Driving SAFe

James Janisse, VP

World Class Software Delivery Program

Agenda

- Our Situation in 2012
- Our Transition to SAFe
- Experience and results
  - The Good
  - The Bad
  - The Ugly

TomTom @ a Glance

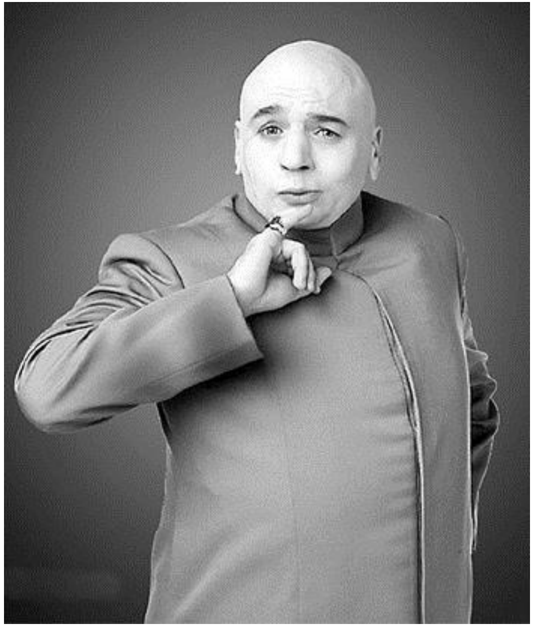
- Founded in Amsterdam in 1991
- 1 billion euros annual revenue
- 4,000 employees in 35 countries worldwide
- Maps cover 112 countries
- Real-time traffic in 32 countries
- 70 million PNDs sold since 2004
- 3 million in dash navigation systems sold since 2009
- **1-2 Million lines/code per navigation product**
- **750-1000 engineers in 11 development sites**

TomTom *January 2012*

- Organised as waterfall projects
- Many projects working in all parts of the code with minimal module or component ownership
- Many releases are months-quarters late
- Multiple code lines and branches
- Negligible automated testing & no continuous integration
- “downstream” teams spend 3,4,5 months accepting the code and often changing it
- Poor visibility and facts based decision making

Conclusions

- Waterfall and staged gate milestones are not working for us
- Project orientation is wrong
- Branching the code is evil
- Complexity is too high
- Waste
- Insufficient information to make effective decisions



Strategy

- Transform from one-time project orientation to component-based continuous integration and delivery

