F8
Programvaruutveckling metoder

EDAF45
Programvaruutveckling i grupp – Projekt
Boris Magnusson
Datavetenskap, LTH

Innehåll

- Hur började det ? Inspiration från tillverkning
- Vattenfallsmetoden
- Delarna i alla metoder: Analys/Krav, Design, Programmering,Testning, Dokumentation …
- Varianter på dokumentbaserade metoder
- Andra ansatser, hitta kraven, prototyper
- Några spektakulära misslyckanden

Inspiration från tillverkning

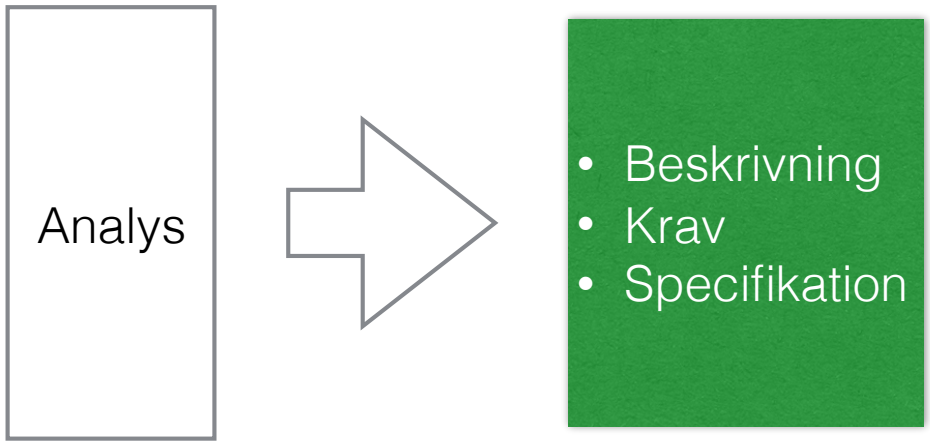
- De första kommersiellt tillgängliga datorerna kom på 60-talet, transistorer gjorde skillnad!
- Att programmera blev lite vanligare.
- Hur gör man detta nya ?
  är det programmering det heter?

- Transistorer gjorde också skillnad för radio.
- Hur gör man en ny ”transistorradio” ?

1. Analys

- Förstå problemet, vad är det man vill ha:
  - Kunna spela modern ”skränig” popmusik
  - Tillräckligt hög volym för ett tonårsrum.
  - Bärbar, inte för tung (ny viktig ”feature”)
  - Batteridriven, driftstid minst …
  - Snabbväljare för 3 kanaler (P1, P2, P3)
  - Färgglad modernt yttre, tilltalande för ungdomar.
  - Kostnad motsvarande 4 veckopengar.

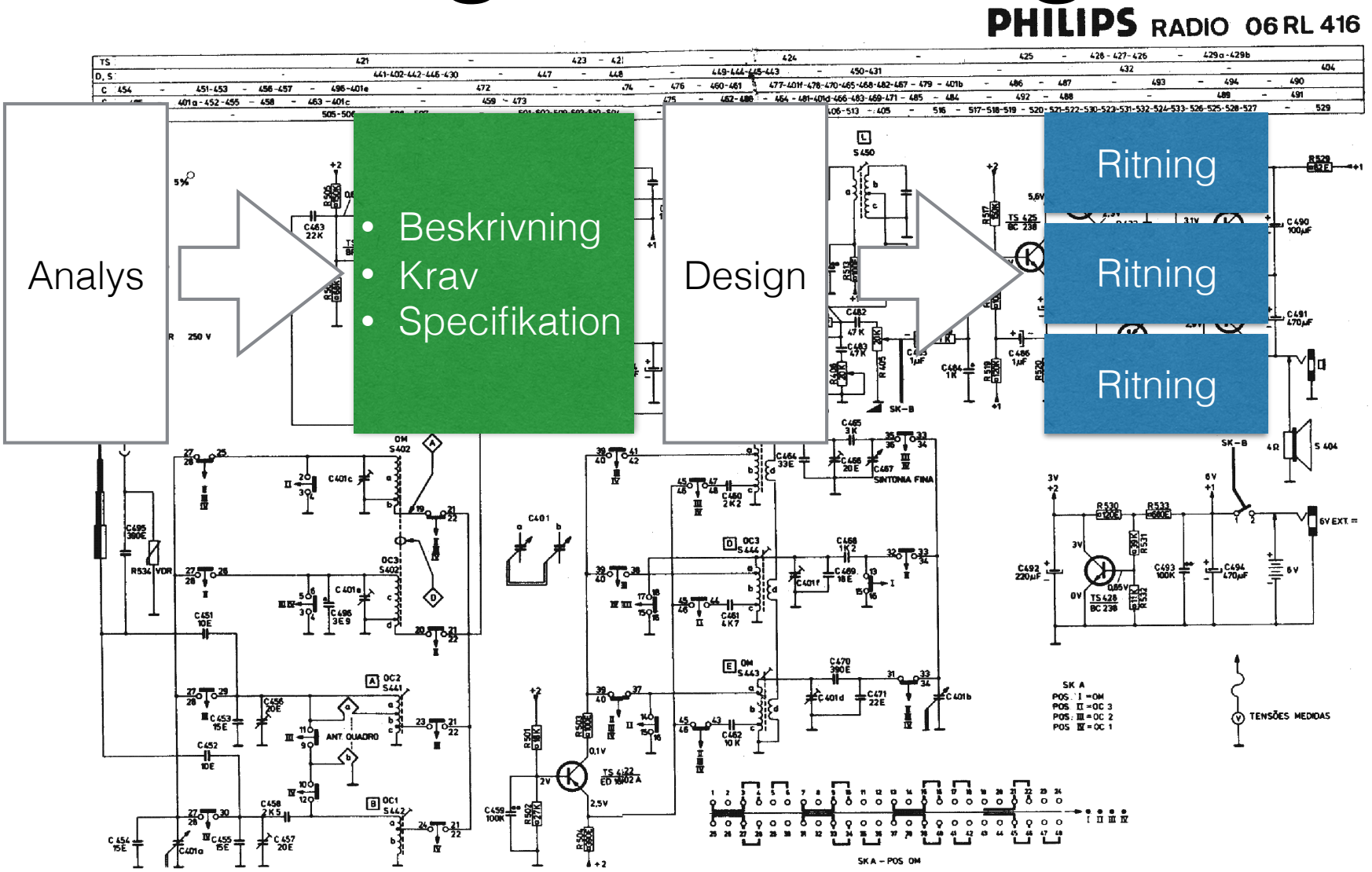
Steg 1, Kravspecifikation



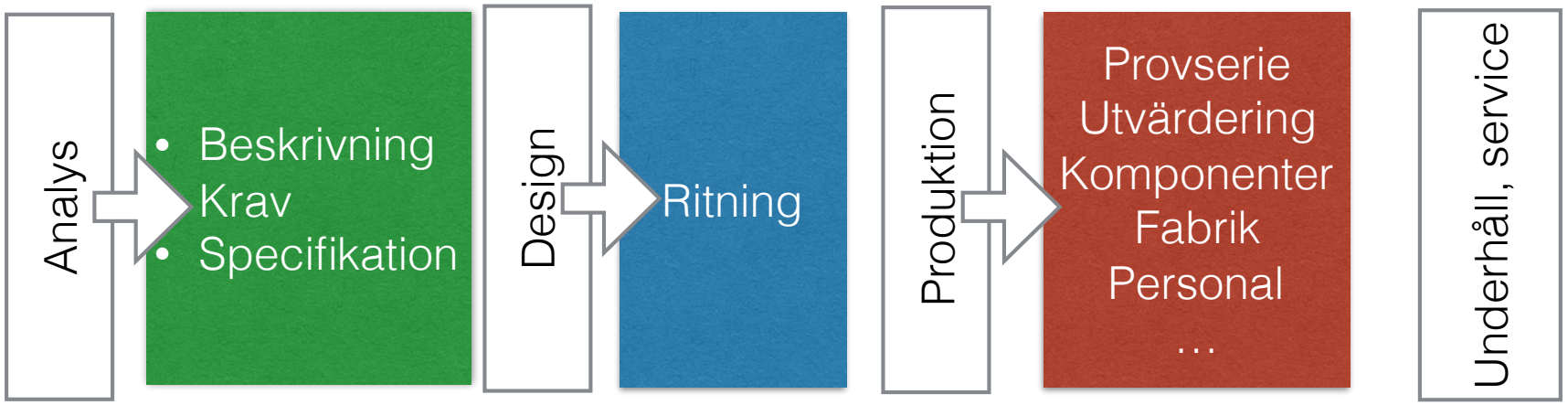
2. Design

- Hur det skall byggas.
- Naturligt att delas upp i 3 aktiviteter
  - Lådan: storlek, bärhandtag, material (trä eller det nya moderna ”plast”?)
  - Strömförsörjning: Batterihållare (ficklampor?), högtalare, (skydd?)
  - Elektronik: hur många transistorer? (dyra), förstärkare - konstruktion, programväljare …

