

# Utvecklingsmetodik

En rejäl crash course i pakethantering, testning och paketering

Dessutom: projektet!



# Dagens föreläsning

- Pakethantering
- Paketering
- Testning

# Metodik

But why?



# Mjukvaruutveckling är dyrt!

- Våra verktyg är dyra
- Våra miljöer är dyra
- Vi är dyra
- **Våra fel** är dyra

Felen vi gör resulterar i:

# Produktionsbortfall



# Tappad försäljning

## Biljettkaoset: Kunder uppmanades åka gratis

SJ Publicerad 6 okt 2017 kl 08.57



Många försökte förgäves köpa biljetter på Stockholms centralstation på fredagen. Men vid lunchtid ...

Foto: LEIF BRÄNNSTRÖM

# Olyckor



# Dödsfall





# Eller värre...



**Ja, men varför?**



# Kodapan

Skriver kod. That's it.

Ett kugghjul i ett maskineri:

- Skriver kod mekaniskt
- Saknar överblick
- Utan egentligt ansvar
- Våldigt utbytbar



# Cowboy-kodaren

Skriver kod. That's it.

En “fri själ”, men inte så bra:

- Ofta orutinerad
- Ostrukturerad
- Saknar kvalitetstänk
- Inte hållbar





# Programmeraren

Skriver bra kod. Programmerar.

En renässansmänniska:

- Skapande
- Konstnärlig
- Ensamt geni
- Inte hållbar i längden



# Utvecklaren

Skapar lösningar. Utvecklar.

En ingenjörstyp:

- Metodisk och strukturerad
- Grupporienterad
- Kvalitetstänkande
- Hållbar



# Hur blir vi ordentliga utvecklare?

Genom att arbeta strukturerat och rationellt!

- Tänk först – skriv kod sedan. Gör vi rätt saker?
- Kan vi återanvända tidigare skriven kod? Hur?
- Kan vi effektivt samarbeta kring vår mjukvara?
- Hur dokumenterar vi?
- Hur säkerställer vi att koden fungerar?

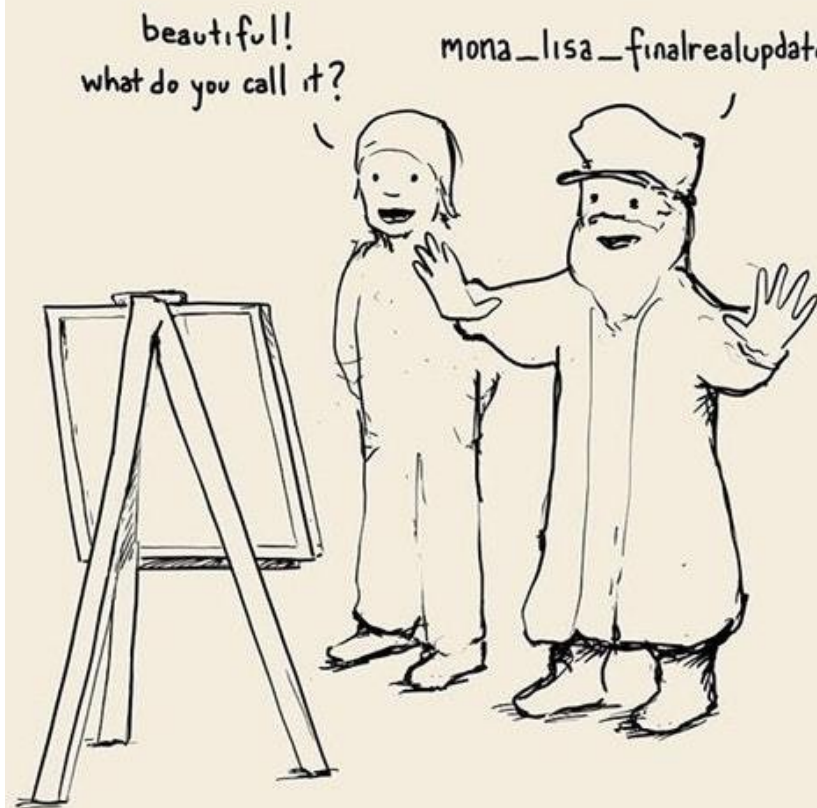


# Versionshantering



## Vårt problem

- Riktiga mjukvaruprojekt är stora
- Riktiga mjukvaruprojekt kommer att innebära flera – och parallella – releaser av samma mjukvara
- Hur hanterar vi detta på ett effektivt sätt?



# Vad är versionshantering?

Versionshantering är den metod med vilken vi kan hantera flera parallella versioner av en mjukvara

- Möjliggör samtidig utveckling av olika delar av mjukvaran
- Ger oss möjligheter att känna till – och dra nytta av – tidigare versioner av mjukvaran
- Möjliggör samtidig utveckling av nischade versioner eller utgåvor av en mjukvara

# Fördelarna med versionshantering

- EN SANNING – Vi kan alla vara överens om vilken kod som är den aktuella
- Samtidig utveckling
  - Gör det enklare att dela och återanvända kod
  - Olika team kan arbeta på samma projekt
- Historia och spårbarhet
  - Återgå till tidigare versioner av ett projekt
    - Förstörde din kursare din kod? Ingen fara, den finns kvar!
  - Underhåll flera olika releaser av samma fil, modul eller applikation
  - Vem gjorde vad? Nu kan vi veta!
- Minskade risker
  - Ha inte all kod på en enda dator
  - Låt inte vem som helst förändra koden

**Vad vill vi  
versionshantera?**



# Versionshanterar vi inte allt?

Nej. Viss data gör sig dåligt i ett repo.

I git hanterar vi detta genom att specificera vilka filer vi INTE vill hantera i filen `.gitignore`.

Ett problem som ni i era projekt måste lösa är hur ni hanterar känslig data och konfigurationsfiler. Tips: använd exempelkonfigurationsfiler och håll koll på känslig data utanför GitHub!

