



# FOSTERING AN EVOLVING ARCHITECTURE IN THE AGILE WORLD

jPoint





# THE PORT OF ROTTERDAM

---

A black and white photograph of a middle-aged man with short hair, wearing a dark suit jacket over a light-colored shirt. He is smiling broadly and looking towards the camera. He is positioned behind a wooden desk. On the desk, there is a computer monitor displaying some text or graphics, a keyboard, and a small potted plant. The background is slightly blurred, showing what appears to be an office environment with other desks and equipment.

# HARBOUR MASTER MANAGEMENT INFORMATION SYSTEM

# HaMIS

# SO WHAT IS HaMIS ?

Interactive map

Security

Admittance policies

Patrol vessels (RPA)

Vessel inspections

Dangerous cargo

Crisis control

Cargo checklists

Harbour communication

And more...



450+ USERS

# The history of HaMIS

- IVS, a console based application
- Hardware no longer available:
  - Craigslist and ebay
  - Thinking about a replacement



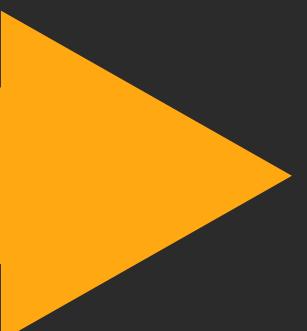
2004    2006    2008    2010    2012    2014    2016    PRESENT

# The history of HaMIS

- Government procurement for 'HaMIS'