Steg 2, Ritning

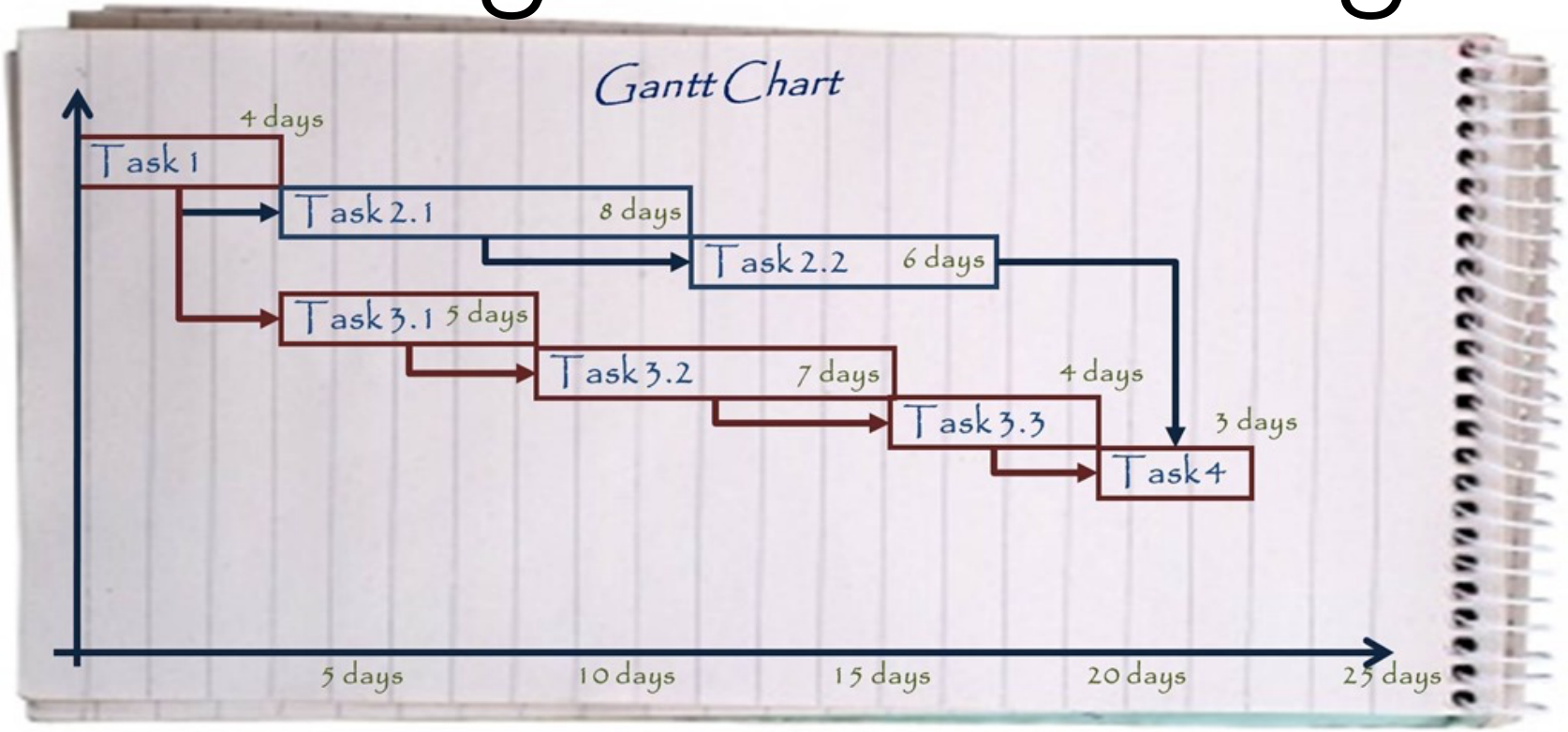


Steg 3: Produktion



- Varje steg avslutat innan nästa börjar.
  - Signering och godkännande
- Varje dokument:
  - Tydligt nog för driva nästa steg

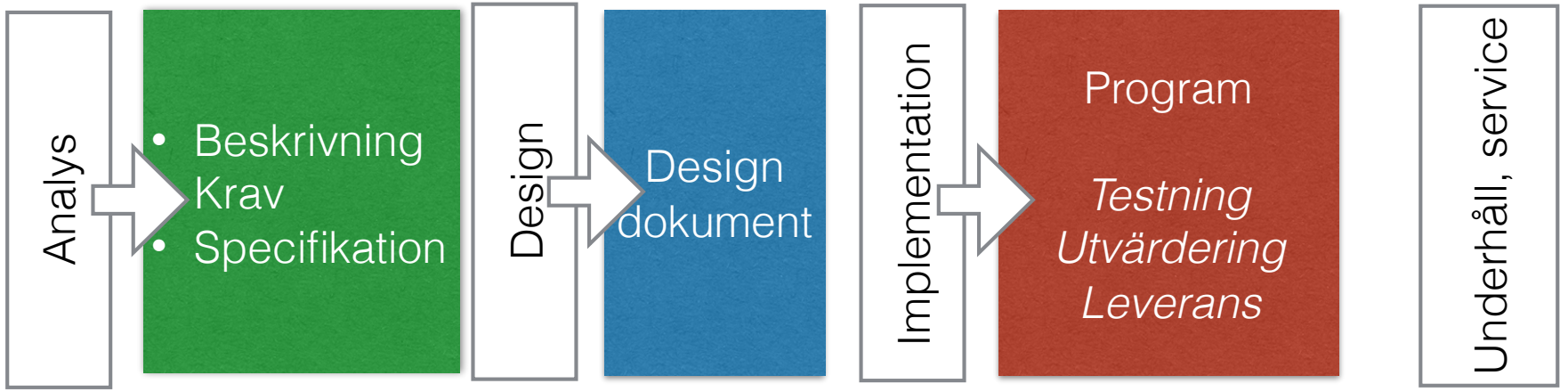
Planering - GANTT-diagram



- Nedbrutet i deluppgifter, tidssatta
- Parallellt eller Sekvens, kritisk linje
- Beslutspunkter

Henry Gantt, uppfann dessa diagram 1917, för att strömlinjeforma produktion av örlogsfartyg under första världskriget.

Applicerat på programvara



- All steg avslutade och godkända innan nästa steg startas.
- Dokumenten tillräckliga för att driva nästa steg.
- Har kommit att kallas **vattenfallsmodellen**.

Vattenfall ? Kommer från?

- Dr. Winston W. Royce: Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26 (August): 1–9. 1970.

