Introduktion

Objekt-orienterad Programmering och Design (TDA551)

Alex Gerdes, HT-2016

Vad är ett ”bra” program?

- Korrekt?
- Effektivt?
- Användbart?
- Flexibelt?
- Robust?
- Skalbart?

- Enkelt?
- Läsbart?
- Testbart?
- Maintainable?
- Reusable?
- Extensible?

Externa faktorer
(berör användare)

Interna faktorer
(berör utvecklare)

Fokus för denna kurs!

Småskalig programmering

- Triviala program
- Få klasser
- Några 100-tal rader kod
- En eller ett fåtal programmerare
- Ingen eller kort livstid
- ”Just do it”

Storskalig programmering

- Mycket komplexa programsystem
- Flera miljoner rader kod
- 100-tals programmerare, geografiskt utspridda
- Lång livstid
- ”Software engineering”
  - Behov av verktyg
  - Behov av processer

Buggar, buggar, buggar

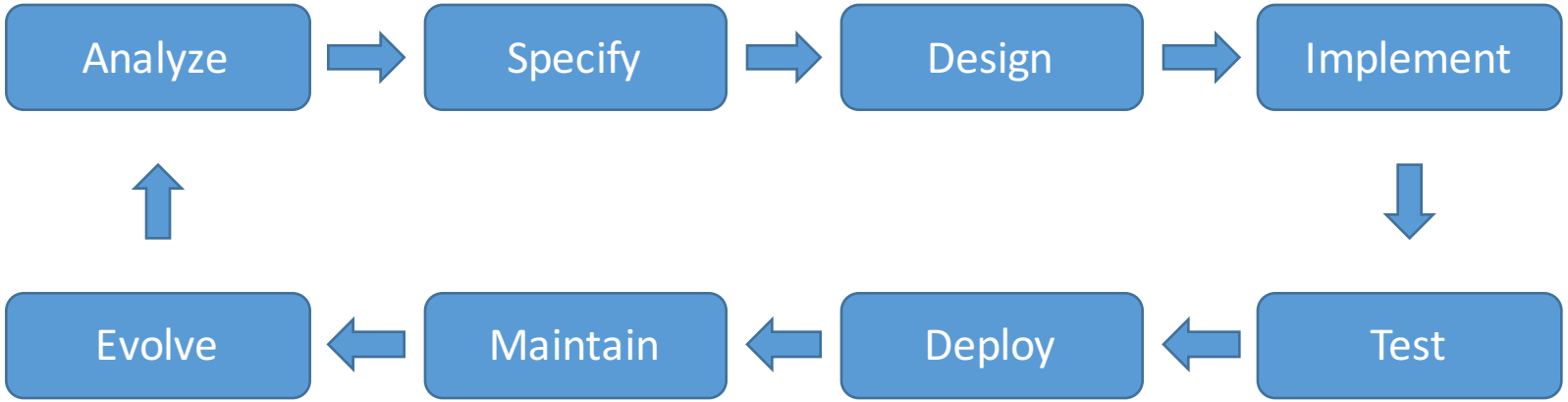
Anything that can go wrong, will go wrong. *(Edward A. Murphy)*

- Kvarstående buggar i programvara som körs i drift:
  - 1-10 buggar/kloc (ordinär industriprogramvara)
  - 0.1-1 buggar/kloc (hög kvalitativ (t.ex. Javas bibliotek))
  - 0.01-0.1 buggar/kloc (extremt säkerkritiskt (t.ex. NASA))
- Murphy's lag skall vara vägledande vid allt ingenjörsmässigt konstruktionsarbete – se till att inget kan gå fel.

Murphy was an optimist. *(Tara O'Toole)*

No operation extends with any certainty beyond the first encounter with the main body of the enemy. *(Helmuth Von Moltke)*

Livscykel för mjukvara



- Kommersiella programsystem har lång livstid och kraven på systemen förändras under deras livstid. Ny funktionalitet läggs till.
- Ingen person kan vara insatt i och förstå alla enskilda delar i systemet.
- Andra programmerare än de som utvecklade systemet utför uppdateringar och systemunderhåll.
- En stabil design och en bra dokumentation är mycket viktigt.

Objekt-orientering

- Objekt-orientering är en *metodik* för att – rätt använd! – reducera komplexitet i mjukvarusystem.

- Rätt använd ger objekt-orientering stora möjligheter till:
  - Code reuse
  - Extensibility
  - Easier maintenance

- Fel använd riskerar objekt-orientering att skapa extra komplexitet
  - ”Big ball of mud”
  - Det finns mycket dålig kod därute…
  - Det finns väldigt många missförstånd kring objekt-orientering.