Agenda

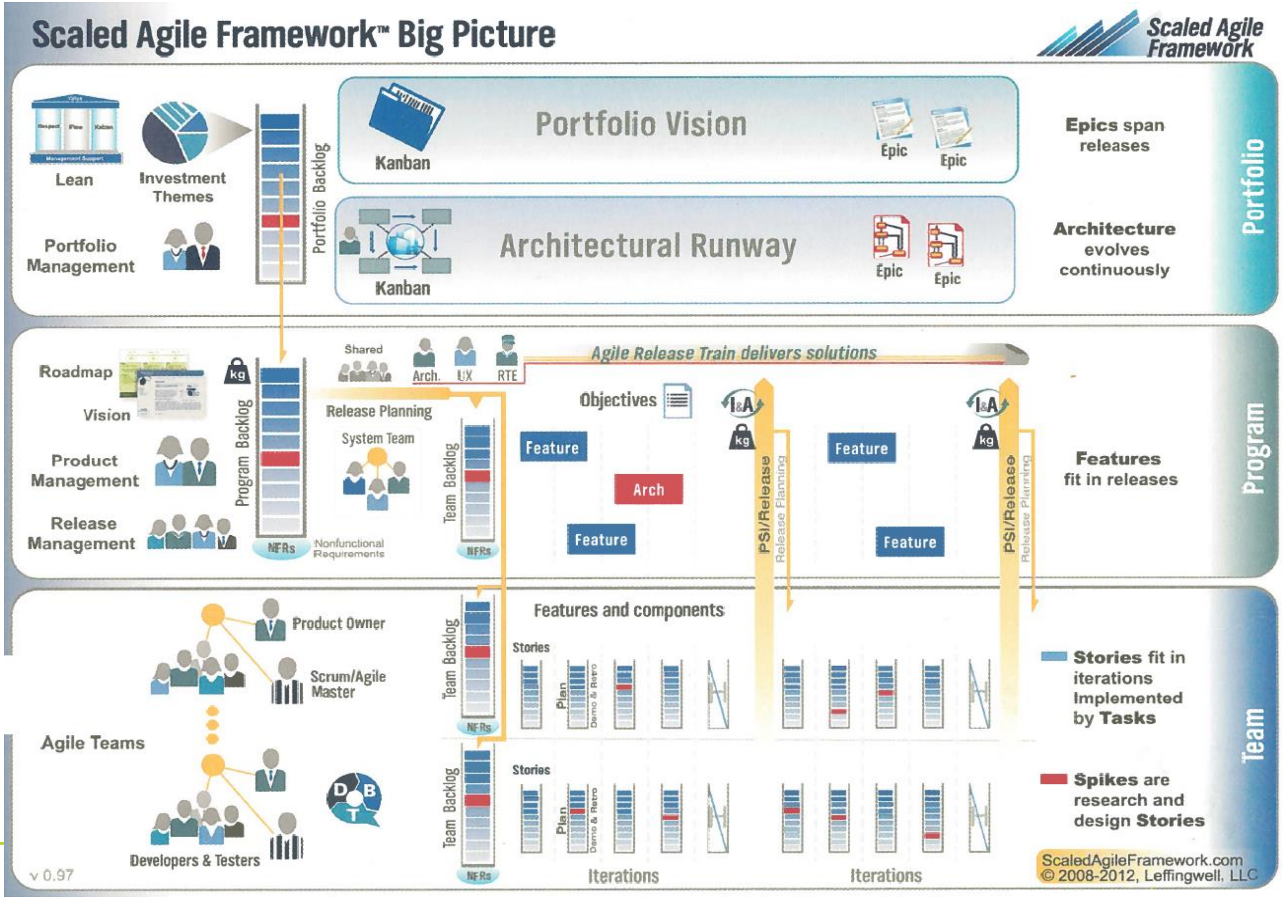
- Our Situation in 2012
- Our Transition to SAFe

We First Invented Our Own Solution: *Feature Flow*

Then We Made a Discovery

Agile Software Requirements
Agile Softwa
Requirements
NEW!
Lean Requirements Practices for Teams, Programs, and the Enterprise
Dean Leffingwell

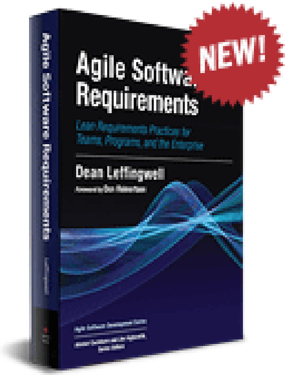
Then We Made a Discovery



ScaledAgileFramework.com
© 2008-2012, Leffingwell, LLC

Adoption Timeline

1. Book arrives on-site and is a read by a few Agilistas
2. Replace homegrown *Feature Flow* with SAFe
3. Give Book to SVP who reads it cover to cover on his vacation
4. SVP buys book for CTO and other SVPs
5. Attend SAFE Training
6. Trained 50 Certified Scrum Masters & 50 CPOs
7. Re-org


NEW!
Agile Softwa
Requirements
Lean Requirements Practices for Teams, Programs, and the Enterprise
Dean Leffingwell
Agile Software Requirements

We Re-organised from Scrum Teams up

1. Key assumption is value is only created by scrum teams
2. Organise into Product clusters and component scrum teams
   - One Agile Release Train per Product
   - Every active/viable module/component is allocated to one and only one scrum team
3. Adjust/supplement all scrum teams to have scrum master, developers, etc
4. Everyone not in a scrum team is put in a backlog i.e. Project Managers, Resource Managers, Team leads….
5. Design a thin/lean program support team to feed the scrum teams: Product Owner, Architect, Systems Team
6. Design a thin/lean portfolio team