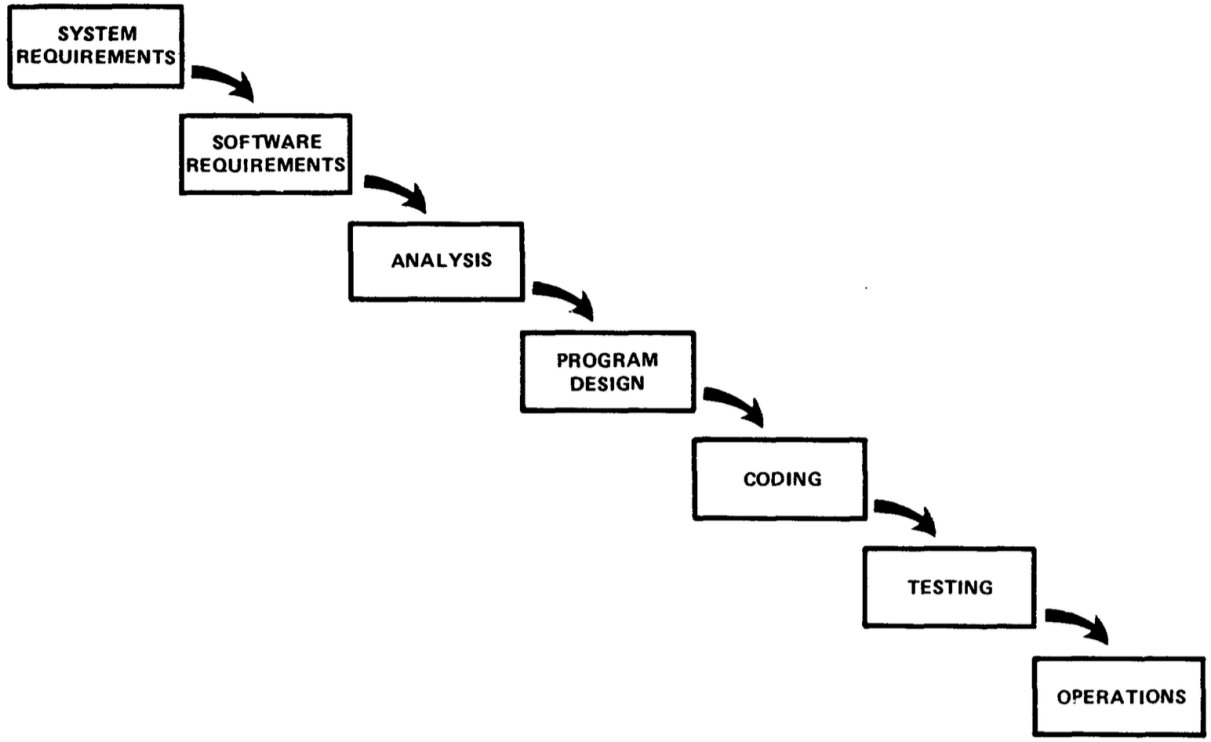


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

- Pappret förklarar varför denna modell *inte* fungerar - men alla läste inte så långt.

Hur tänker man att detta skall fungera?

- Programvaruprojekt ofta problematiska:
  - Försenade
  - Fungerar inte
  - Gör inte vad man förväntat
- Om man utnyttjar beprövad metodik från andra ingenjörsområden borde det väl lösa problemet ?!
- Mer process (ordning och reda helt enkelt)!

Software Process

- Activities:
  - Software **Specification -** functionality, constraints
  - Software **Development -** produce sw which meets spec.
  - Software **Validation -** ensure sw meets customer/users expectations

Ian Sommerville: Software Engineering, Addison Wesley

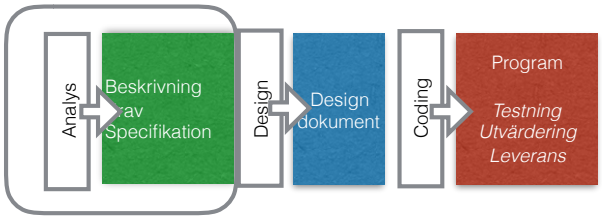
The Waterfall approach

- Softare development process in steps:
  1. **Analysis**, Requirements, and Specification.
  2. System and software **Design**
  3. **Implementation** and unit testing
  4. **Integration** and systems testing
  5. **Operation** and maintenance.
- Each stage result in a document.
- After each stage is defined - it is ’signed off’, and development goes on to the next stage.

Låt oss, för ett ögonblick, ta det på allvar

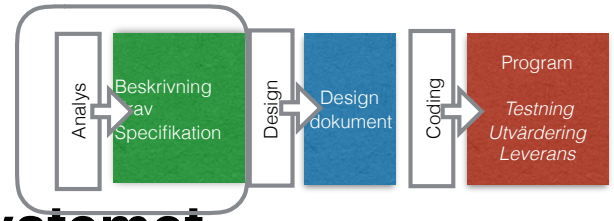
1. Analys: Förstudie

- Feasibility Studies / Förstudie
  - Kan man alls bygga ett sådant system?
  - Alternativ - andra system, inte alls kanske?



Analys: Funktion?

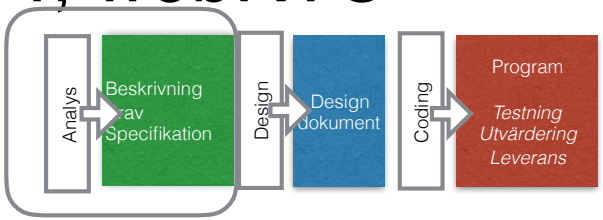
- Förstå domänen
  - t ex en situation i vården.
- Vilka funktionskrav finns:
  - Insamling, intervju, studiebesök, tidigare lösning
  - Format: text, diagram, tillståndsmaskiner, kommunikation, formella notationer.
- Klassificera, sortera, prioritera, lösa konflikter.
- Validering: fullständigt, konsistens, stämmer med förväntningarna.
  - Granskning, genomgång med kund,
- Systemkrav (plattform, tid, minne, kostnad etc).



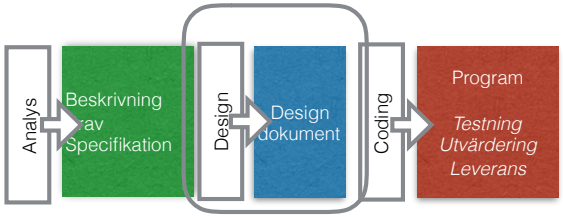
Dokument som fullständigt specificerar hela systemet

Formella specifikationer

- Strict validering förutsätter en formell notation
- Blir i praktiken en sorts programmering på hög nivå:
  - Z, VDM, CSP, Petri-nät
  - eller algebra, en sorts matematik:
    - Larch, OBJ, Lotos
- Kommunikationsprotokoll, kryptering, kompression
  - eller en referensimplementation, BT, webRTC
- men inte för användarinteraktion …



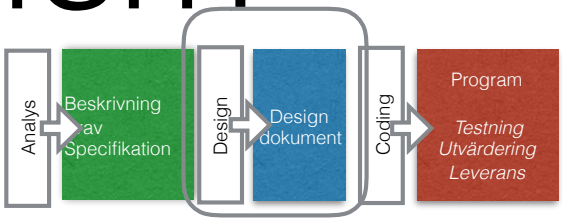
2 Design



- Ritning över hela systemet !
- *Tillräckligt detaljerad för att skriva all kod.*
- Systemstruktur (High-level design):
  - delsystem, kommunikation, lagring, kontroll
- Nedbrytning i komponenter (Detailed design):
  - delsystem, paket, klasser, metoder
  - API, databas schema, protokoll, interface
- Strukturera: High-level, Detailed Design, (Formal spec)