Objekt-orienterad modellering

- Ett program är en modell av en (verklig eller artificiell) värld.
- I en objekt-orienterad modell består denna värld av en samling objekt som *tillsammans* löser den givna uppgiften.
  - De enskilda objekten har *specifika ansvarsområden.*
  - Varje objekt definierar *sitt eget beteende*.
  - För att fullgöra sin uppgift kan ett objekt behöva *support* från andra objekt.
  - Objekten samarbetar genom att *kommunicera* med varandra via meddelanden.
  - Ett meddelande till ett objekt är en begäran att få en *uppgift utförd*.

Design

- Designen (modellen) utgör underlaget för implementationen.
  - Bra design minskar kraftigt tidsåtgången för implementationen.
  - Brister i designen överförs till implementationen och blir mycket kostsamma att åtgärda.

- Vanligaste misstaget i utvecklingsprojekt är att inte lägga tillräckligt med tid på att ta fram en bra design.
  - Bra design är svårt!!

- I allmänhet bör mer tid avsättas för design än för implementation.

Maintenance (underhåll)

All systems change during their life cycles. This must be borne in mind when developing systems expected to last longer than the first version. (Ivar Jacobson)

- Fel måste fixas.
- Mjukvara måste uppdateras till nya användarkrav.
- Den som ändrar koden är sällan den som ursprungligen skrev den.
- Största delen (~80%) av pengarna och tiden under ett systems livstid går till underhåll.

Utveckla för förändring!

OPC: The Open-Closed Principle

> *Software modules should be open for extension, but closed for modification.*
> *(Bertrand Meyer)*

- Förändring är det enda som är konstant.
- Utveckla *framåt-kompatibelt* – förutsäg *var* förändring kommer behövas.

Design smells: Symptom på dålig design

- Stelhet (rigidity)
  - Svårt att genomföra en ändring pga beroenden med många andra delar av systemet.
- Bräcklighet (fragility)
  - En ändring skapar fel i delar av systemet som inte konceptuellt är kopplade till den ändrade modulen.
- Orörlighet (immobility)
  - Koden är svår att återanvända i andra applikationer.
- Seghet (viscosity)
  - En ”bra” ändring kräver stora insatser, lättare att göra ett ”hack”.
- Oklarhet (opacity)
  - En modul är svår att läsa och förstå.

Evolving systems

> *A system that is used will be changed. An evolving system increases its complexity unless work is done to reduce it.” (Manny Lehman)*

- Alla system ”ruttnar” med tiden, behöver kontinuerligt motverkas.
  - Ursprunglig design fungerar inte för nya krav (för rigid).
  - Inkompetenta utvecklare förstår inte designen.
  - Ändrade krav tas inte på allvar.
  - Snabba hack under tidspress.
  - Nya beroenden mellan komponenter ökar komplexiteten.
  - …

Motverka design smells

- Inse från början att kraven kommer att förändras.
  - ”Utveckla för förändring”

- Använd beprövade tekniker och design patterns för att minska beroenden.
  - On the shoulders of giants…
  - Tala samma språk – lättare för framtida utvecklare att förstå koden

- Refaktorera (refactor) kontinuerligt
  - Refactoring = omstrukturering av koden som bevarar funktionalitet med förbättrar struktur

Code smells: Symptom på dålig implementation

Synliga tecken i koden på att designen är på väg att ruttna:

- Duplicerad kod, ”klipp-och-klistra”-programmering
- Långa metoder
- Långa parameterlistor
- Stora klasser
- Klasser med enbart data
- Publika instansvariabler
- Långa if- eller switch-satser
- Avsaknad av kommentarer
- Onödiga kommentarer
- Oläslig kod
- …

Grundläggande designprinciper

- Enhetlig kodstil
- Genomtänkt namngivning
- Dokumentation

- Modularitet
- Abstraktion
- Decoupling
- Delegering
- Väldefinierade designmönster
- Återanvändning av kod

Modulär design

Fördelar med en *välgjord* modulär design:

- Lätt att utvidga
- Moduler går att återanvända
- Uppdelning av ansvar
- Komplexiteten reduceras
- Moduler går att byta ut
- Tillåter parallell utveckling.

Reflektion Övning ”DrawPolygons”

- Korrekt?
- Enkelt?
- Maintainable?
- Extensible?
- Reusable?

- Modulärt?
- Väl avgränsade ansvarsområden?
- Decoupled?
- Välanvända designmönster?

Live demo

- …

Studentrepresentanter

- Jesper Blidkvist
- Are Ehnberg
- Lovisa Landgren
- Lucas Ruud
- Mattias Torstensson

Övrig kursinformation

- Gå med i Google-gruppen!
  - Maillista som når alla er, samt mig och assistenterna
  - Använd för alla frågor om kursen – administrativt eller innehållsmässigt

- Laborationer
  1. Använda ett ramverk för att implementera spelet Snake (verktyg)
  2. Bygga ett paket för geometriska former (design)
  3. Analys och refaktorering av spelramverket från laboration 1 (mönster och principer)

Nästa modul (1B)

- UML (Unified Modelling Language)

- Recap: reference semantics; klass vs objekt