2004    2006    2008    2010    2012    2014    2016    PRESENT

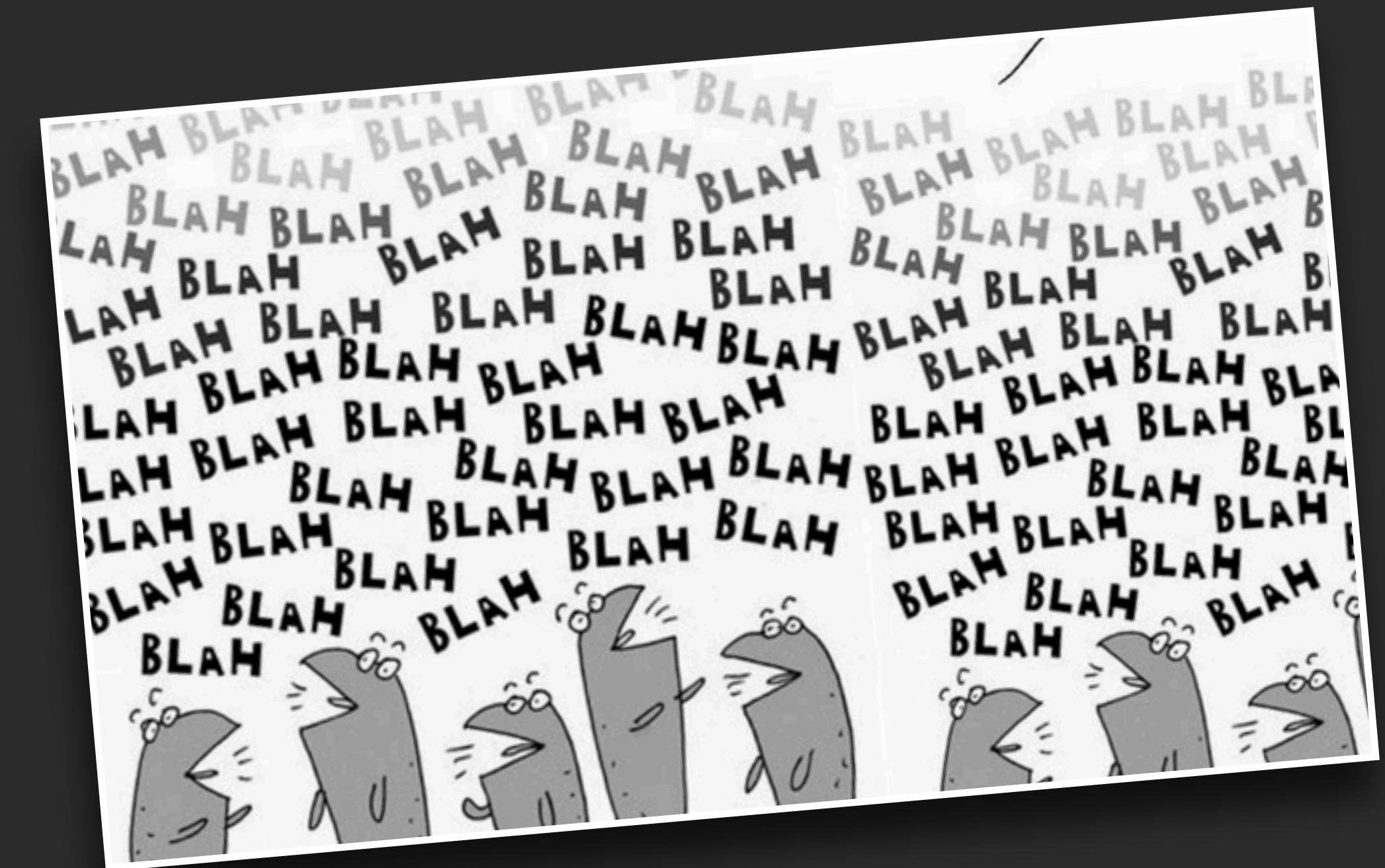


# The history of HaMIS

- 15+ architects
- 1 developer
- IBM WebSphere Partner Gateway

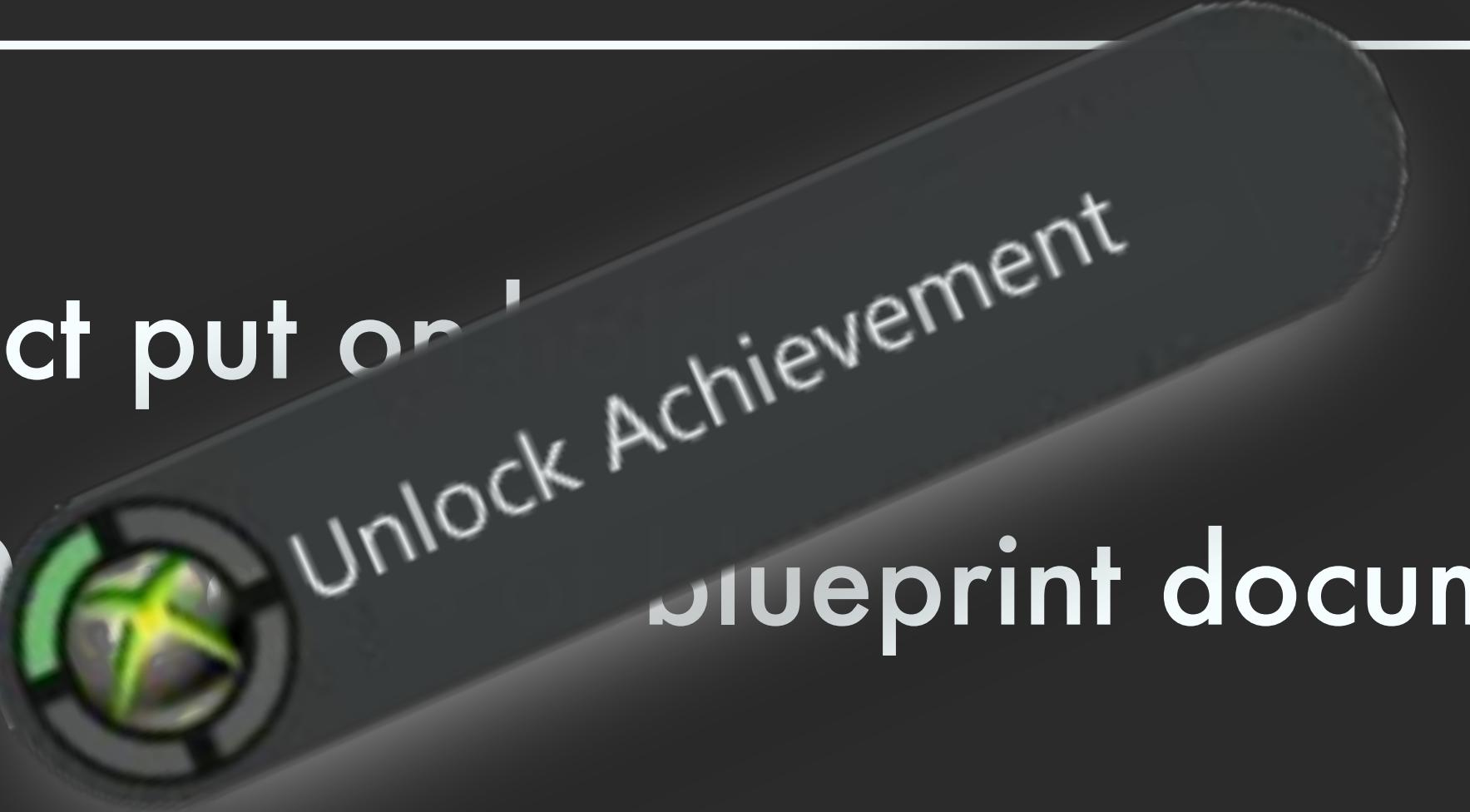


2004    2006    2008    2010    2012    2014    2016    PRESENT



# The history of HaMIS

- Project put on hold
- 100 Blueprint documents



# The history of HaMIS

- Ditch all the architects, hire developers
- A working application, live in production



2004    2006    2008    2010    2012    2014    2016    PRESENT

↔ Sprint 1

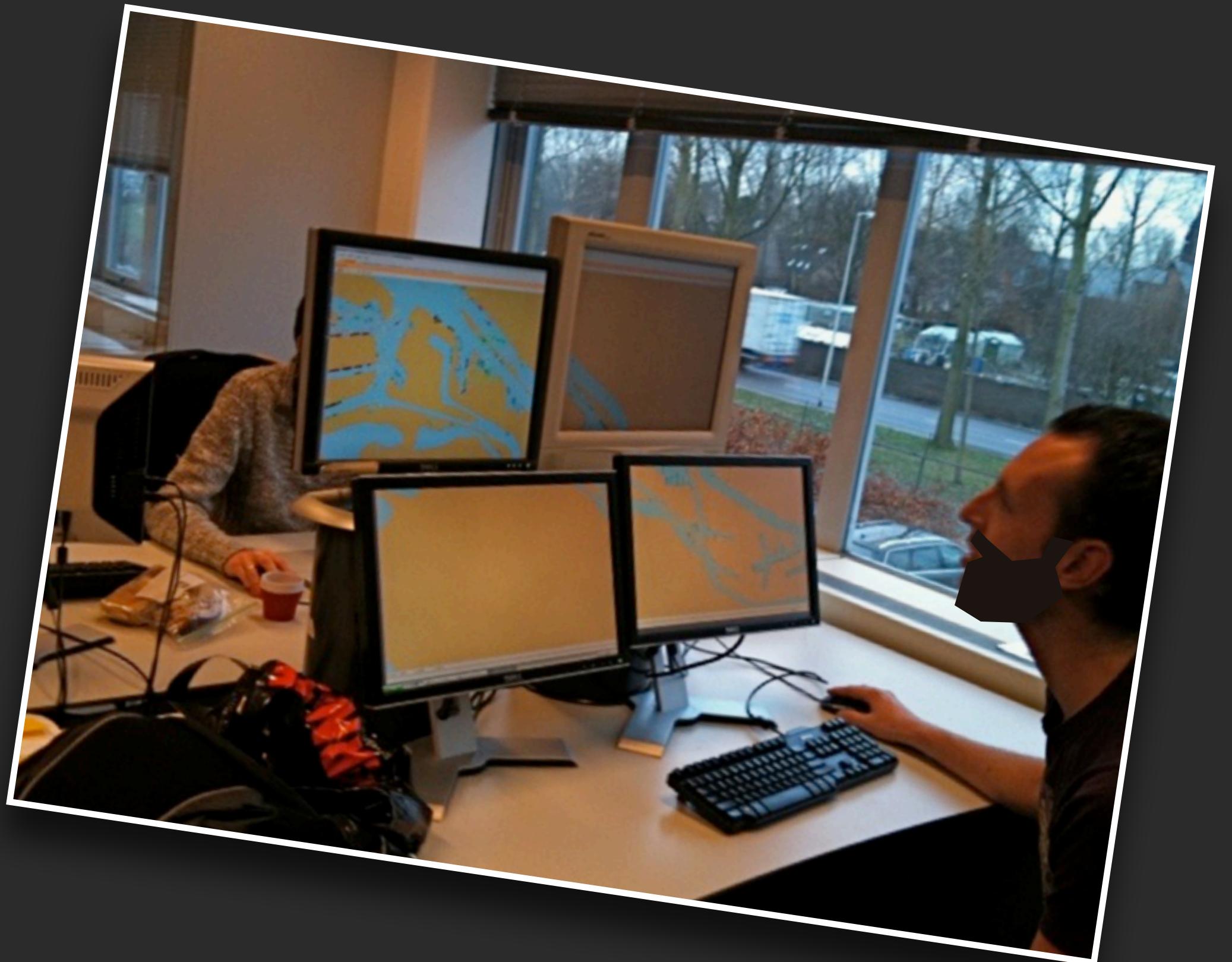
# The history of HaMIS

- I joined the team!