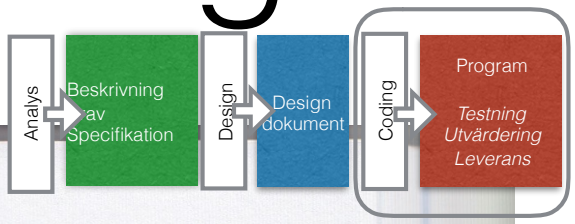
Jfr Krutchen 4+1 views of Documentation

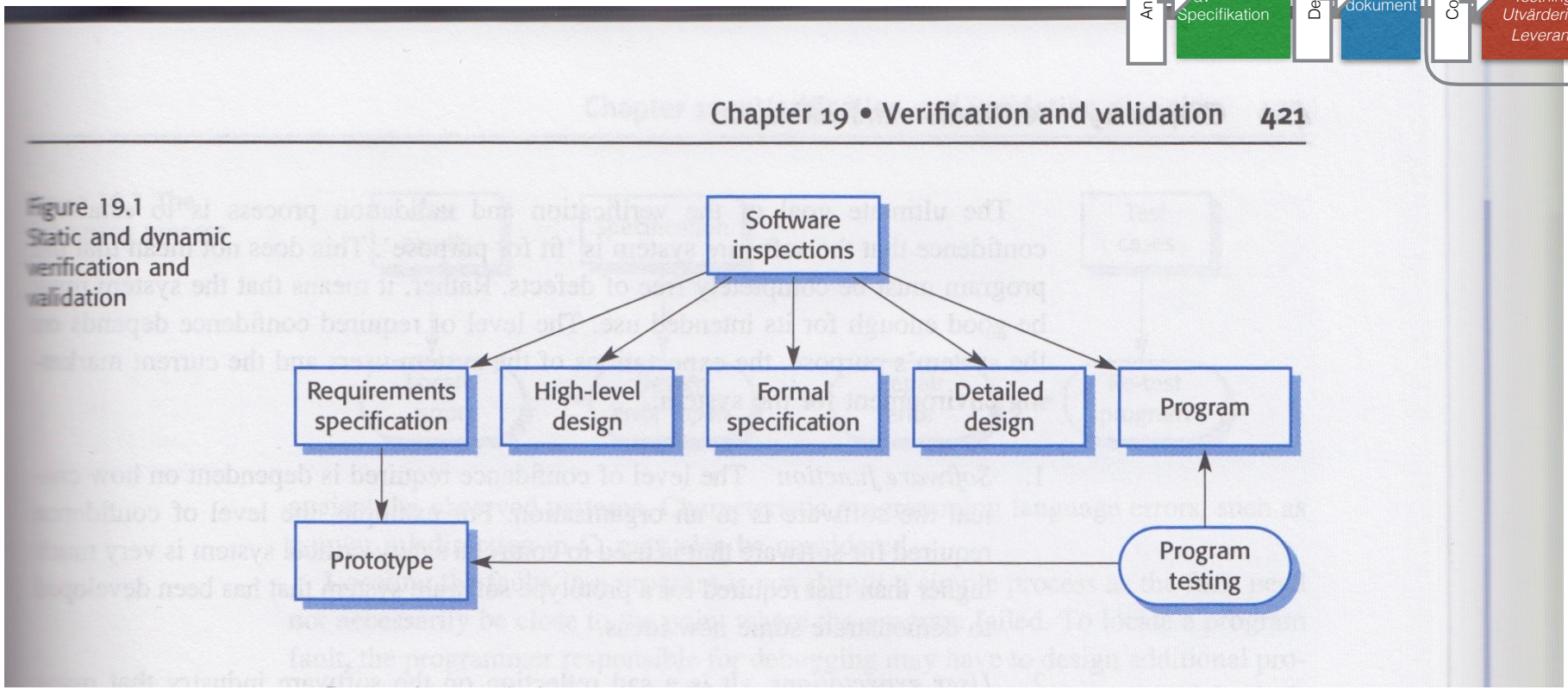
Design formalism



- Driva nästa steg - Validering?
  - konsisitent, fullständigt, stämmer med specifikationerna.
- Architectural Description Languages - försök finns men inga generella praktiskt användbara. Delproblem:
  - Tillståndsmaskiner - dödlägesanalys etc.
- UML - halvformellt.
- OO-modellering passar med OO språk

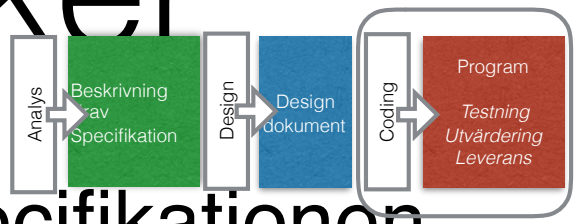
Kodning - testning





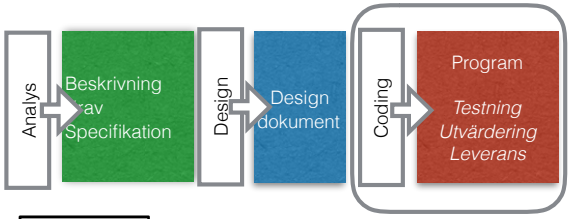
- Software processes säger oftast inte så mycket om själva programmeringen.

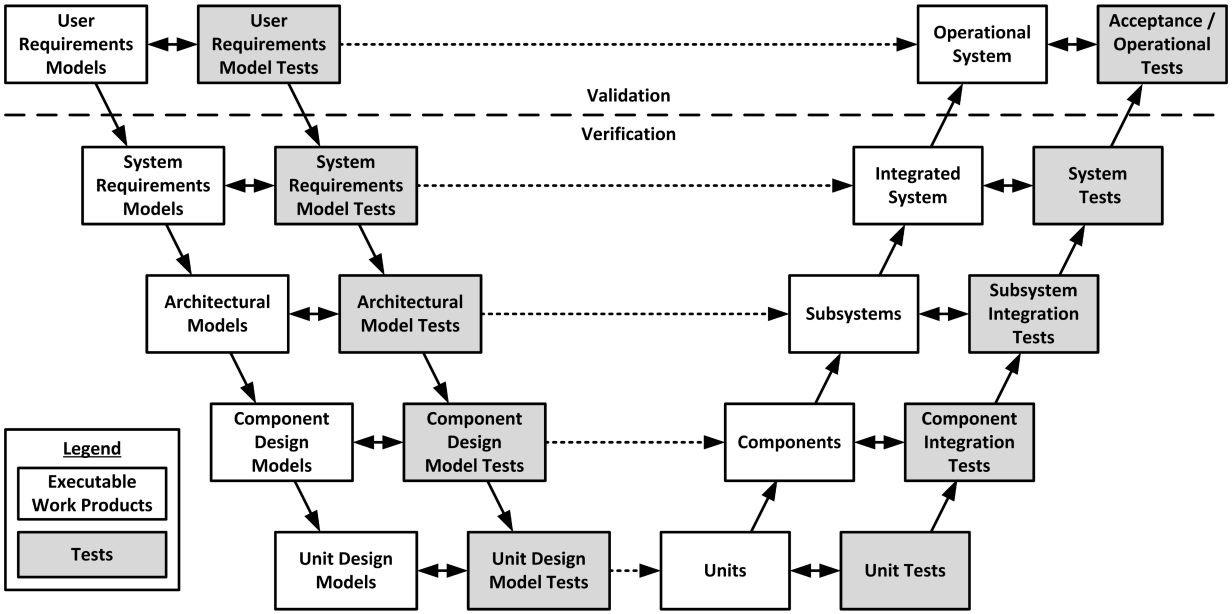
Testningstekniker



- Verifiering - uppfyller systemet specifikationen
  - Black-box - test mot specen
  - White-box - tillgång till koden - täcka all kod
- Validation
  - Funktionalitet - utvärdering av kund
  - Systemkrav, prestanda, minnesåtgång, robusthet, mot t ex felaktig input, överbelastning