# Vad som versionshanteras

- Källkod, inklusive exempelvis tester
- Dokumentation
- Viss metadata kring projekt:
  - Git-relaterad data
  - Paket- och beroendeinformation, exempelvis:
    - *pom.xml* i Java-projekt
    - *gradle.yml* i Java- och Android-projekt
    - *package.json* i Javascript-projekt
    - *requirements.txt* i Python-projekt

# Vad som versionshanteras, del 2

- Viss metadata kring projekt:
  - Licensinformation
  - Git-relaterad data
    - .gitignore
    - .git
  - Paket- och beroendeinformation, exempelvis:
    - *pom.xml* i Java-projekt
    - *package.json* i Javascript-projekt
    - *requirements.txt* i Python-projekt



# Vad du INTE ska versionshantera!

- Känslig information
  - Användarnamn
  - Lösenord
  - API-nycklar
  - IP-adresser och URLer (127.0.0.1 och *example.com* är okej)
- Din IDE:s konfigurationsinformation
- Ditt operativsystems specialfiler, exempelvis *.DS\_Store* på Mac
- Tillfälliga filer, exempelvis:
  - Data som laddas in med jämna mellanrum
  - Sessionsfiler

# Vad du INTE ska versionshantera #2

- Beroenden och bibliotek, exempelvis:
  - jar-filer i Java- och Android-projekt
  - dll-filer i Windows-projekt
  - *node\_modules* i Javascript-projekt
  - *modules* i Python-projekt
- Binärfiler (om vi kan slippa det):
  - class-filer i Java- och Android-projekt
  - pyc-filer i Python-projekt
  - exe-filer i Windows-projekt

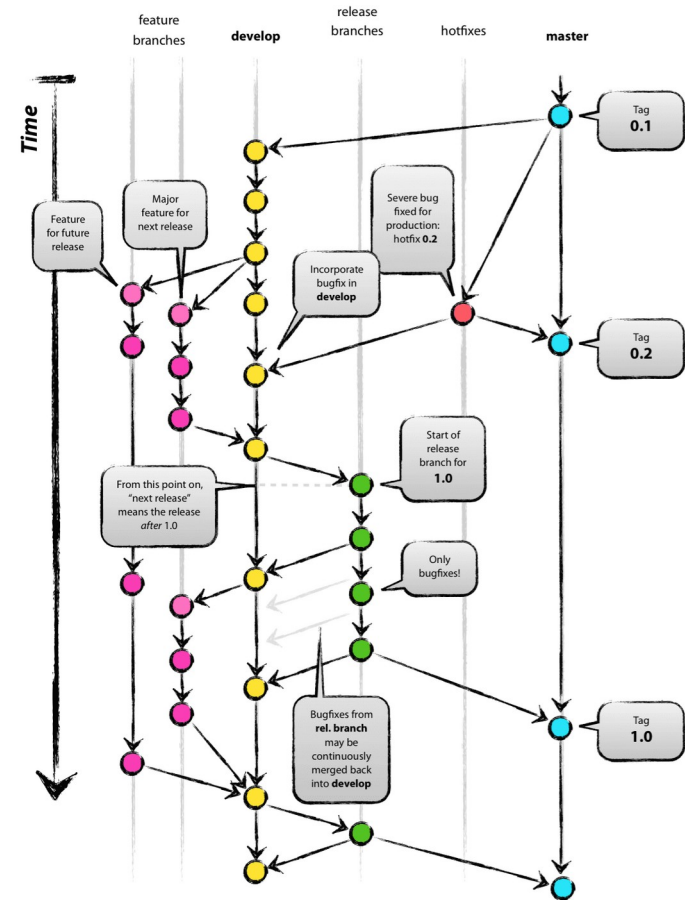
# Effektivt arbete med Git – Git Flow och Github Flow



# Git Flow

En vanlig modell för att arbeta med Git

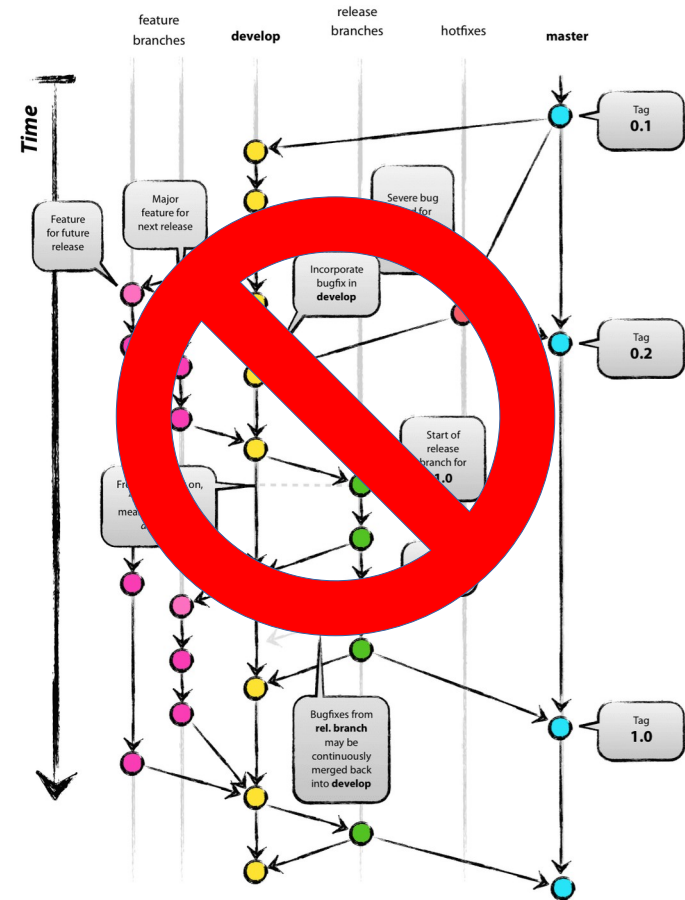
- Välansvänt och vältestat
- Använder mycket branching
- Väl anpassat för tigha utvecklingsteam
- Inte lika väl anpassat för större, löst organiserade team
  - Varje utvecklare måste kunna göra en push
- Negativt:
  - Långlivade grenar tenderar att leda till integrationsproblem
  - “Buskig” historik kan vara svår att följa