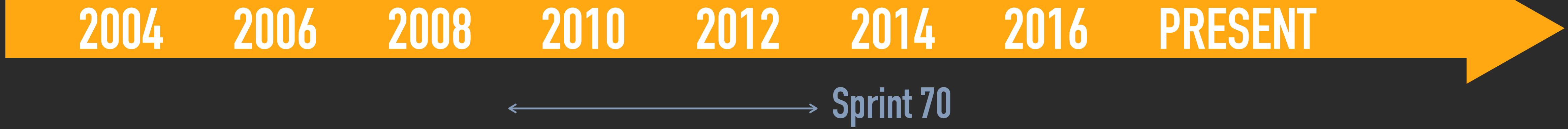


2004      2006      2008      2010      2012      2014      2016      PRESENT

↔ Sprint 12



# The history of HaMIS



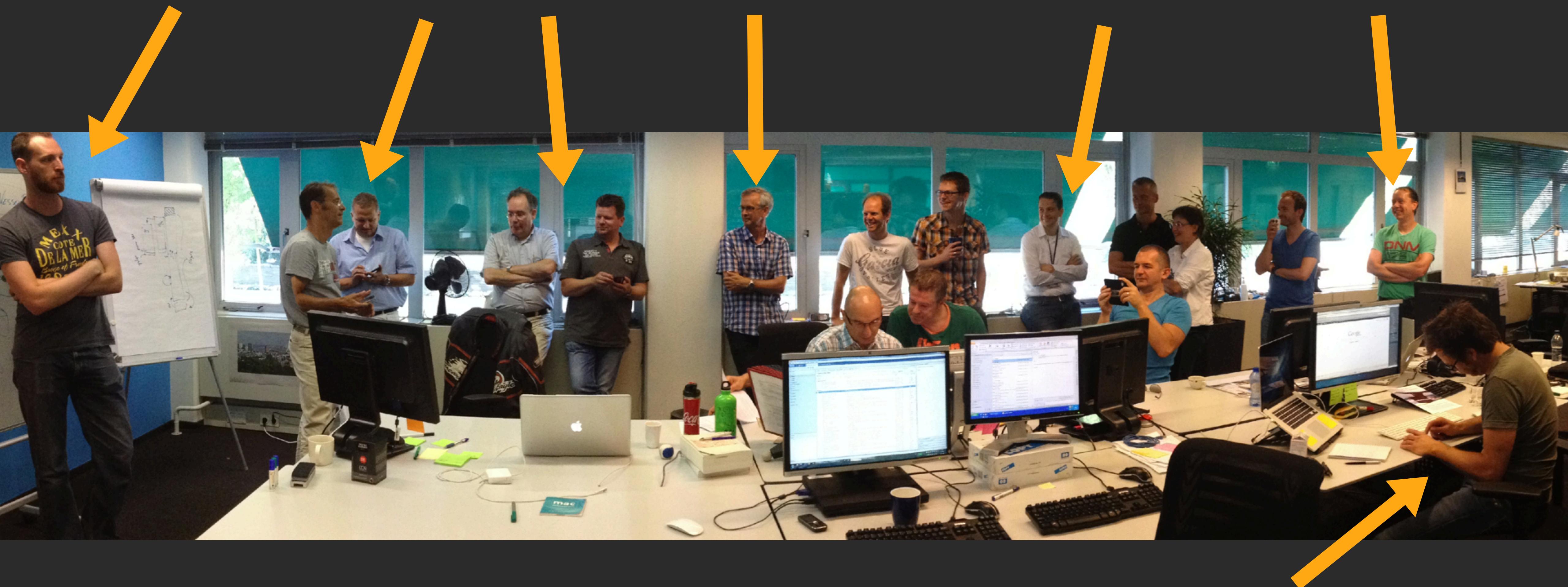
# The history of HaMIS



# The history of HaMIS



# The history of HaMIS



# The history of HaMIS

- Another party: Sprint 100
- More and more users and epics
- Application grows and grows



# The history of HaMIS



# HaMIS: The team

- Multiple teams (5x +/- 5 people) working on the same codebase
- 100% cross functional

# Technical details: HaMIS

HaMIS 0.0.0-SNAPSHOT 20170419-1213 :: Ontwikkelomgeving

Bestand Profiel Gegevens Venster Help

Startmenu Zoeken | Havenkaart Gemelde gevaarlijke stoffen | Uitgevoerde inspecties | Beoordeling Inspectie | Inspectiechecklist | Havenkaart

Havenkaart Selecteer preset

**Dashboard**

- Reizen
- Bezette ligplaatsen
- Sluisplanning
- Rozenburgsesluis
- Hartelsluis

**Havenkaart**

**Zoeken**

Schaal: 1 : 30000 X: 61741 Y: 445221

Zoeken Gemarkeerde schepen | Meldingen overzicht | Te beoordelen binnenvaart | Schepen | Zoeken Typ hier om te filteren

Id UCRN Schip Huidige ligplaats Bestemming Type schip ETA MC Scheepsnummer Roeplettters Actueel bezoek