V-modellen

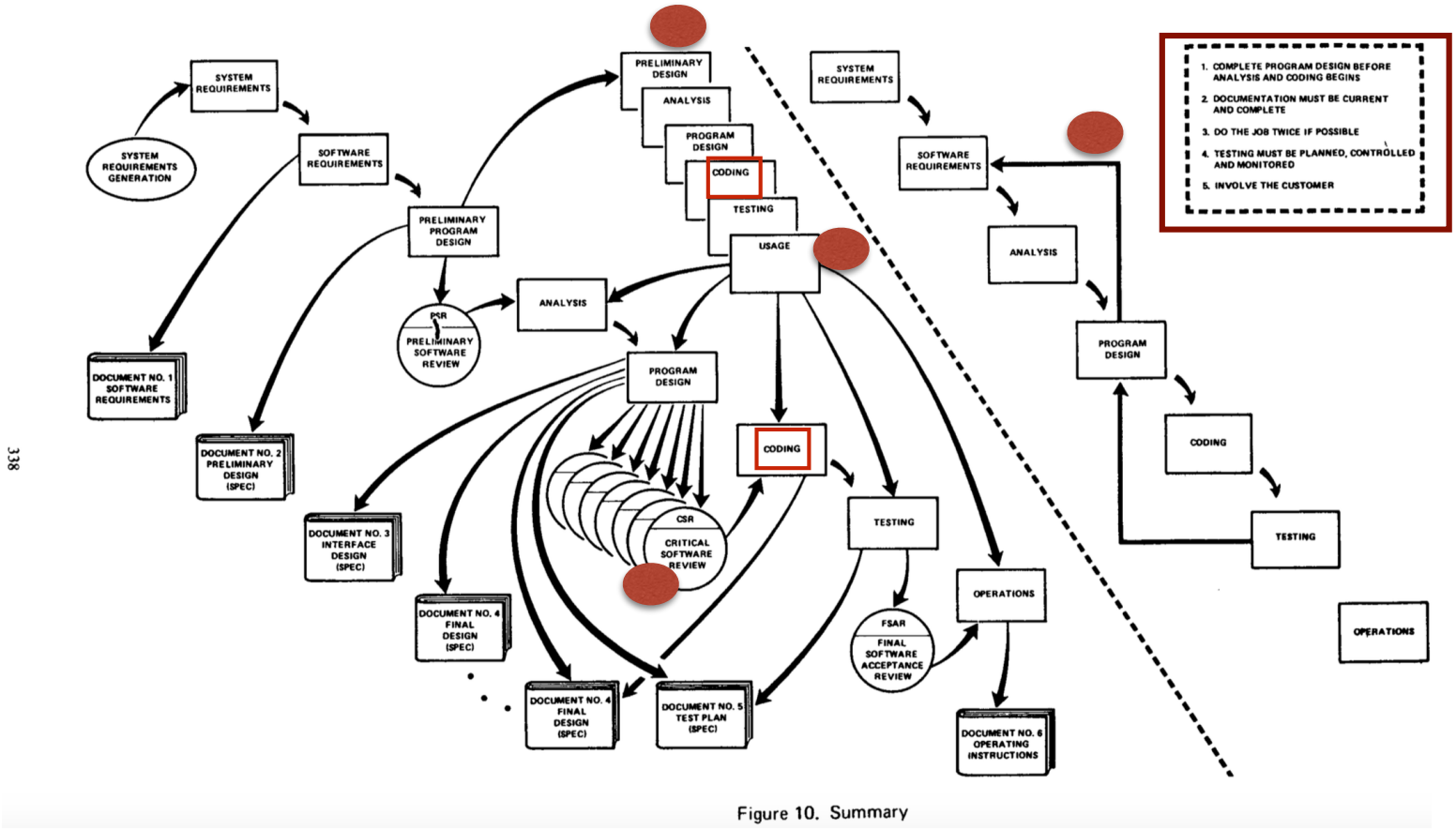




Från SEI.s hemsida

- V-modellen kommer ursprungligen från Systems Engineering
- Syftet är att bygga modeller (t ex matematiska modeller av ett dynamiskt system)uppifrån-och-ner som successivt verifieras.
- I SE har den kommit att användas för beskriva nerbrytning i komponenter.

Kritik av vattenfallsmodellen



338

Figure 10. Summary

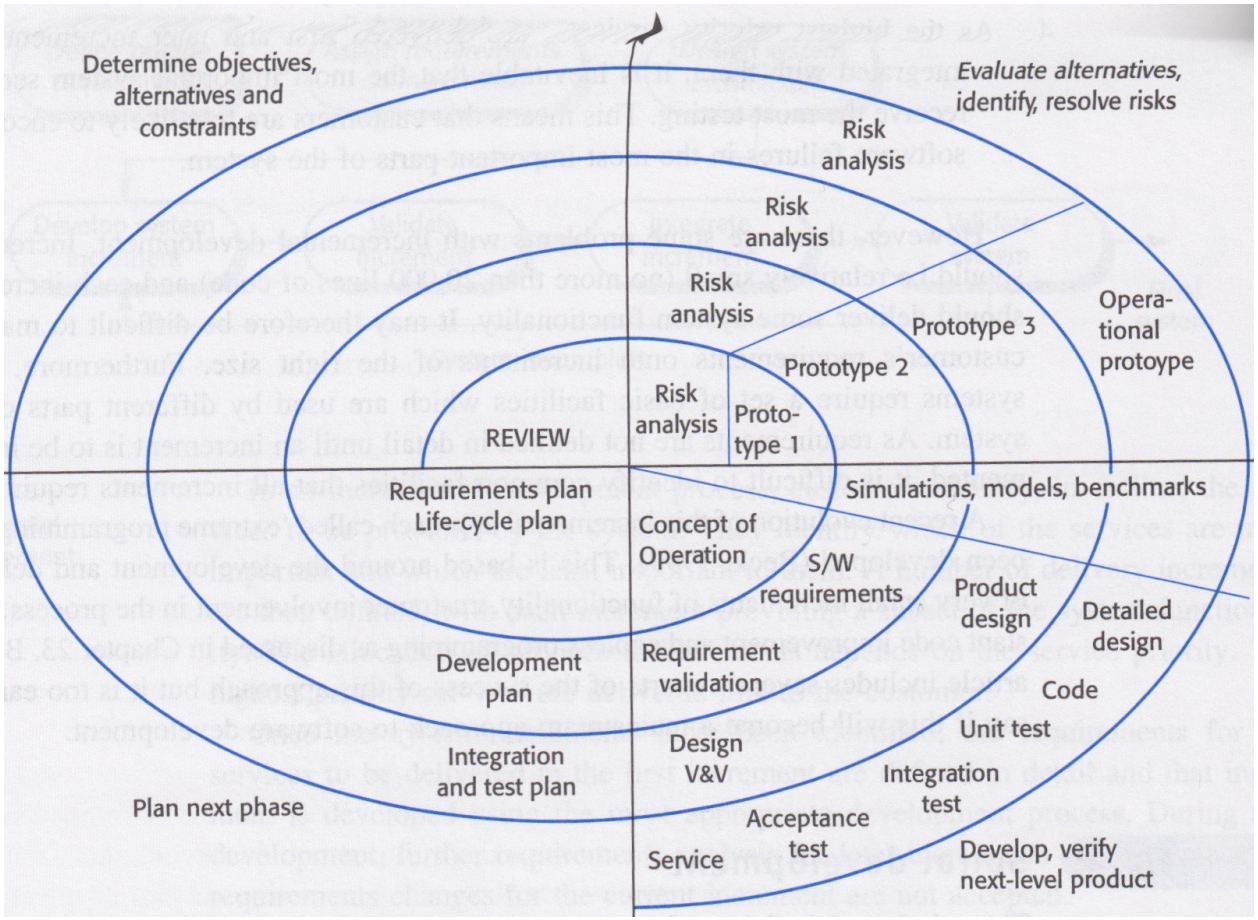
- Redan Royce artikel (1970) menar att man inte kan arbeta sekventiellt, problem slår ofta flera nivåer bakåt i kedjan.
- Han föreslår en rad tillägg.

Varianter av vattenfall

- Grundproblem - man kan inte göra hela designen av ett stort system utan återkoppling.

- **Inkrementell utveckling.**
  - Utveckling i delar där varje del följer modellen

Spiralmodellen



Berry Boehm Spiral modell, från I Sommerville

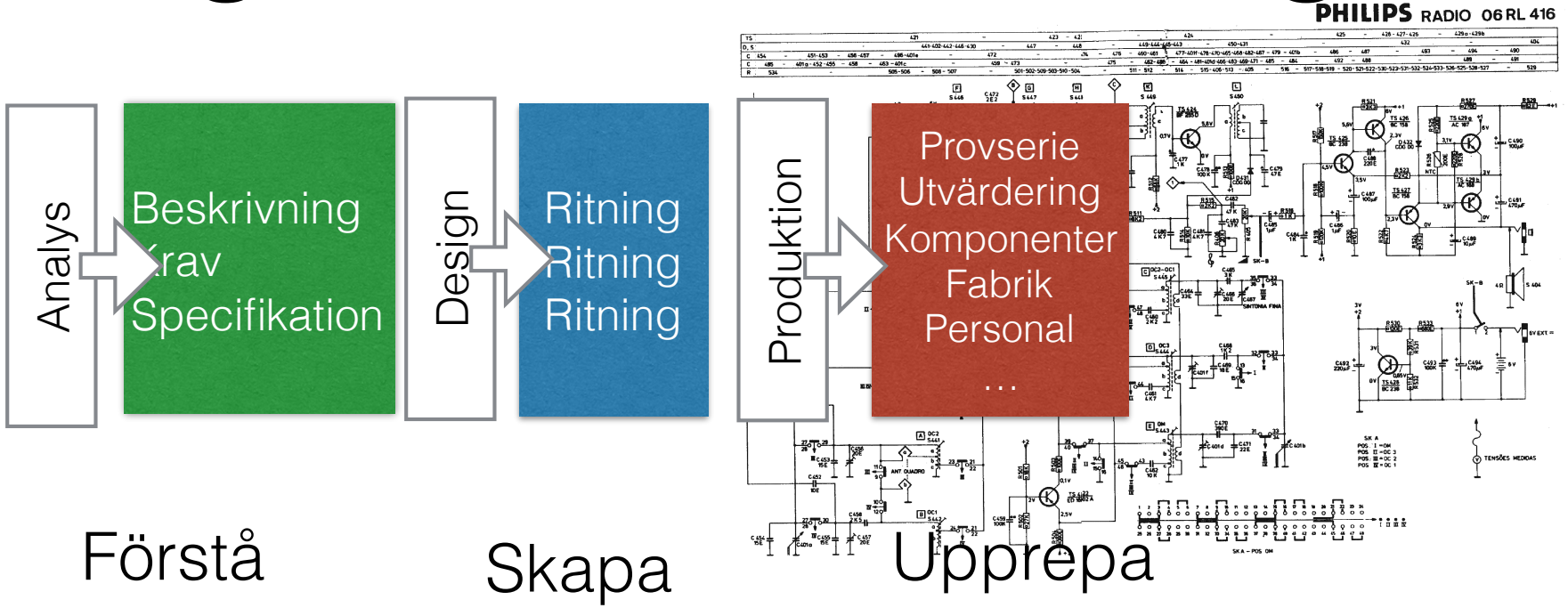
Extremvarianten: Cleanroom

Milles et. al. 1987

- Incremental - specification and validation in parts, defined early in the process.
- **Formal specification** - State-transition model, system response to stimuli.
- Structured, **restricted programming** - only parts of a language is used.
- **Transformation** - Program is developed as transformations of the specification. Validated in each step.
- **Static verification** - no testing during development.
- **Statistical testing** of system based on expected use.