# GitHub Flow

En reaktion på Git Flow, avsett att

- Vara enklare att använda
- Mer avsett för CI/CD
- Negativt:
  - Master-branchen är inte garanterat körbar
  - Svår att arbeta med när du vill ha ett release-schema

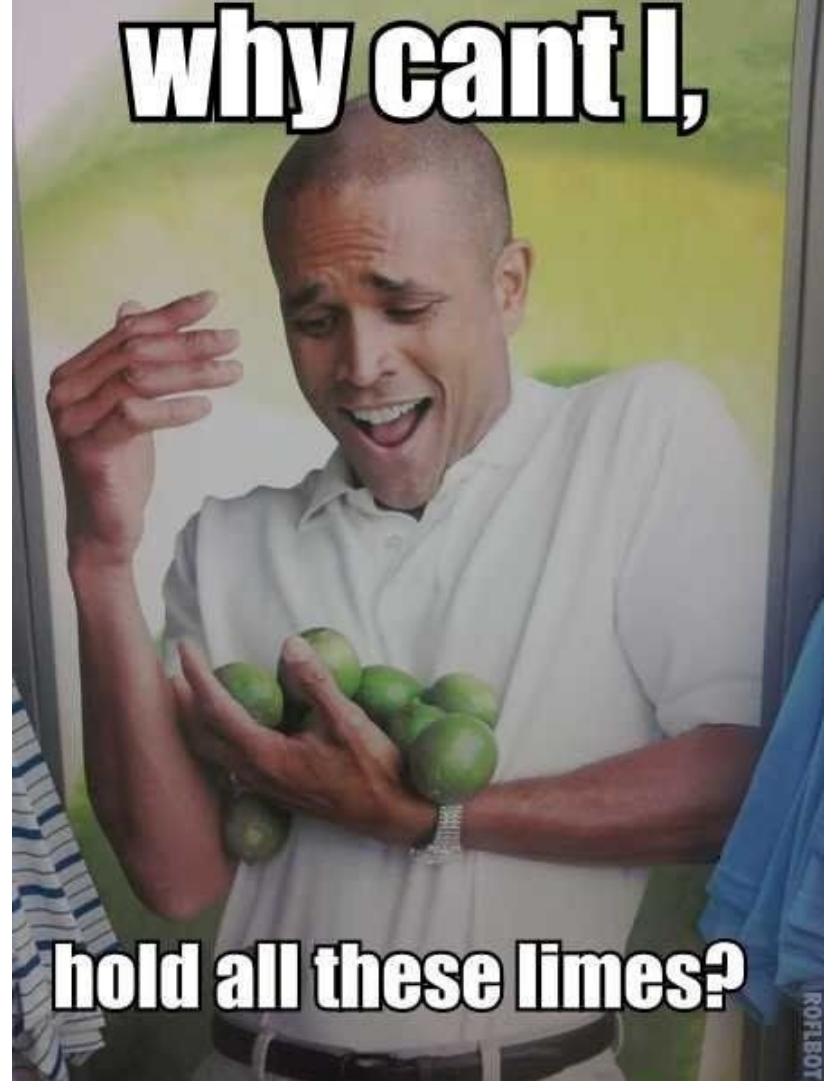


# Pakethantering



## Vårt problem

- Riktiga mjukvaruprojekt är stora
- Riktiga mjukvaruprojekt innehåller oftast externt skriven mjukvara
- Hur hanterar vi detta på ett effektivt sätt?



# Vad är pakethantering?

Modulbaserad mjukvara behöver kontrolleras – detta görs med fördel med en pakethanterare

- Återanvänd din (och andras) kod
- Separation of concerns
- Slipp ifrån jobbet med att dra ner paket manuellt