0 van 0 resultaten

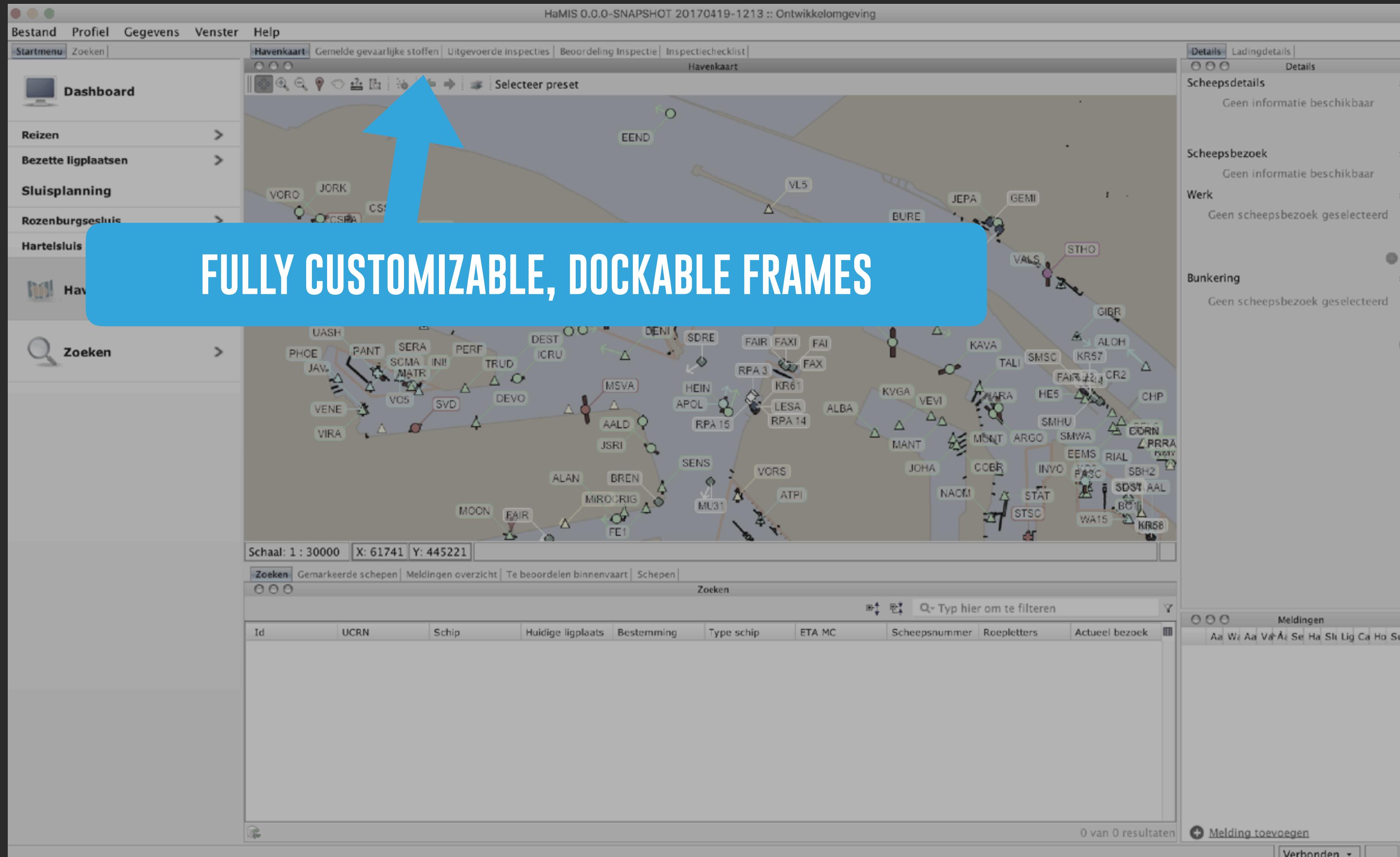
Meldingen Aa Wi Aa Va Aa Se Ha Sli Lig Ca Ho Su

+ Melding toevoegen

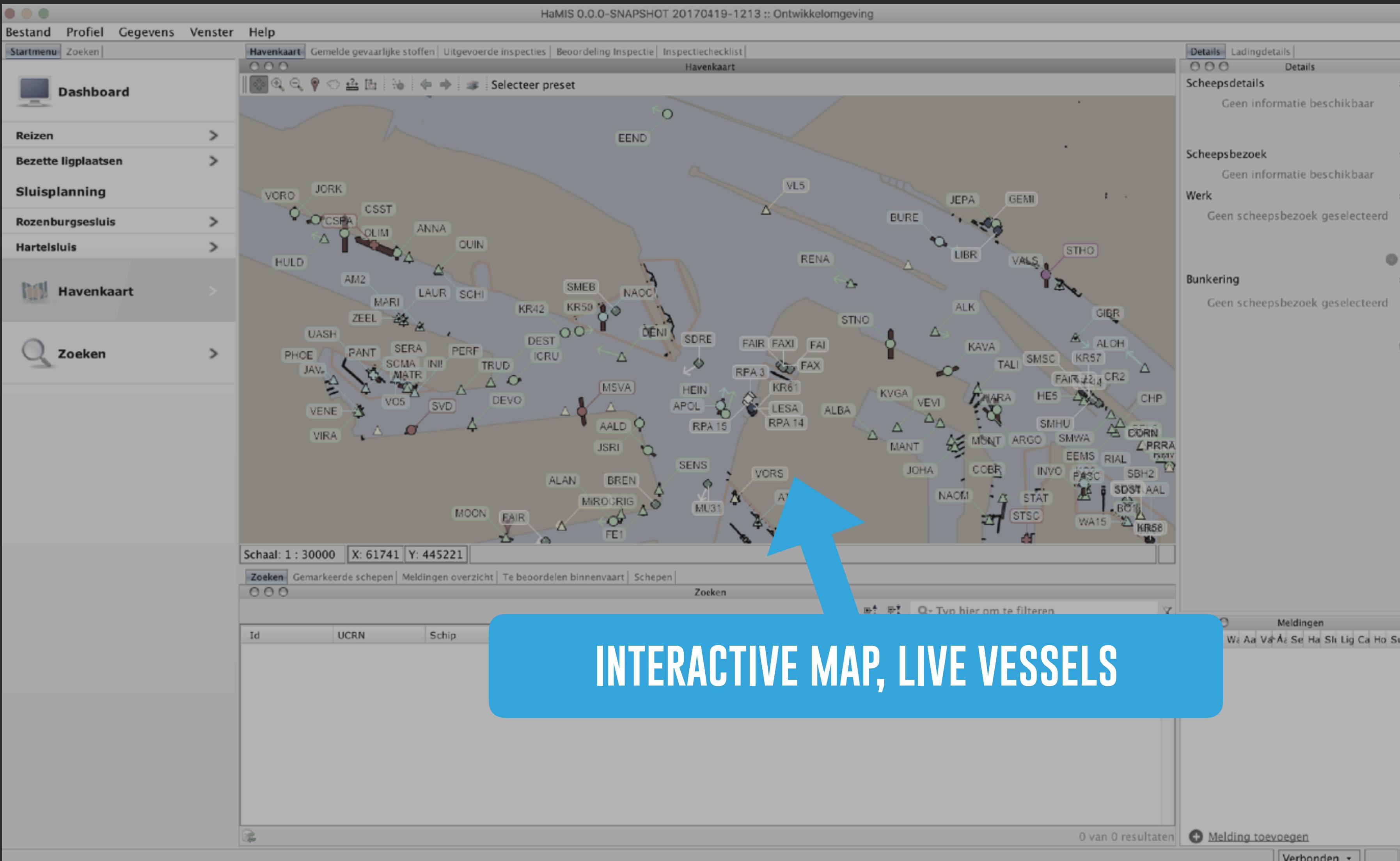
Verbonden

The screenshot displays the HaMIS software interface, specifically the 'Havenkaart' (Port Map) module. The main window shows a map of a port area with numerous ships marked by green dots and labeled with names such as EEND, VLS, BURE, JEPA, GEMI, etc. The map includes various port facilities and channels. On the left, there is a navigation menu with options like Dashboard, Reizen, and Havenkaart. Below the map is a search bar and a table for ship details. On the right, there are sections for Scheepsdetails, Scheepsbezoek, Werk, and Bunkering, all currently showing 'Geen informatie beschikbaar'. The software version is HaMIS 0.0.0-SNAPSHOT 20170419-1213 :: Ontwikkelomgeving.