The system is not executed during development.

Programvara - ritningar?

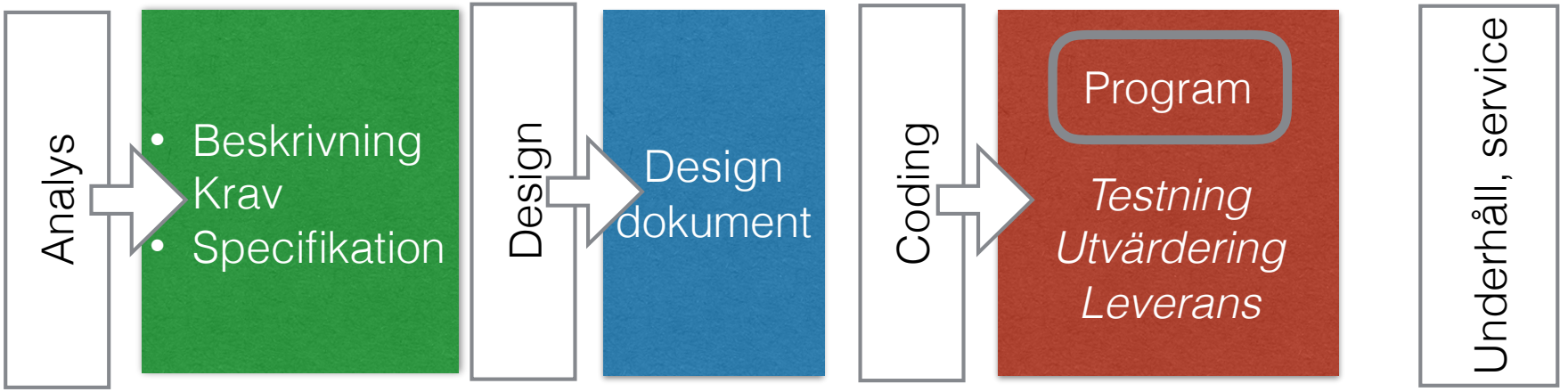


- Är programmering bara en (automatiserbar) process som översätter från (högnivå-) Design ?

Eller

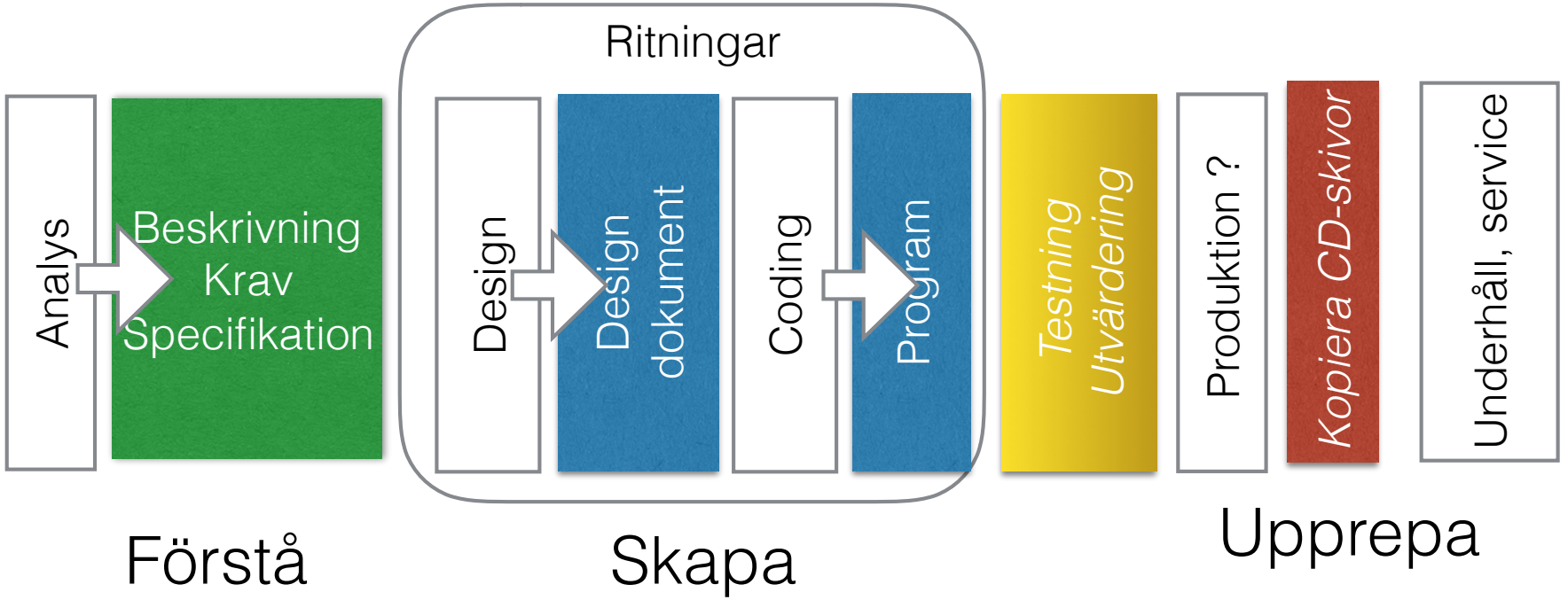
- Är programmering en del av Design med mer detaljer ?

Vattenfallsmodellen



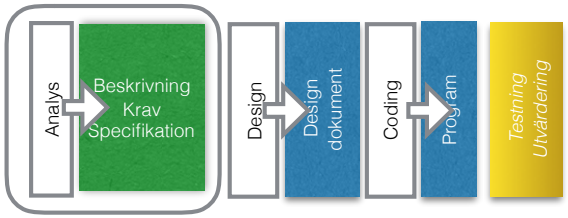
- Dokumenten tillräckliga för att driva nästa steg.
- Tolkning: Programmering =
  - en process som översätter från Design

En Agil tolkning



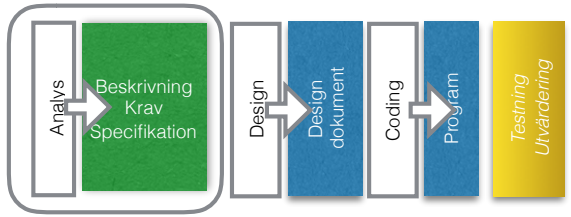
- Program motsvarar en elektronik-ritning - först nu är produkten fullständigt beskriven.
- (i XP blir Testerna också ”ritningar”)
- ”Produktion” blir trivial.

Vem bidrar till kraven?

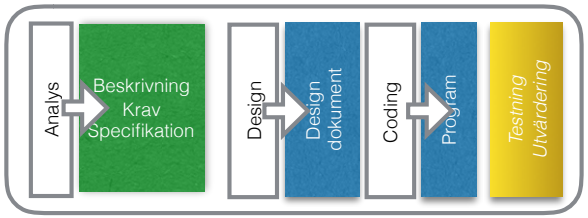


- Chef eller annan (IT-) organisation:
  - Fel vinkel - vet oftast inte i detalj.
- Intervjua användarna:
  - Svårt att föreställa sig, ofta förenklad bild
- Etnografiska studier
  - Observera hur användare gör, studiebesök, video för senare analys
- Problem:
  - Lång tid mellan kravinsamling och leverans - behovet ändras.
  - Systemet i sig påverkar - nya/andra krav nu när det används.

Prototyping

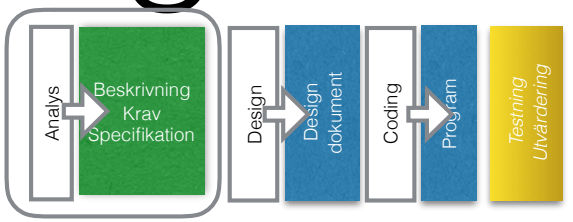


- *Det är en del av problemet att förstå problemet*
- Förenklad implementation för att få användarnas synpunkter, ”det var inte så jag menade”.
  - Rita, Bildspel, fake-program med bara GUI
  - Tänkt att kastas ersättas när man fått erfarenhet.