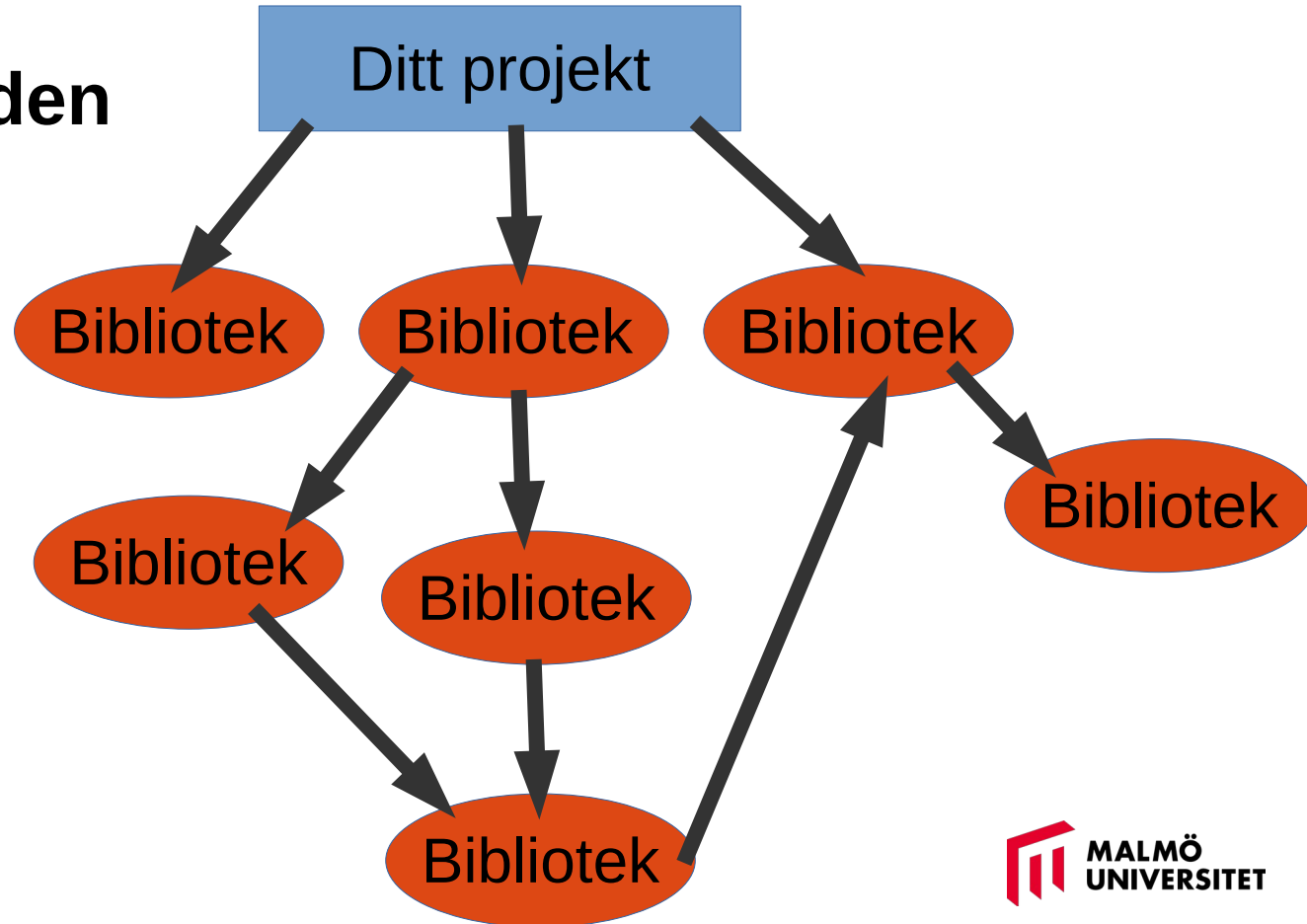




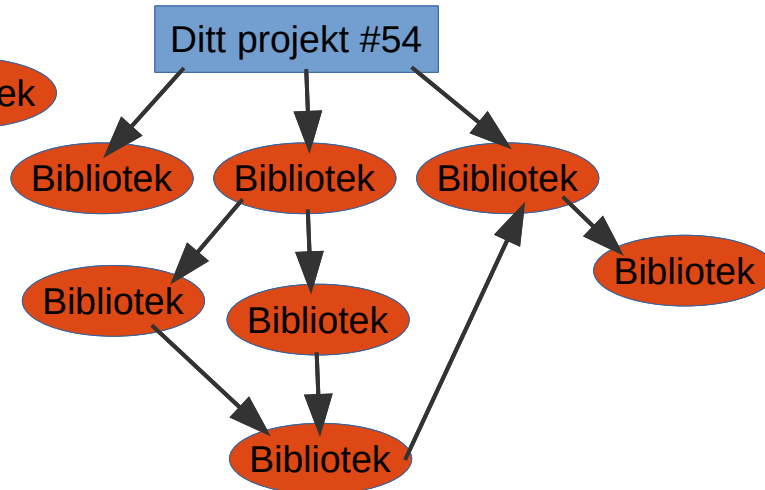
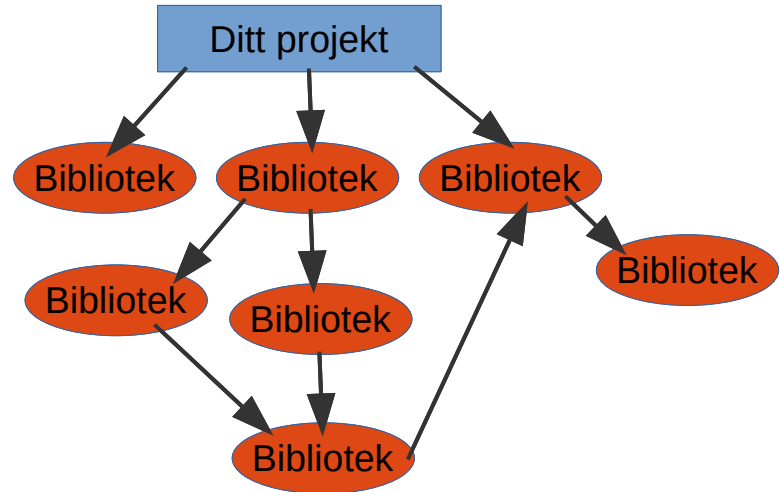
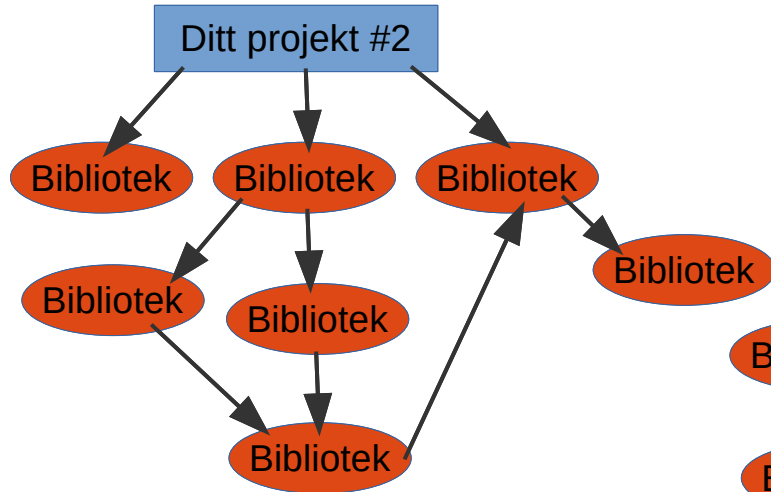
# Några viktiga termer