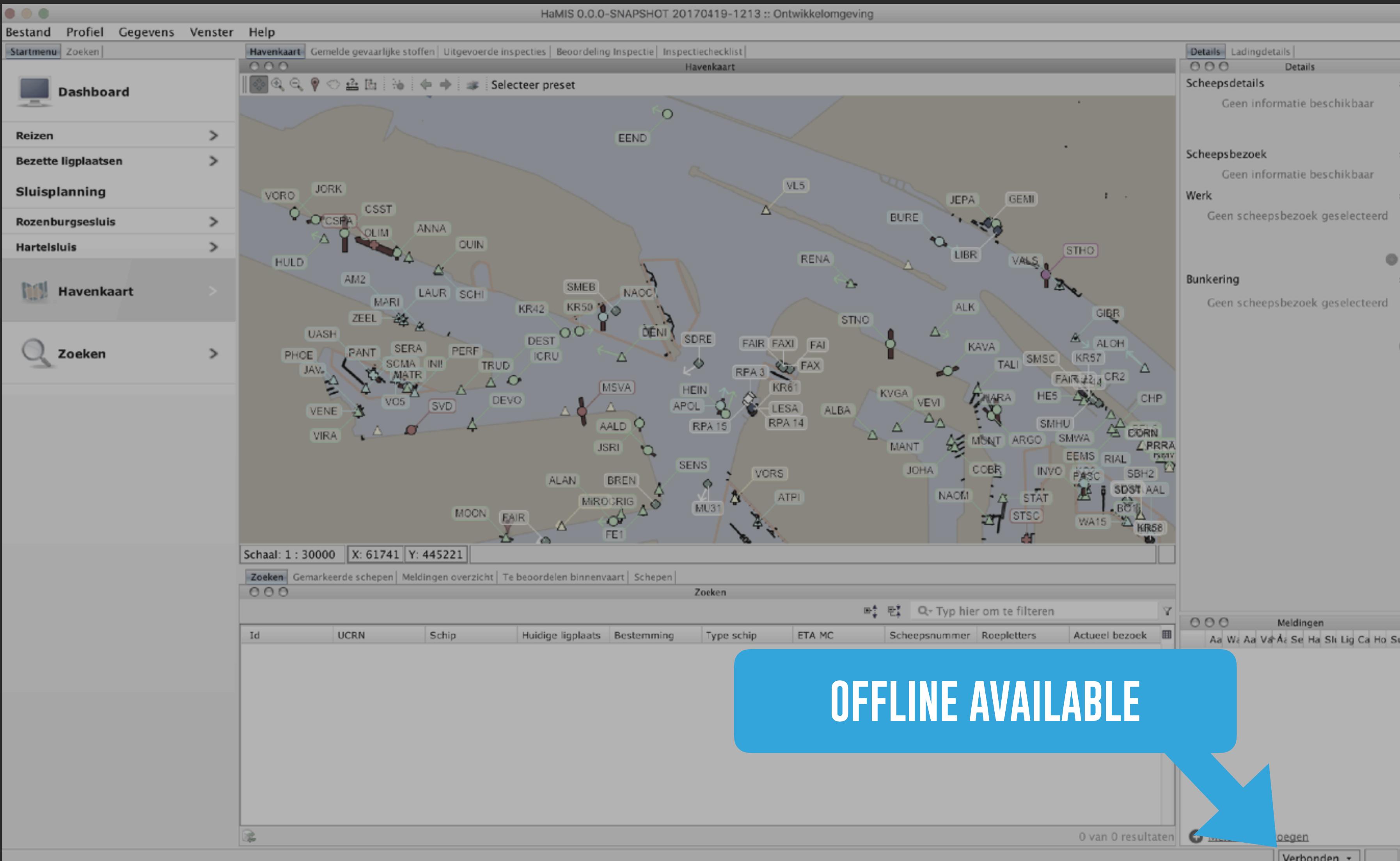
# Technical details: HaMIS



# Technical details: HaMIS



# Technical details: HaMIS



# OFFLINE AVAILABLE

# Technical details: HaMIS

- Frontend:
  - Java WebStart, cached locally
  - JIDE 'docking framework', Swing panels
  - Communication to backend: Hessian



# Technical details: HaMIS

- Backend:
  - Java EE, running on JBoss AS > WildFly 10
  - Two nodes, shared state: Hazelcast
  - Oracle database with Hibernate



# Lessons learned

---

1. Teaching is learning
2. Bureaucracy
3. Being agile
4. Technical debt
5. Evolving architecture
6. Self-organising teams
7. Microservice transition
8. Team dynamics
9. Cargo cult
10. Survivorship bias

# First rule of HaMIS: Always experiment

- Is it successful?      Great!
- Is it unsuccessful? Also great, maybe even \*more\* valuable !

# **Sharing: Helping each other out**

---

**teaching is learning something the second time over**

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Bureaucracy

---

**Bureaucracy is compensating  
for incompetence and lack of discipline**

- Jim Collins

# Bureaucracy is about trust

---

Freedom needs **trust**

Trust gives **freedom**

# Spiral of bureaucracy

- Project grows, more actors, more functionality
- Complexity creeps in
- Need to 'manage' the complexity
- Rules, paperwork, standardised procedures
- Creativity decreases, responsibility decreases... bureaucracy.



- James C. Collins

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Agile vs agile

- Scrum is a set of best practices
- Don't just follow the rules, **apply where needed**
- Ask yourself: **Why** are we doing **X**?