- **Evolutionary prototype** - så som man tänker i XP

Participatory Design



- Evolutionary prototyping med ”ständig” medverkan av användare.
- Användare bidrar med både problem och lösningar.
- Kombinerat med snabb återkoppling kan det fungera väldigt bra.
- Går utmärkt att kombinera med XP

Erfarenheter

- Att bygga stora IT-system är fortfarande svårt
- Många större IT-projekt havererar med stora kostnader.
  - Nordea - 5 Miljarder
  - SEB - 2 Miljarder
  - Försäkringskassan - 400Miljoner
  - Försvaret Prio - 2,4 Miljarder
  - GB: EHR (Journalsystem) - 12 Miljarder Pund (!)
- Gemensam Vård Data (GVD) - 1,6Miljarder -> NPÖ

Ett annorlunda exempel Polisens Utrednings Stöd

- System 1 - PUST-Java
- Agila metoder:
  - Avgränsad del (cykelstöld)
  - Begränsad lansering - prov i mindre grupp
- Iterativt, fler brott, senare bredare utrullning.
- Alla nöjda:
  - Utredningstider gick ner, effektivitet ökade med 85%
  - Jusititeminstern i TV: ”Nu blir det fler poliser på gatorna”

Polisens it-avdelning: Inte standardsystem måste byggas om.

- För dyrt med fortsatt underhåll av egenutvecklat system
- System 2 PUST Sibel
  - Baseras på standardsystem
    - Siebel ett ärendehanteringssystem från Oracle.
  - Total ny implementation med bibehållen funktionalitet (användarna märker inget)

PUST/Siebel resultat

- Konventionell utveckling, ideala förutsättningar:
  - PUST/Java som spec, eller proto
- Svårt att hitta kompetenta utvecklare, förseningar.
- Resultat:
  - extremt långsamt i drift
  - kompromisser i utformningen
  - förändringar i Siebel - ej längre standard!
- PUST/Siebel stängdes ner av arbetsmiljöskäl
  - Stressen på Poliserna blev för stor!
- Kostnad 300Mkr direkt, 20Gkr för samhället, Polisfacket

Exempel: Tunnel genom Hallandsåsen

- Tågtunnel - inte första gången i världshistorien
- Vad behövs:
  - Borras två hål, läggas räls, elinstallationer, anslutningar, en ny station. Stort men välkänt.
- Upphandling: så noggrann spec:
  - Detaljer, krav, tider för leverans, … 1997.
- Kraftbyggarna vann upphandlingen (billigaste budet)
  - Byggt mycket tunnlar till vattenkraft i Norrland.
  - Borrat i urberg - nu en grusås skall också gå.

Hallandsåsborr fortfarande oanvänd

Publicerat måndag 7 april 2003 kl 11.24

BÅSTAD. För tio år sedan invigdes den specialbyggda jätteborren Hallborr som skulle borra de två järnvägstunnlarna genom Hallandsåsen. Men borren, som kostade byggföretaget Kraftbyggarna 130 miljoner kronor, kom bara 18 meter och sen dess har den inte används.

- Så mycket för den planeringen!
- (Tunneln blev 18 år försenad)

Varför gör man då så här?

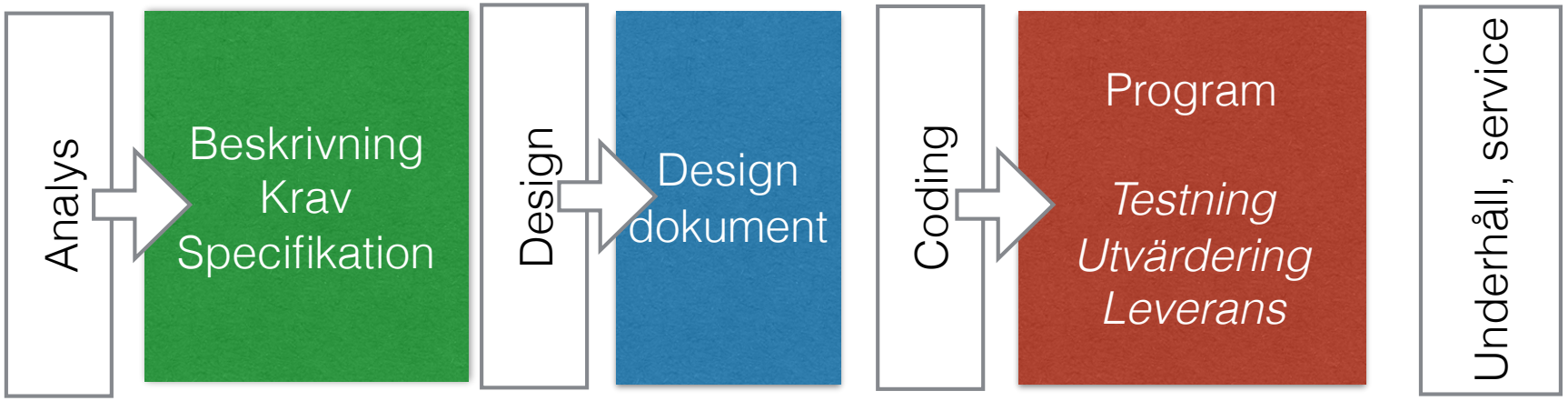
- Tradition - svårt att ändra.
- Ekonomiska, juridiska ramar:
  - Avtal mellan kund och leverantör
  - Baseras på beskrivning av vad systemet skall göra, specifikation.
- Sen är man fast:
  - Kunden vill inte ändra kostnaden
  - Leverantören vill inte ändra innehållet.
- LOU (Lagen om Offentlig Upphandling) gör det ännu värre.

Varför går det fel?

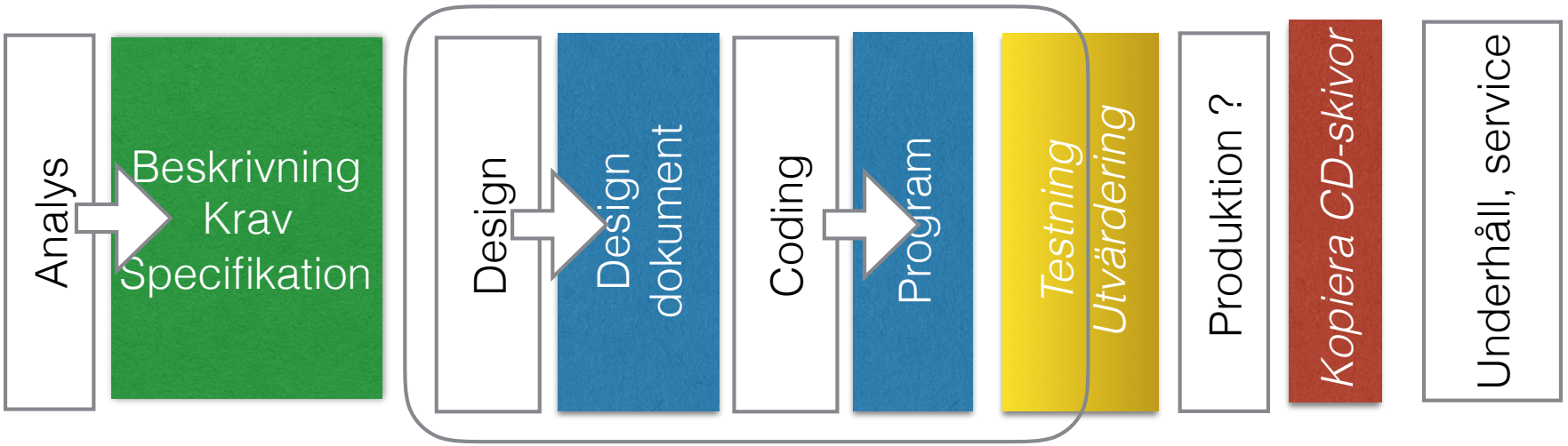
- Stora projekt är svårt, inte bara programvara
  - Svårare ju större, ju längre tid, flera parter
- Speciellt för programvara:
  - Behovet svårt att fånga
  - Behovet ändrar sig
  - Påverkar människors arbetsvillkor
- Specifikationer/Krav läggs fast för tidigt.
- Som kontrast - Agila metoder (XP)
  - Återkoppling, prova - ändra, förbättra.