- **Paket:** En samling av filer som levereras tillsammans. Kan vara körbara filer, bibliotek eller andra resurser, så som media, typsnitt eller konfigurationsfiler. I JavaScript kallas dessa allmänt för **moduler**.
- **Pakethanterare:** En mjukvara som hanterar paket.
- **Repository** (eller **repo**): En plats där paket kan publiceras, sökas efter och laddas ner från.
- **Beroende:** Ett eller flera paket som behövs för att ett specifikt paket ska kunna användas.
- **Beroendekonflikt:** Ett tillstånd som uppstår då två paket beror på samma paket.
- **Version:** En specifik utgåva av ett specifikt paket. Genom att känna till versionsinformation kan en pakethanterare hålla koll på flera versioner av ett paket, vilket minskar risken för beroendekonflikter.

# Beroenden



# Beroenden



# Beroendehantering i mjukvaruprojekt

- Python: pip
- Java: Maven/Gradle
- PHP: Composer
- C#/.NET: NuGet
- **Javascript: npm**

# Varför beroendehantera i mjukvaruprojekt?

- Robustare projekt och mjukvara
- Enklare att starta nya projekt
- Enklare för nya utvecklare att börja arbeta
- Mindre kod i era kod-repositories

# Pakethanteraren npm

- De facto-standard för vettig beroendehantering i JavaScript
- Bygger på Node.js – npm (Node Package Manager) är dess pakethanterare
  - Bygger på en kodstandard som heter CommonJS
  - Går att använda till annat än just Node.js
- Sköter installation av moduler åt dig
  - Du behöver inte längre versionshantera och leverera alla beroenden – bara en fil som berättar vilka paket som krävs

# Pakethanteraren npm

- Hanterar beroenden i flera led
- Kan skilja mellan olika versioner av moduler
- Förenklar uppdatering av mjukvaran när underliggande moduler uppdateras
- Dessutom kan npm användas för att automatisera paketering, testning och mycket annat!

# Paketbeskrivning: package.json

```
{
  "name": "thin-red-line",
  "version": "0.5.7",
  "private": true,
  "scripts": {
    "start": "grunt docs && node ./bin/www",
    "test": "mocha --recursive test/unit_test -R dot",
    "system-test": "jasmine-node test/system_test"
  },
  "dependencies": {
    "bluebird": "~3.0.6",
    "body-parser": "~1.10.2",
    "xml2js": "^0.4.9"
  },
  "devDependencies": {
    "chai": "^3.0.0",
    "chai-as-promised": "5.1.*",
    "rewire": "~2.5.1"
  }
}
```



# Demo

Johan leker med npm

# npmjs.com

♥ No Prescribed Meaning

[npm Enterprise](#) [Products](#) [Solutions](#) [Resources](#) [Docs](#) [Support](#)

npm

🔍 Search packages

Search



npm is the package manager for javascript

## Popular libraries

lodash  
request  
chalk  
react  
express  
commander  
moment  
debug  
async  
prop-types  
bluebird  
react-dom

## Discover packages

IoT  
mobile  
front end  
backend  
robotics  
css  
testing  
cli  
documentation  
math  
coverage  
frameworks

## By the numbers

Packages

**976 532**

Downloads · Last Week

**7 743 270 304**

Downloads · Last Month

**41 102 284 383**

## Recently updated packages

### @controlla/controla-postinstall

Lightweight npm postinstall message

**ivansotelo** published 1.0.19 • 3 minutes ago

### @brd.com/deploy-it

BRD's node deployment tools. In simple terms, this module has some sensible defaults for deploying plan node apps and nuxt apps. These get put together in the `gitlab-functions`, and can be overridden by javascript files in the `deploy` directory. Any

# npmjs.com

- Npm:s repository för JavaScript-moduler
- Modulerna är fria att använda i dina projekt
- Använder **semantisk versionering** för att skilja mellan olika versioner av moduler

# Semantisk versionering

1 . 0 . 0 . b1

major minor micro qualifier

# Alternativ till npm

- Yarn – I princip det enda alternativet till npm. Snabbare, kraftfullare, men inte lika välanvänt.

# Paketering



## Vårt problem

- Riktiga mjukvaruprojekt är stora
- Riktiga mjukvaruprojekt behöver paketeras innan de levereras
- Hur hanterar vi detta på ett effektivt sätt?



# Vad är paketering?

Utvecklad mjukvara innehåller som regel kod och artefakter som är ointressanta för slutanvändaren. Dessa tar stor plats och kan dessutom vara säkerhetshål. En paketerare tar bort alla onödiga artefakter och levererar en färdig mjukvara.



# Begreppen “build” och “bundle”

- En build är den process som tar din utvecklingskod – källkod, grafiska element och annat – och slår samman dem till en enhet.
- En bundle är den artefakt som är resultatet av din build-process.

# Paketering med Webpack

- Webpack är ett av många – men ett av de mer använda – paketeringsverktygen för webbprojekt
- Skrivet i JavaScript
- Slår samman flera filer av samma typ till en (se demo)
  - Flera JavaScriptfiler blir en
  - Flera CSS-dokument blir ett, etc
- Genererar en uppsättning filer per HTML-dokument som ska användas
  - Sparar mycket laddningstid
- Klarar i grunden av JavaScript, men kan utökas med andra filtyper
- Hanteras med hjälp av konfigurationsfiler – precis som pakethanterarna