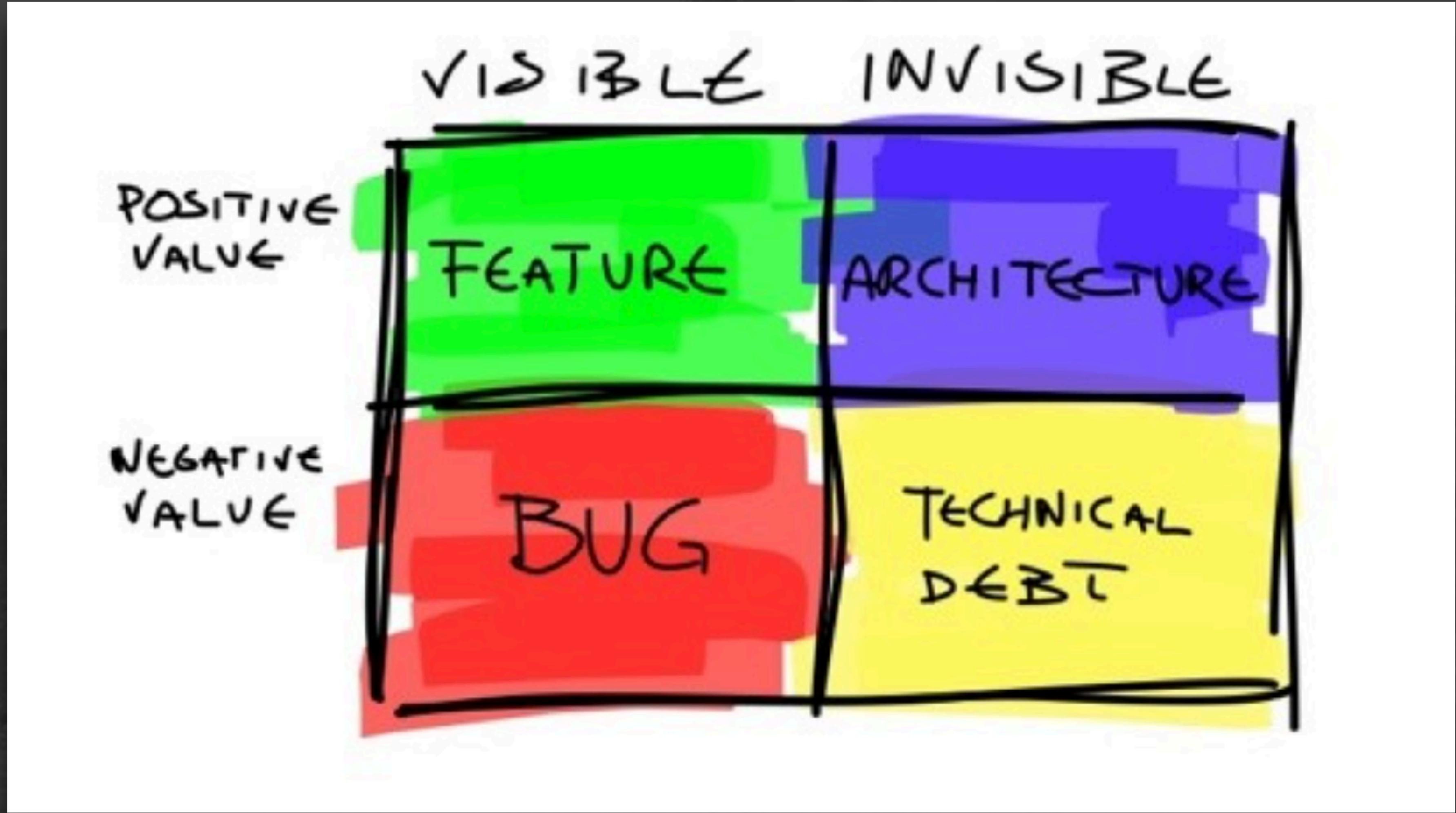
**don't do Agile, be agile**

# Lessons learned

---

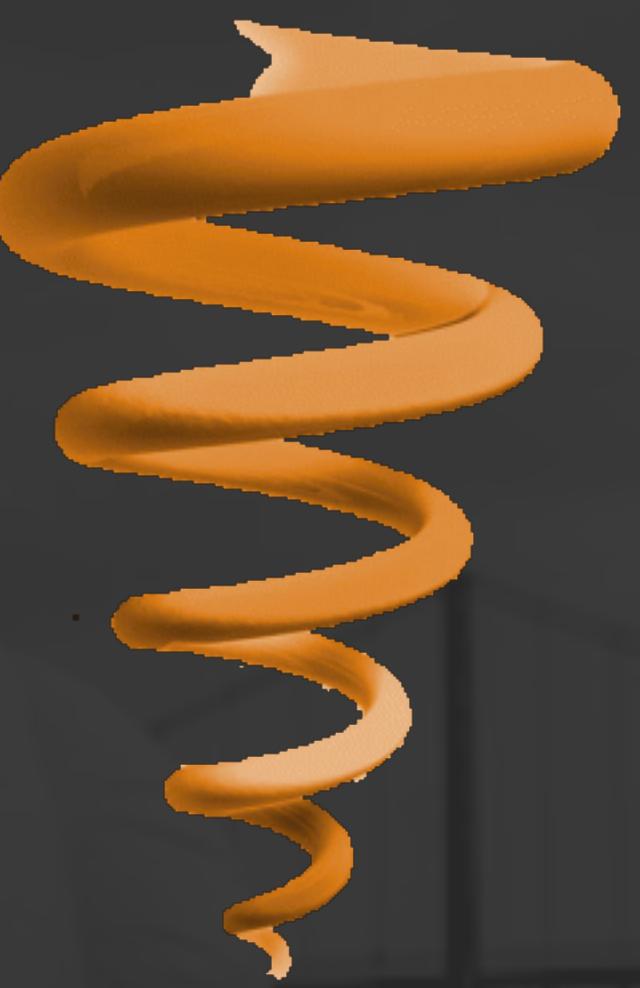
- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Technical debt



# Spiral of Technical debt

- Velocity goes down
- Take shortcuts
- Focus on functionality instead of quality and architecture
- Velocity decreases even more
- Take more shortcuts (etc etc)



# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Software architecture

- Software architecture is about making fundamental structural choices which are **costly to change** once implemented
- Better: Always try to **design for change**

- Wikipedia  
- Viktor Grgic

# Software architecture

- Just in time architecture, find the last **responsible** moment
- Decisions up front are often made with **limited** knowledge
- Do the **simplest** thing that could work

- Mary Poppendieck  
- Barry Boehm

# Software architecture

- Let every **line of code** you write be part of an evolving architecture
- Architecture is about **code**, not documents

# Architecture, shared responsibility

- When firing the architects, we spared no-one
- Architecture became a **shared responsibility**



# Architecture, joined effort

- Architecture **tribe** to brainstorm and create a **vision**
- Team members convinces the PO to **prioritise** 'debt' and tech stories
- Code Camp sessions: spread the vision

# We need someone...

- Who organises the tribe meetings and code camps?
- What if there is a stalemate?
- Who keeps spreading the vision?
- Who handles all the questions from outside the project #shitfilter



We do need someone for this... lets call him/her: the **ARCHITECT**

# Modern/servant architect

- Acts as glue between teams
- Is free to think about issues **unrelated** to current affairs/epics
- Asynchronous **buffer** between:
  - The teams
  - Management and teams

