid mellom kunder og testere
- Betatest
  - Utvalgte kunder tar i bruk systemet før offisiell lansering

Testing: Testplan

- V&V-plan (kvalitetssikringsplan) er en del av prosjektplanen
- Testplanen er en del av en V&V-plan
  - Fokus på feilavdekking/-retting ved kjøring av programmer
- Testplanen bør inneholde beskrivelse av prosedyrer for
  - enhetstest
  - modultest
  - systemtest
  - akseptansetest
- Stor sannsynlighet for at man må re-planlegge som en følge av forsinkelser og endring av ”scope” under utvikling

Testing: Eksempel på innhold i testplan

- Generelt:
  - Tidsplan (hvilke tester når)
  - Ressursbruk (mennesker/systemer/maskiner/data)
  - Testleveranser (testscript, testrapporter, dokumentasjon, etc.)
- For hvert testdesign
  - Gjennomføring av test for en eller flere krav
  - Krav som skal testes/ikke testes
  - Hvilke test scenarier inngår
- For hver test scenarie
  - Situasjoner som tester krav til programvaren
  - Inndata og forventet utdata (f.eks. hvilke kundedata som skal ligge i databasen)
  - Betingelser (f.eks. at alle moduler er oppe og kjører normalt)
  - Testprosedyre (detaljert steg-for-steg beskrivelse for gjennomføring av testen)

Testing: Testdata

- Tilstanden til testdata må resettes mellom hver test kjøring
  - Eksempel:
    - Teste at en kunde som flytter fra Oslo til Svalbard slutter å betale moms
    - Ok første gang
    - Andre gang feiler testen siden kunden ble flyttet i forrige testkjøring
- Utfordringer:
  - å finne komplette/realistiske testdata
  - å isolere testdata mellom tester
  - å teste konfidensielle data
- Testdatabaser får fort dårlig datakvalitet
  - Blir ikke vedlikeholdt på linje med produksjonsdatabaser
  - Data blir herjet mer med enn data i produksjon
  - Feil i kode kan føre til inkonsistente tilstander

Testing: Ekvivalensklasser og grensesnittsverdier

- Hver klasse av inndata som fører til samme oppførsel tilhører samme “ekvivalensklasse”.
- Testing gjøres ikke på alle data
  - Alle ekvivalensklasser bør være representert
  - Grenseverdier, ekstremverdier og feilverdier bør være representert
- Eksempel:
  - Ved innskudd på sparekonto skal kunder få forskjellig rente
    - De med < 100 000 på konto får 0,5% rente
    - De med >= 100 000 på konto får 1% rente
    - Overtrekk er ikke tillatt
  - Hvilke data bør man teste med?

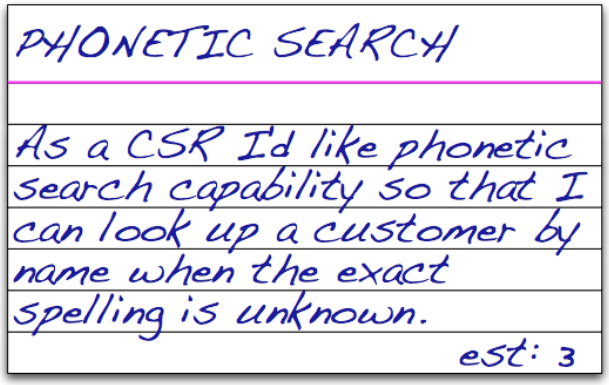
Testing: Gjenbruk

- Kode som gjenbrukes er allerede testet (forhåpentligvis!)
  - Mange penger å spare
- Men ikke nødvendigvis testet for ditt bruk
  - ikke testet i din anvendelse
  - ikke testet med din konfigurasjon

Testing: Automatisk testing med Fit

- Rammeverk laget av Ward Cunningham (2002)
- Verktøy for å forbedre samarbeid og kommunikasjon i teamet
  - Eksempelbasert spesifikasjon
  - Utvikles av kunde, tester og programmer i samarbeid
- HTML dokumenter
  - Kan leses av alle involverte
  - Kjøres som automatiske tester
- Finnes i utgave for Java, C#, C++, Python, Ruby, Smalltalk...