Portfolio
Program
Team
08-2013
/ell, LLC

6 Months Into the Transition a New Goal

Launch the 4th Generation of our Consumer navigation product

- You have 126 Days till launch
  - usually 1 year project

- Bring in teams from 2 other (waterfall) Product Units and work as one integrated team

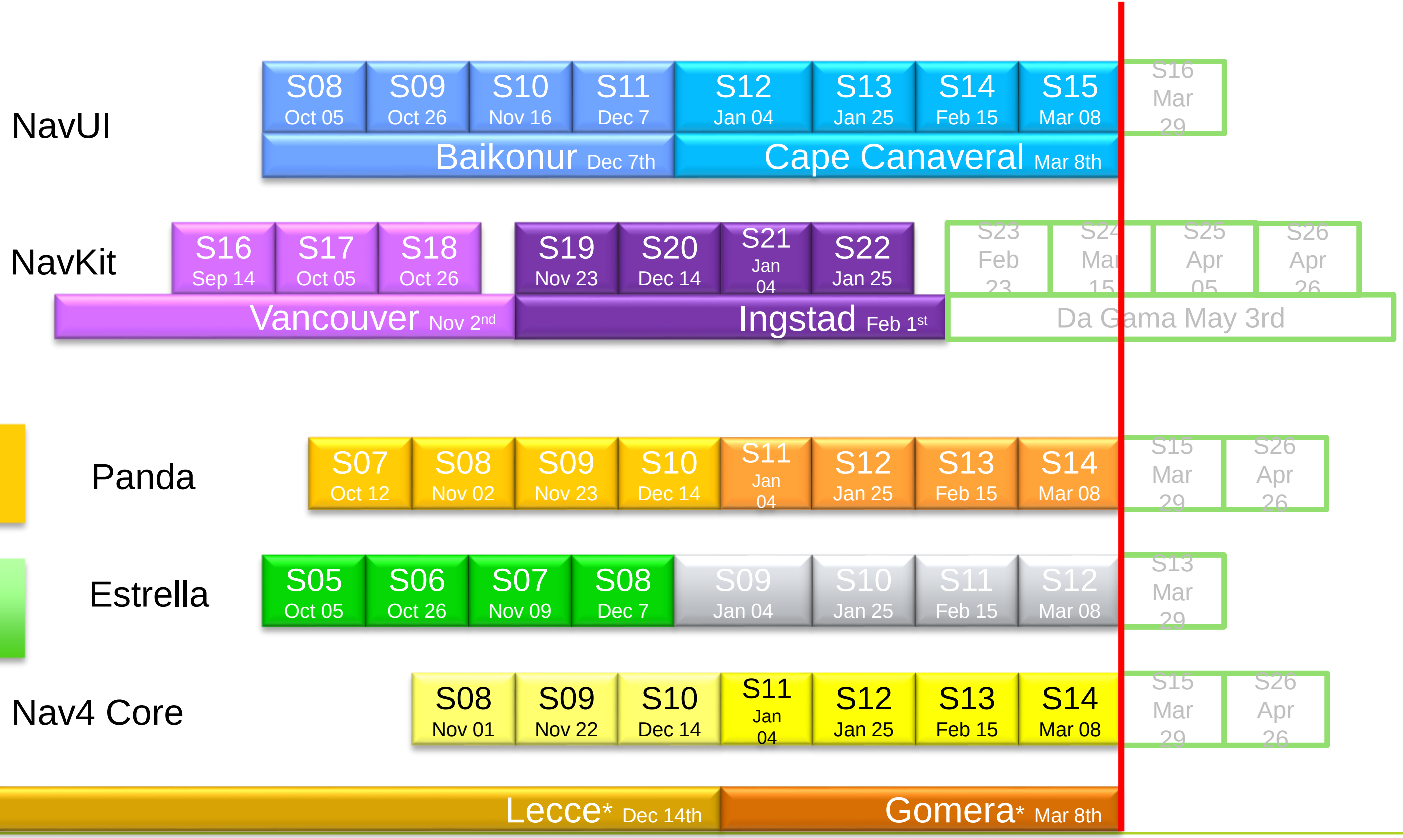
Doubled Number of Agile Release Trains

- 5 Scrum teams
- 3 locations

- 14 Scrum teams
- 4 locations

- 4 Scrum teams
- 2 locations

- 5 Scrum teams
- 2 locations



NavUI

S08 Oct 05 | S09 Oct 26 | S10 Nov 16 | S11 Dec 7 | S12 Jan 04 | S13 Jan 25 | S14 Feb 15 | S15 Mar 08 | S16 Mar 29

Baikonur Dec 7th | Cape Canaveral Mar 8th

NavKit

S16 Sep 14 | S17 Oct 05 | S18 Oct 26 | S19 Nov 23 | S20 Dec 14 | S21 Jan 04 | S22 Jan 25 | S23 Feb 23 | S24 Mar 15 | S25 Apr 05 | S26 Apr 26

Vancouver Nov 2nd | Ingstad Feb 1st | Da Gama May 3rd

Panda

S07 Oct 12 | S08 Nov 02 | S09 Nov 23 | S10 Dec 14 | S11 Jan 04 | S12 Jan 25 | S13 Feb 15 | S14 Mar 08 | S15 Mar 29 | S26 Apr 26

Estrella

S05 Oct 05 | S06 Oct 26 | S07 Nov 09 | S08 Dec 7 | S09 Jan 04 | S10 Jan 25 | S11 Feb 15 | S12 Mar 08 | S13 Mar 29

Nav4 Core

S08 Nov 01 | S09 Nov 22 | S10 Dec 14 | S11 Jan 04 | S12 Jan 25 | S13 Feb 15 | S14 Mar 08 | S15 Mar 29 | S26 Apr 26

Lecce* Dec 14th | Gomera* Mar 8th

“There is no doubt in my mind that without SAFe and Rally we would not have launched this in only 140 days. It is also our best new product ever”