#shitfilter



# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Self-organising teams

- Self organising teams are **\*great\***
- If you want something: **make it happen**, you have the freedom!
- **BUT:** Shared responsibility often means **no** responsibility

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Microservices at HaMIS

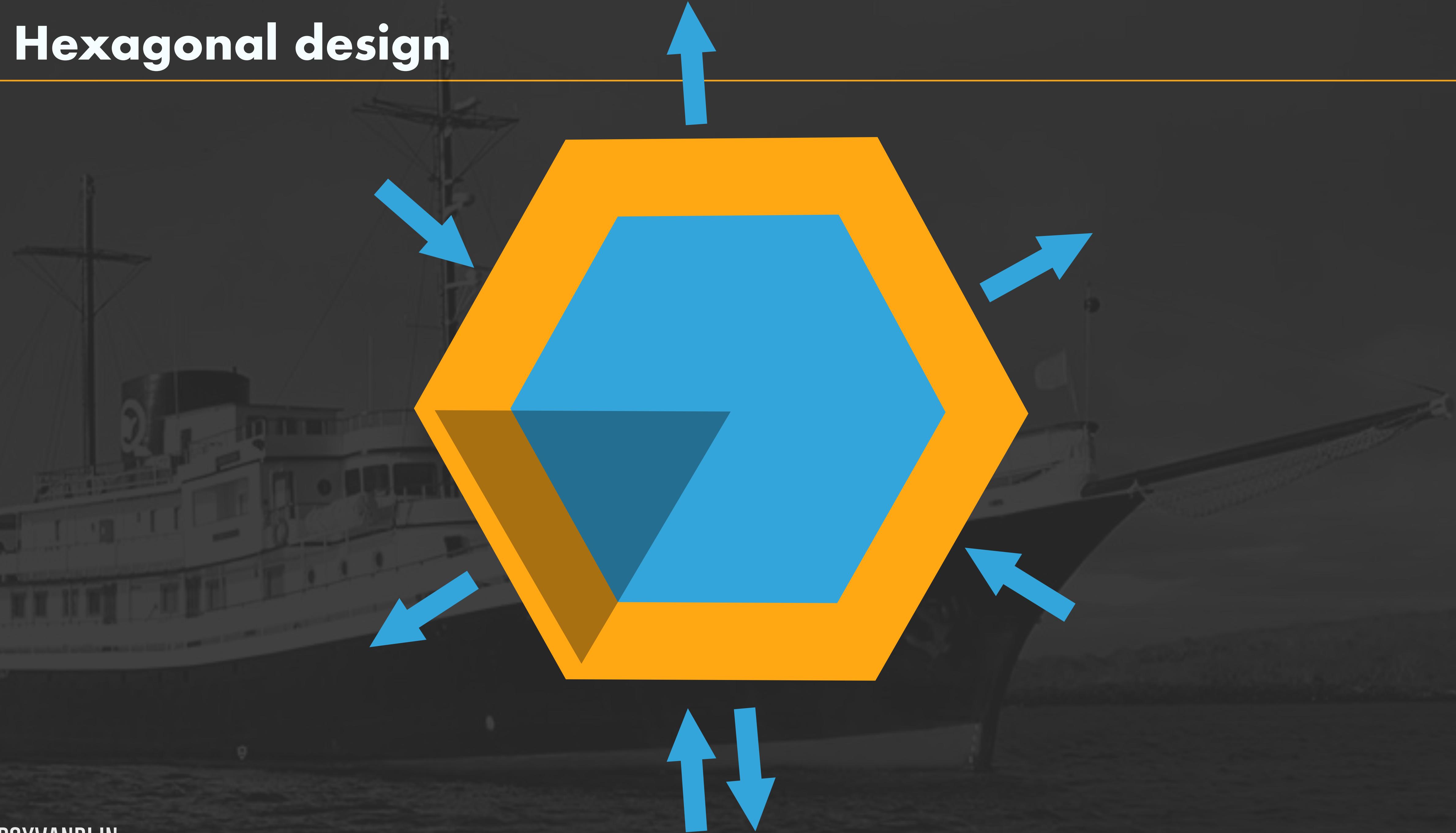
- Problems:
  - Deployment cycle took way too long
  - Stuck with **old** technology, Swing and one big pile of EJBs
  - HaMIS is no longer **sexy**...

# Design evolution

- Three-tier design
- Hexagonal design
- Microservices



# Hexagonal design



# Microservices at HaMIS

How do we make a smooth transition?

# Microservices at HaMIS

- Frontend:
  - Replace JPanels with JXBrowser running Angular/HTML 5
- Backend:
  - Cut from monolith, place in microservice
  - Oracle database, migrate to separate schema

TeamDev



# Microservices at HaMIS

- 30+ backend microservices (WildFly, Spring Boot)
- 14+ Angular projects (NGINX with Angular 2)
- 1 leftover frontend

# Microservices at HaMIS : Hosting

- Everything is running on KPN AppFactory
- Which is: Kubernetes



# Freedom of technology

- Teams are picking their own technology
  - **Good:** Freedom, creativity and state of the art technology
  - **Bad:** Microservices deteriorate much faster...

# Drawbacks in teams

- More drawbacks:
  - Code is being **duplicated/copied**
  - **Monitoring** microservices is (**MUCH**) harder
  - Coding **styles** are diverging

# Save the best for last

- Breaking up our monolith:
  - Start **easy**
  - Work your way down
  - Sounds good right?
  - **Better:** Create separate API for the leftovers

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Conway's Law



**organisations which design systems  
are constrained to produce designs  
which are copies  
of the communication structures  
of these organisations**