# Demo

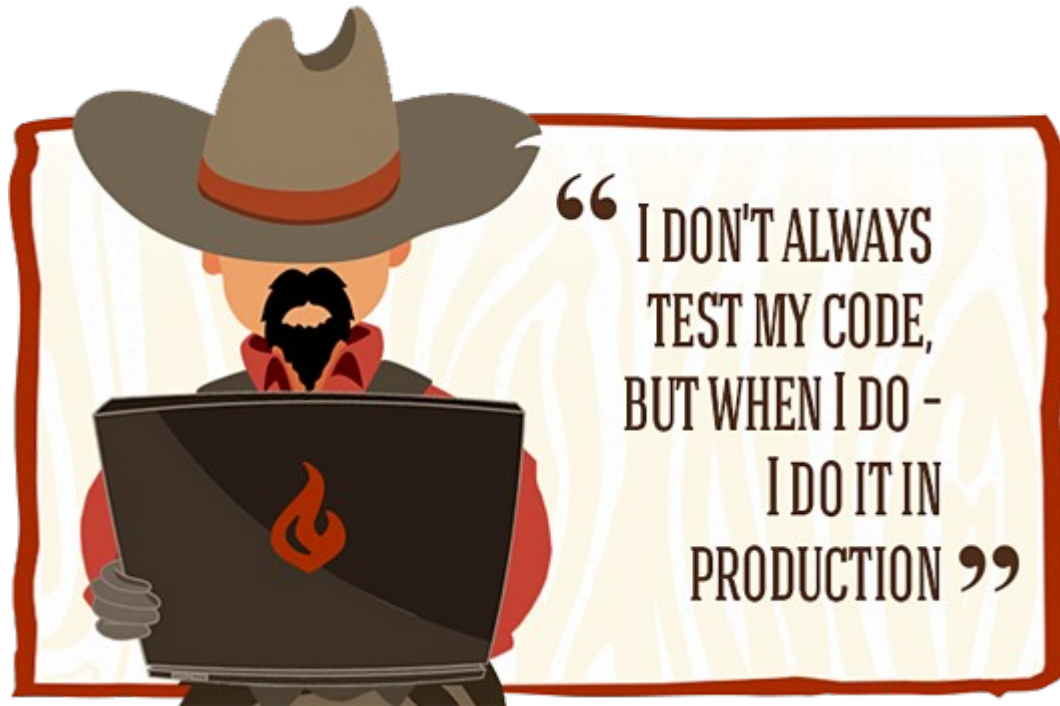
Johan leker med Webpack

# Alternativ till Webpack

- Browserify – Ett kraftfullt verktyg som dock kräver hjälp av andra verktyg, som Gulp, för att utföra saker som att minimera script
- Parcel – Ett verktyg som gör en stor sak av att det inte kräver någon konfiguration
- ...och så klart en massa andra

# Testning





# Vad är testning?

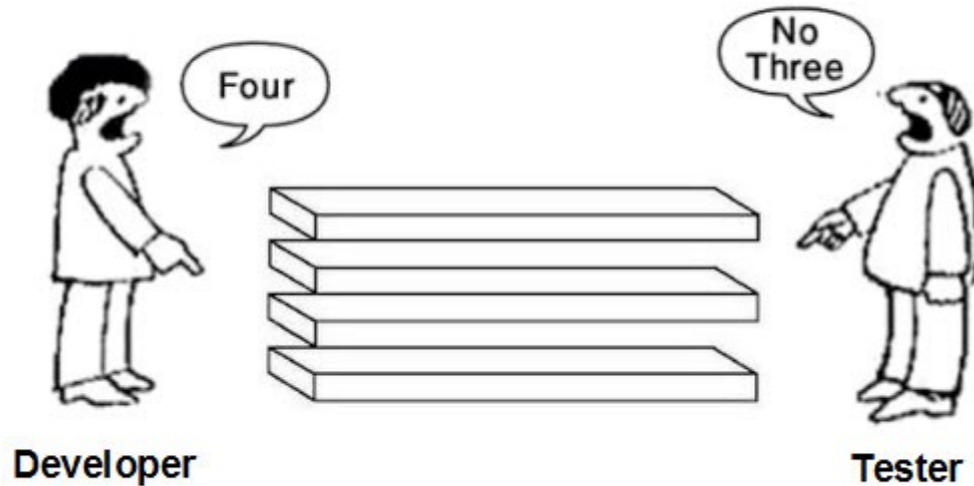
“**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.”

Enl. Wikipedia

- Validering → Bygger vi rätt saker?
- Verifikation → Bygger vi sakerna på rätt sätt?

# Varför testar vi våra applikationer?

Old but True Controversy



[www.softwaretestinggenius.com](http://www.softwaretestinggenius.com)



# Varför testar vi våra applikationer?

- Tekniska skäl
- Utvecklingsteamets skäl
- Ekonomiska skäl

# Tekniska skäl: Säkerställ funktionaliteten

Det här är validering och verifiering!

- Fungerar koden som den är tänkt att göra?
  - Hanteras indatan på rätt sätt?
  - Fungerar koden med felaktig indata?
  - Är koden feltolerant?
- Kommer programmet att fungera i produktion?
  - Matchar vår utvecklingsmiljö produktionsmiljön?
  - Fungerar all kod tillsammans?

My code working well on on my machine

*\* Deploys \**



# Utvecklingsteamets skäl: Förtroende

- En bra utvecklare kan visa att hens kod fungerar
- Bra tester säkerställer att koden fungerar
  - Tester är bra att ha under utvecklingen av en funktion
  - De är ännu bättre att ha när koden har levt en tid
- Tester kan användas som bas i diskussioner

# Utvecklingsteamets skäl: Historik och nya utvecklare

Tester är dokumentation → förenklar introduktion av nya utvecklare

- Väl utformade och beskrivna tester visar hur en klass eller funktion ska fungera
- BDD-tester (user stories-baserade tester) beskriver hur applikationen ska fungera
- Lösta buggar och fel visas med tester

# **Ekonomiska skäl: Driftstörningar är dyrare än utvecklingstid**

- Mjukvarutestning är dyrt
  - Längre utvecklingstid
  - Fler utvecklare/testare
  - Mer infrastruktur
- Fel i mjukvaran är dyrare
  - Nertid (se nästa slide)
  - Dålig PR/goodwill