Utvecklingsmodeller

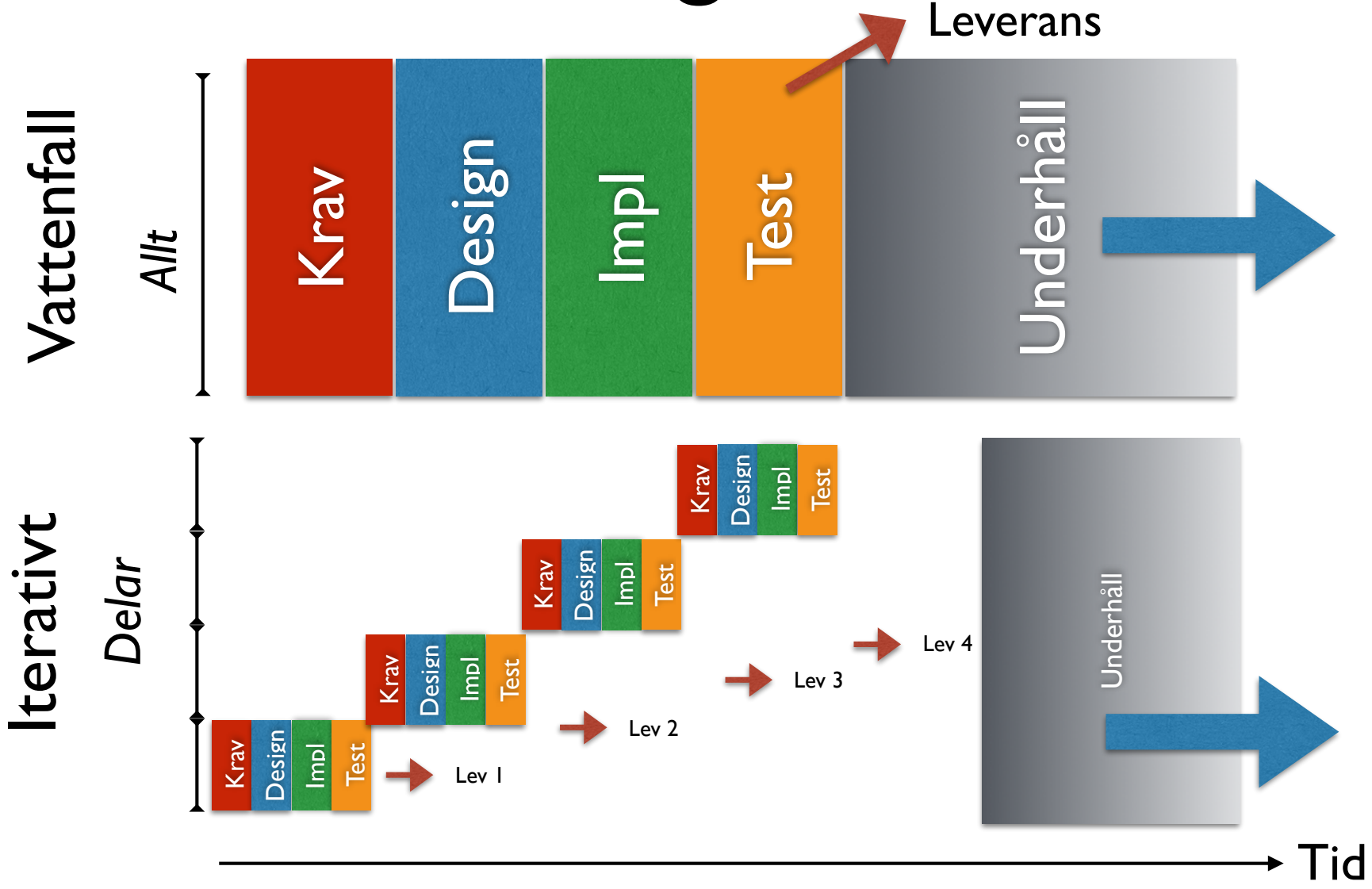
Vattenfallsmodellen, dokumentbaserade



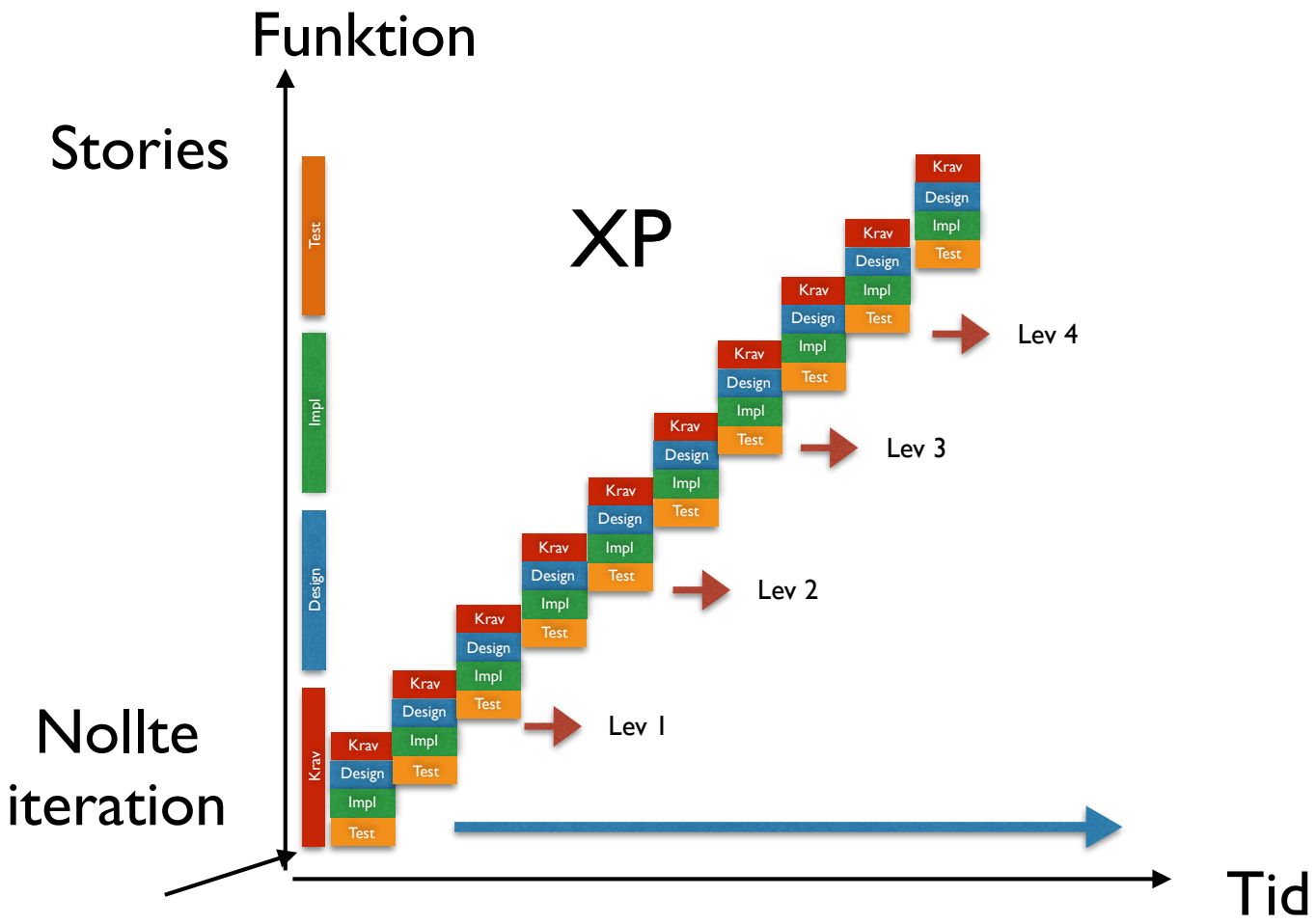
Agila - XP modellen



Faser i traditionella Utvecklingsmodeller



Faser i Agil utveckling



Läsanvisningar

- Software Engineering
  - Winston W. Royce: Managing the development of Large Software Systems, IEEE WESCON, August 1970 http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf
  - Pankaj Jalote: A Concise Introduction to Software Engineering, Springer
  - Ian Sommerville: Software Engineering, Addison Wesley (”uppslagsbok”)
- Debatt om fallerade projekt - PUST
  - PUST: http://computersweden.idg.se/2.2683/1.547944/haveriet-inifran-sa-gick-pust-fran-succe-till-fiasko
  - http://www.dn.se/debatt/sa-avslojar-du-it-projekten-som-riskerar-att-haverera/
  - http://computersweden.idg.se/2.2683/1.546751/annu-ett-fiasko-for-accenture