All New TomTom GO500
EU 45 Countries
Lifetime Traffic
RRP: £**199.99**

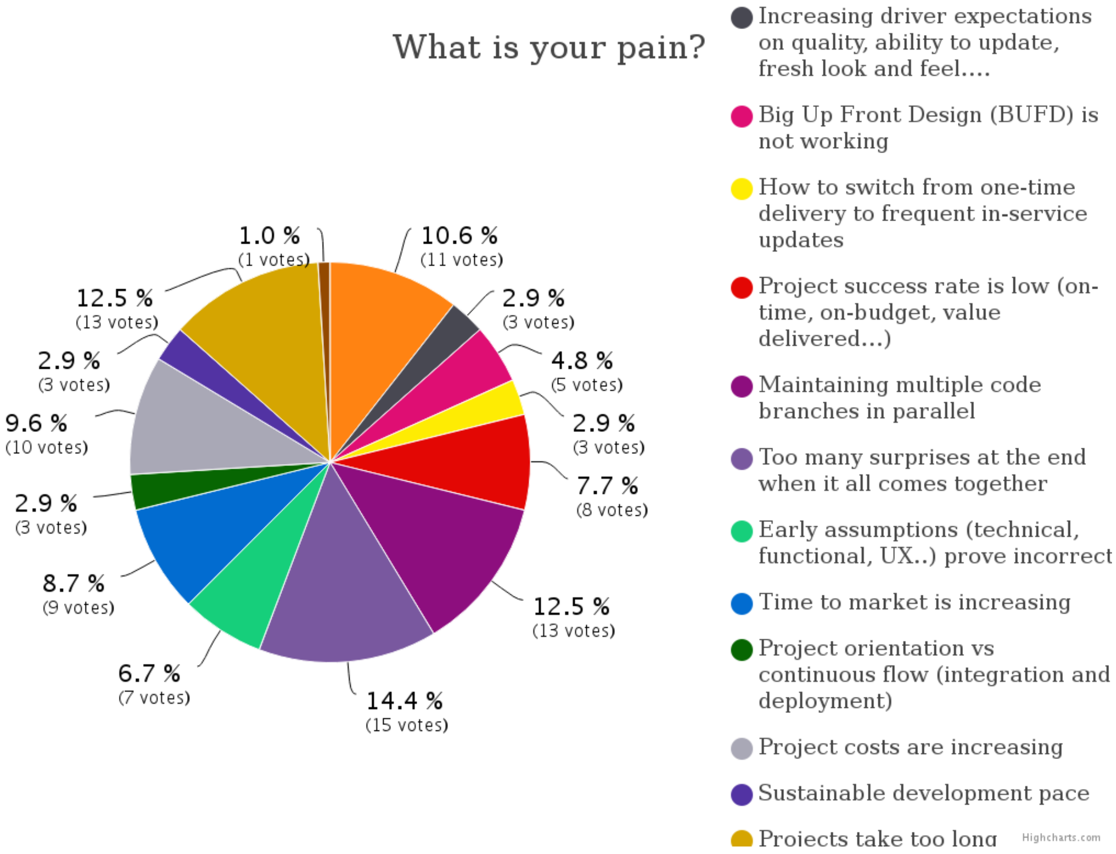
2014 to Present

- SAFe is adopted by all large product teams
  - Approximately 750 FTEs
    - Navigation software
    - Online services
    - Map creation software