# Ekonomiska skäl: Driftstörningar är dyrare än utvecklingstid

## Average Cost of Downtime

Even if your company survives a disaster, the costs are staggering:

- Brokerage \$6M - \$7M / hour
- Banking \$5 - \$6M / hour
- Credit Card \$2M - \$3M / hour
- Pay Per View \$1 - \$2M / hour (up to \$50M for fights)
- Airline Reservations \$1M / hour
- Home Shopping \$100K / hour
- Catalog Sales \$100K / hour
- Tele-ticket \$70K / hour
- Package Shipping \$30K / hour
- ATM Fees \$20K / hour

Average mean time to repair or recover: 4.0 hours



**Morpheus**, *How to manage app uptime like a boss* -

<https://www.morpheusdata.com/blog/2016-04-06-how-to-manage-app-uptime-like-a-boss>

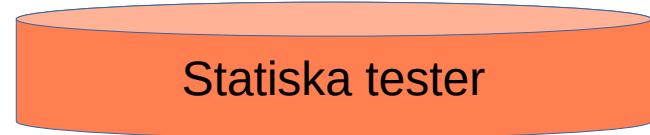
# Olika typer av testning

- Statisk testning
- Enhetstestning
- Integrationstestning
- End-to-end/acceptanstest



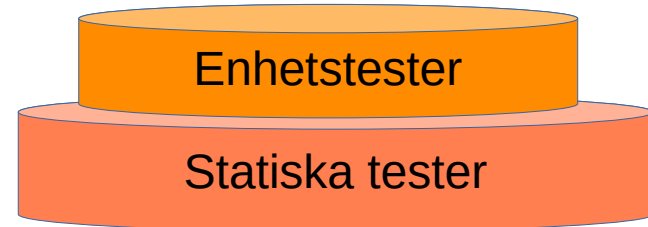
# Statisk testning

- Kräver ingen körning av mjukvaran
- Innefattar analys av:
  - Krav
  - Designdokument
  - Kod
- Kan göras både manuellt och med verktyg



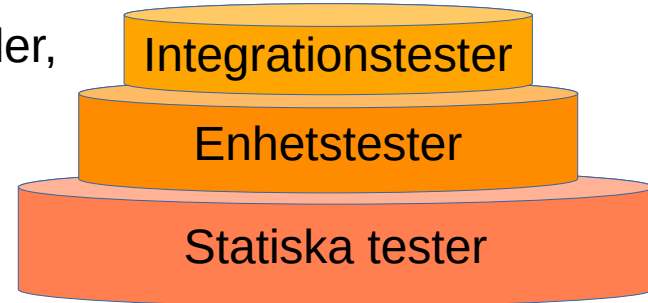
# Enhetstestning

- Testning av små delar – enheter – av koden, exempelvis klasser eller funktioner
- Körs ofta – så fort koden ändras
- Testfallen bör definieras innan koden skrivs
- Utförs med verktyg, oftast automatiskt
- Kan ses som specifikation och dokumentation



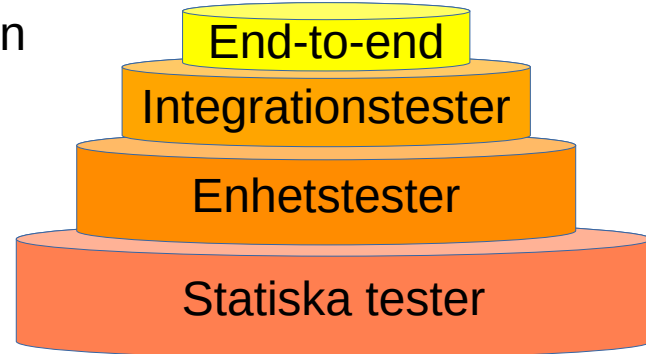
# Integrationstestning

- Tester som involverar flera enheter av koden – exempelvis två moduler, klasser eller funktioner
- Görs för att säkerställa att enheterna fungerar tillsammans
- Testerna körs ofta som en del av förberedelserna inför en release – projekt tenderar att använda specifika verktyg för detta



# End-to-end-testning

- Testar användarflöden i mjukvaran
- Heltäckande och testar många funktionaliteter samtidigt
- Körs inför leveranser
- Görs påfallande ofta av vanliga människor, men kan även göras med hjälp av automatiserade verktyg

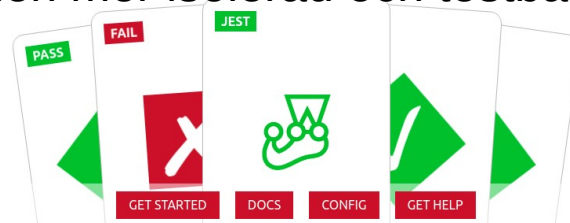
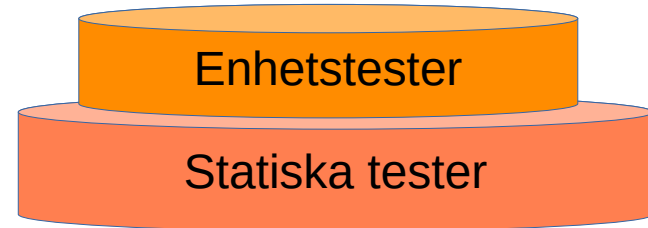


# Testning i JavaScript/HTML

- Lintning för att säkerställa att vår kod och våra dokument är välformade:
  - ESLint eller JSLint för JavaScript
  - HTML Tidy för HTML
- Enhetstester
  - Jest
- End-to-end-tester
  - Selenium