Fit eksempel: Feature uttrykt som user story



Fit eksempel: Fit-tabell spesifiserer oppførsel

Phonetic search for customer	
Last name	Hits?
Myre	Mire, Myhre, Myre
Myhre	Mire, Myhre, Myre
Mire	Mire, Myhre, Myre
Myrer	[none] expected Myhre, Myre actual
Myren	[none]
Myr	[none]
Myhr	[none]
Mir	[none]

Fit eksempel: Fixture knytter tabell til system

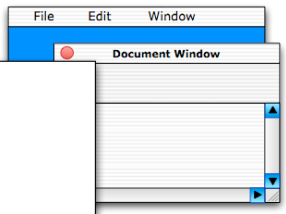
Fit table

System under test

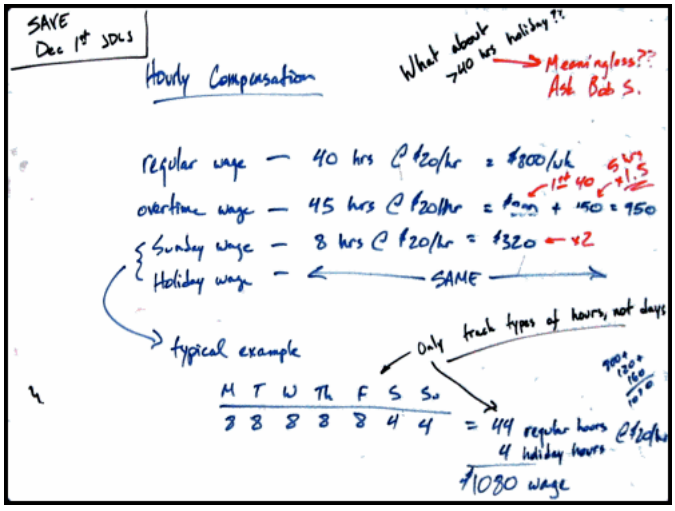
Fixture

Phonetic search for customer	
Last name	Hits?
Myre	Mire, Myhre, Myre
Myhre	Mir
Mire	Mir
Myrer	[no Myh
Myren	[no
Myr	[no
Myhr	[no
Mir	[no

```
class PhoneticSearchForCustomer
  extends ColumnFixture {

  public String lastName;

  public String[] hits() {
    DNF dnf = DNF.getInstance();
    Person[] persons =
      dnf.soundexSearch(lastName);
    return lastNamesOf(persons);
  }

  private String[] lastNamesOf(Person[] p) {
  // ...
}
```

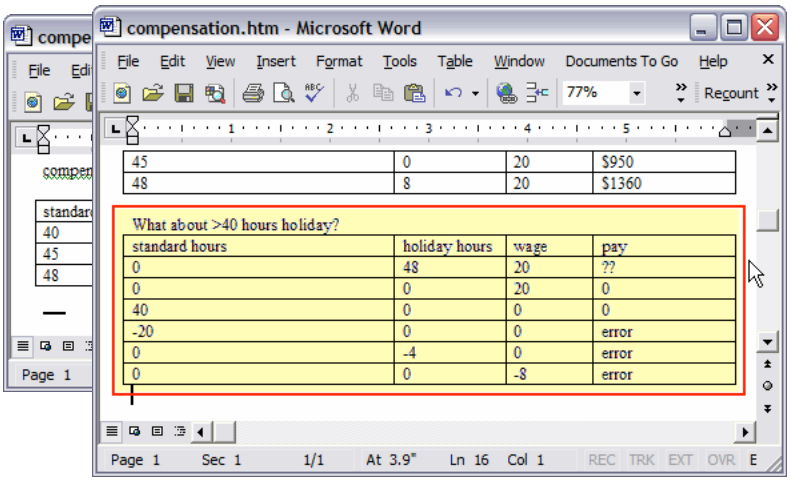


Fit arbeidsprosess: Beskrivelse av en feature



Source http://fit.c2.com/

Fit arbeidsprosess: Raffinering av eksempler



Source http://fit.c2.com/

Fit arbeidsprosess: Fitifisering av tabeller

```
public class WeeklyCompensation : ColumnFixture
{
    public int StandardHours;
    public int HolidayHours;
    public Currency Wage;

    public Currency Pay()
    {
        WeeklyTimesheet timesheet = new WeeklyTimesheet(StandardHours, HolidayHours);
        return timesheet.CalculatePay(Wage);
    }
}
```

What about >40 hours holiday?