    - Sports software
  - 515 users of CA Agile (Rally)
  - 200+? people trained in SAFe

Agenda

- Our Situation in 2012
- Our Transition to SAFe
- Experience and results

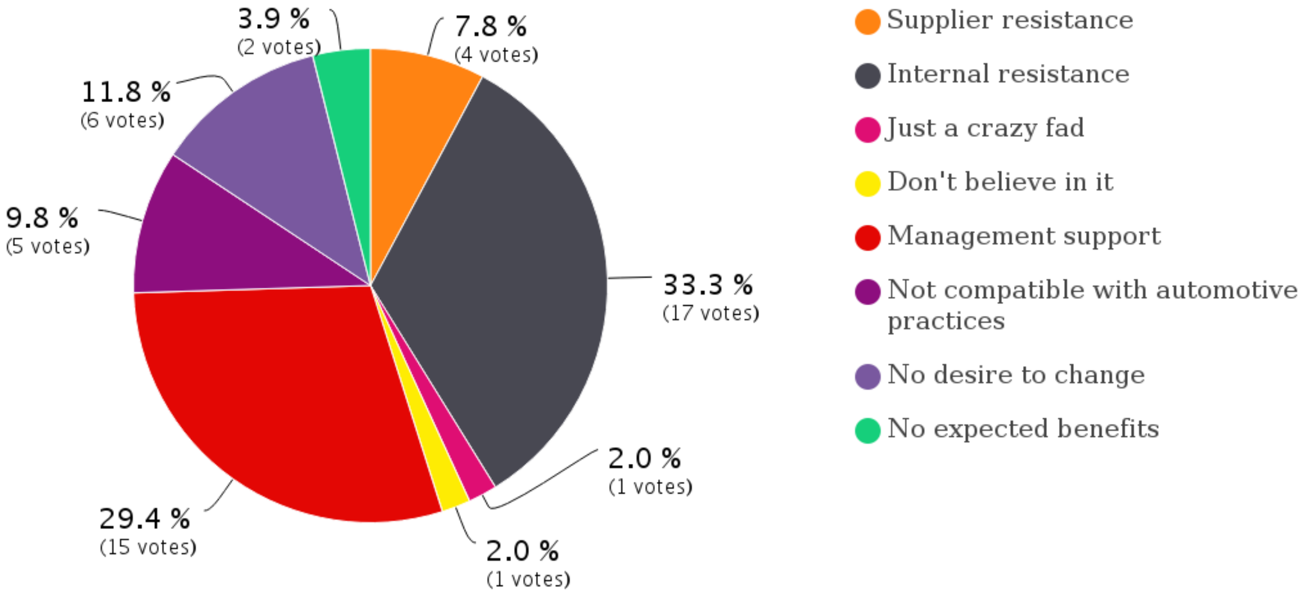
Agile in Automotive, Stuttgart, November 2015