- Melvin Conway

# Inverse Conway's Law

---



**organisations which radically  
change their system design  
should expect changes  
in communication structure**

- Roy van Rijn

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Cargo cult, the origins

a tribe in Melanesia



# Cargo cult in IT



Spotify does X



Spotify is successful



We don't do X



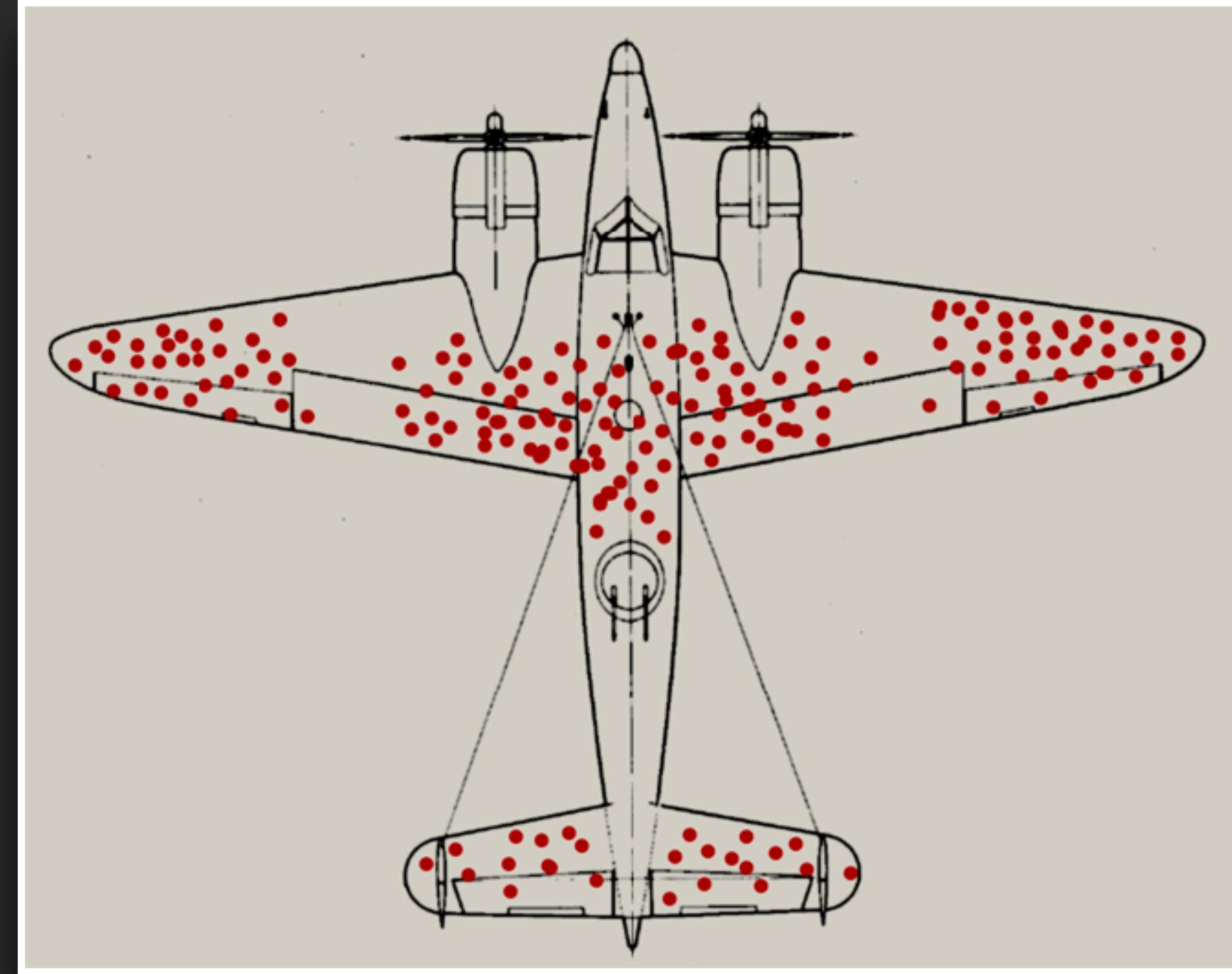
If we do X too, we'll be successful!

# Lessons learned

---

- 1. Teaching is learning
- 2. Bureaucracy
- 3. Being agile
- 4. Technical debt
- 5. Evolving architecture
- 6. Self-organising teams
- 7. Microservice transition
- 8. Team dynamics
- 9. Cargo cult
- 10. Survivorship bias

# Survivorship bias, two examples



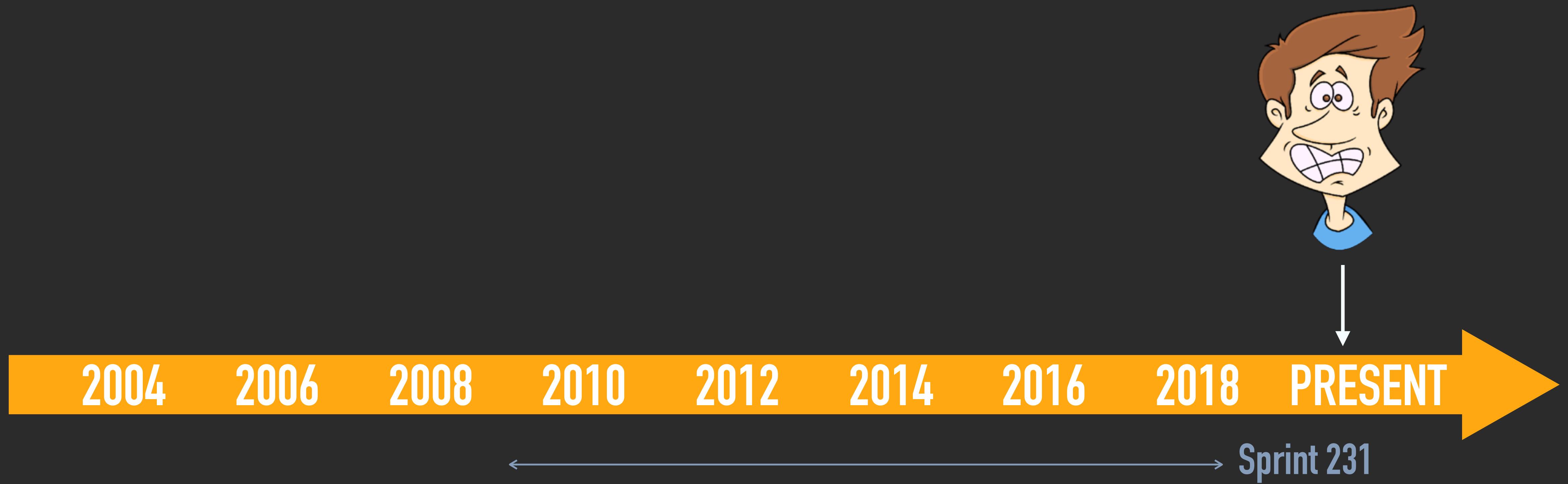
# Survivorship bias in IT/conferences

---

I went to a conference and heard six talks about successful microservice projects

1. Nobody talks about failed projects using microservices...
2. There are more successful projects **not** doing microservices

# HaMIS timeline



# 99 problems...

---

- Big scary refactorings are left unfinished
- No clear vision about the frontend and the product(s)
- Scaled Agile Framework (SAFe) and LeSS are **not** friends

# The future

- But we'll be okay...
- We'll keep evolving!



IF YOU HAVE  
**QUESTIONS** OR **ANSWERS**

---

RAISE YOUR HAND