Payroll.Fixtures.WeeklyCompensation			
StandardHours	HolidayHours	Wage	Pay()
0	48	20	??
0	0	20	0

Page 1 Sec 1 1/1 At 1.5" Ln 4 Col 1 REC TRK EXT OVR E

Source http://fit.c2.com/

Fit arbeidsflyt: Inkrementell utvikling

result.htm - Microsoft Word

File Edit View Insert Format Tools Table Window Documents To Go Help

77% Recount

compensation: first 40 hours x1, overtime x1.5, holiday x2

Payroll.Fixtures.WeeklyCompensation			
StandardHours	HolidayHours	Wage	Pay()
40	0	20	$800
45	0	20	$950
48	8	20	$1360 expected $1040 actual

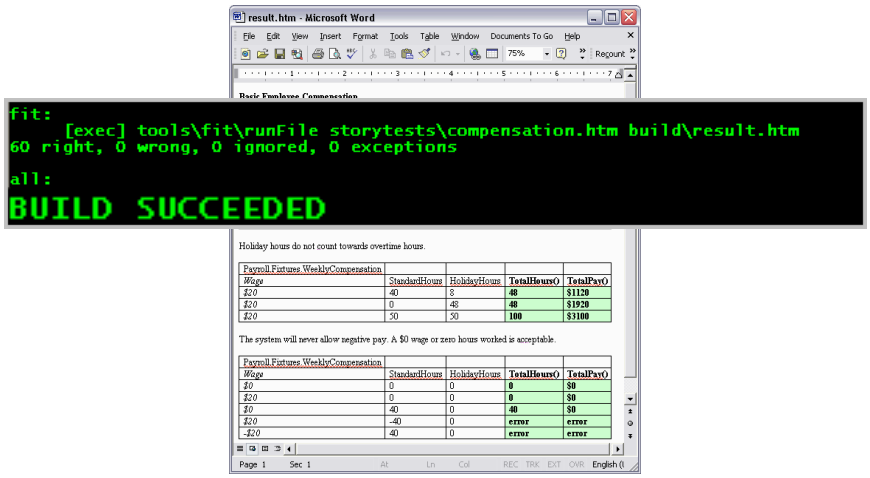
What about >40 hours holiday?

Payroll.Fixtures.WeeklyCompensation			
StandardHours	HolidayHours	Wage	Pay()
0	48	20	$1920 expected $0 actual
0	0	20	0
40	0	0	0

Page 1 Sec 1 1/1 At 1" Ln 1 Col 1 REC TRK EXT OVR E

Source http://fit.c2.com/

Fit arbeidsflyt: Ferdig Fit-dokument



Source http://fit.c2.com/

Fit: Forskjellige fixture-typer

ColumnFixture

eg.Division		
numerator	denominator	quotient()
1000	10	100.0000
-1000	10	-100.0000
1000	7	142.85715
1000	.00001	
4195835	3145729	

RowFixture

eg.music.Display					
title	artist	album	year	time()	track()
Handy Man	James Taylor	JT	1977	3.30	7 of 12
Sailing To Philadelphia	James Taylor	October Rose	2001	5.47	3 of 3
Ananas	James Taylor	Hourglass	1997	5.73	5 of 13
Another Grey Morning	James Taylor	JT	1977	2.73	4 of 12
Copperline	James Taylor	New Moon Shine	1991	4.37	1 of 12

ActionFixture

eg.music.Realtime			
enter	select	2	pick an album
press	same album		find more like it
check	status	searching	
await	search complete		
		ready	
s		2	

Source http://fit.c2.com/

Q&A