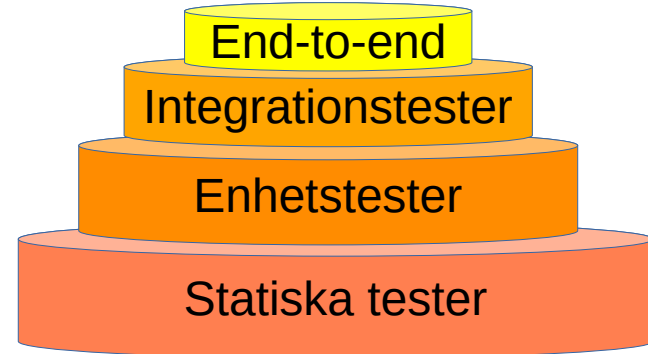
# Enhetstestning med Jest

- Bygger på samma tankesätt som ni sett i Junit (Java) och i unittest (Python)
- Testkoden lever separat från produktionskoden och kan köras direkt av npm
- Delar av koden kan ersättas med mock-objekt för att göra produktionskoden mer isolerad och testbar under testningen



# End-to-end-testning med Selenium

- Ett kraftfullt verktyg för att utföra frontend-tester i webbläsare
- Kan även användas för att automatisera arbetsflöden som kräver fysisk användning av webbläsaren
- Selenium körs med *web drivers*, som kan driva flera olika webbläsare under testningen
- Kan användas i flera olika språk och miljöer, men vi nöjer oss med att testa med JavaScript och Node



# Demo

Johan leker med Jest och Selenium



**Till sist**



# Glöm inte att programmering är kul!

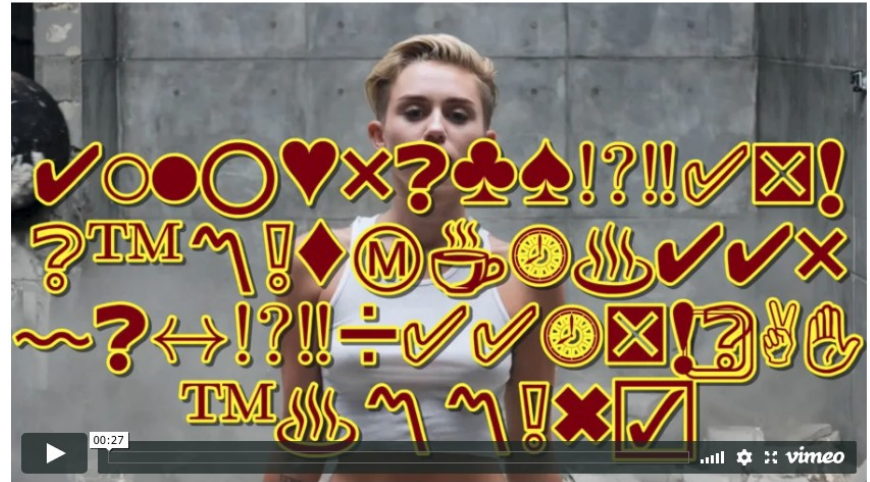
## Outcognito Mode

Outcognito Mode is a Chrome extension that publicly tweets every website you visit and everything you type. By [Harold Cooper](#).



## The Emoji Subtitle Creator

The Emoji Subtitle Creator by [Ross Goodwin](#) and [Seth Kranzler](#) automatically translates normal subtitles into ascii symbols.



# Det är okej att vara cowboy ibland



# What about projektet?



# Projektinformation

- Projektet görs i grupper om **tre** personer
  - Om ni inte hittar nog med folk, kommer vi att slå samman grupperna åt er
- Ni ska bygga en webbapplikation i ett valfritt Javascript/Typescript-baserat ramverk (exempelvis React, Vue eller Angular)
- Ni ska kunna motivera ert val av ramverk genom att jämföra ert valda ramverk med två andra ramverk
  - Exempel: Ni har valt Vue – varför valde ni inte Angular eller React?

# Projektinformation

- Webbapplikationen ska använda sig av *minst* ett externt webb-API, som inte tidigare använts i kursen
  - Anrop till det externa webb-API:t ska göras med hjälp av ramverket lämpliga funktioner för webbanrop
- Webbapplikationen ska antingen göra det möjligt för en användare att spara data, eller spara data utan hjälp från användaren. Datan som sparas ska ha en relation till det externa webb-API:t. Exempel på detta:
  - Ett filmbibliotek som sparar länkar till OMDb
  - En väderapplikation som sparar de senaste sökningarna hos SMHI

# Projektinformation

- Webbapplikationen ska ha ett grafiskt gränssnitt. Det är viktigt att ni sätter fokus på en god användarupplevelse på flera typer av enheter (mobil, desktop, etc)
- Ni ska använda er av Git och GitHub för att versionshantera ert projekt
  - Det är viktigt att ni upprätthåller god Git-disciplin och använder en rimlig branchningsmodell, skriver kvalitativa commit-meddelanden och inte versionshanterar felaktiga filer
  - Ert användande av Git är betygsgrundande
- Ert projekt ska dokumenteras i en README-fil, som innehåller en beskrivning av ert projekt, samt resonemanget kring ert valda ramverk jämfört med andra

# Projekthandledning

- Vi erbjuder projekthandledning. Dessa fungerar som laborationerna, med följande skillnader:
  - Ni kommer dit med hela gruppen
  - Ni har förberett frågor kring era problem innan vi ses
- Se schema för handledningstider



# Redovisning

- Varje projektgrupp ska under cirka tio + fem minuter presentera sitt projekt och svara på frågor – ungefär som i webbtjänster-kursen. Ni ska utnyttja de tio minuterna så optimalt som möjligt, så presentationen kan inte vara för kort. Förbered er genom att öva innan presentationen
- Följande frågor ska besvaras:
  - Introduktion till ert projekt – varför har ni byggt just detta?
  - Vilket ramverk har ni valt? Varför valde ni just det?
  - Vilket/vilka externa webb-API:er använder ni?
  - Vad har ni lärt er under projektet? Hur gick arbetet?
- Dessutom ska ni demonstrera er applikation