🟨 Getting requirements 100% correct up front

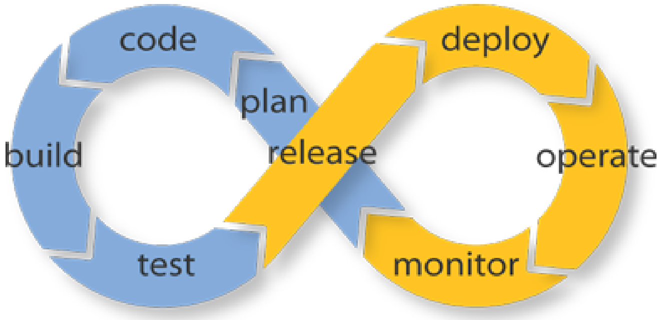
Agile in Automotive, Stuttgart, November 2015

What is stopping your from adopting Agile practices



Question

What are the differences between…

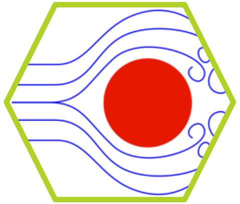


and

Delivering value via projects

Delivering value via continuous integration & delivery

Flow



2012

- Batch size = 1
- Work is organized by stages:

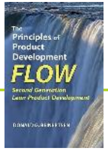
  Plan > Analyse > Develop > Test

- Risk remains till end
- Value only delivered at end

Now

- Uncouple required features to flow independently *and* as fast as possible

- High risk & high value working first
- Release value in pieces ASAP



The Principles of Product Development FLOW

Second Generation Lean Product Development

The Principles of Product Development Flow

Donald Reinertsen

Organisation

2012

- Bring people to the requirements

- Overhead of “Resource Managers”
- New team has no history of working together
- Throughput is unknown
- Impact of schedule risk

Now

- Bring requirements to the team(s)

- Minimal overhead
- Proven historical velocity
- Established way of working
- Clear long term ownership
- Self managing teams
- Team not managers commit

Requirements

2012

- Assumes that we, or our customers, can fully understand **all** the requirements up front
- Change is discouraged
- One chance to be “perfect”
- One chance so ask for everything

- Does not work for high user experience scenarios, where you need to see it working first
- Discourages innovation

Now

- Add features and fine tune performance over time
- Fail fast, learn, improve
- assume that change will be constant, and we deliver in small increments to better track change

- Fine tunes and improves the user experience
- Reach minimum marketable product faster & with less waste

Architecture

2012

- Big Up Front Design (BUFD)

- Cannot spend months and months designing a future proof perfect architecture
- Often based on many untested assumptions and hypothesis

Now

- Architectural vision & emerging runway

- Better to have a vision and "barely sufficient" architecture
- Test assumptions & hypothesis ASAP
- Change is constant so architectural runway is built just in time to prevent waste
- The need to refactor is not failure

Performance KPIs

2012

- On-time
- On-budget
- Code Maturity
- Timesheets & estimate to complete

- Never ask if project X is an improvement over Project Y
- Mostly time accounting focus
- Time spent does not easily equate to business value delivered

Now

- Measure actual value delivered
- Cycle time trend
- Velocity trend
- Sprint and Release Burndown

- Focus is on trends and continuous improvement
- No more timesheets

Demo

2012

- Runs near the end of the project
- Hope for a miracle near the end of the project
- Very little opportunity to improve or re-factor
- Cost to change behavior is high

Now

- System always runs
- Demo production ready, tested software every 2-3 weeks
- Release to beta testers every 2-3 weeks if not daily
- Opportunity to verify designs, make improvements or move on

Code

2012

- Branched for each project
- Ownership unclear

- Significant waste and overhead
- Effort to merge code back in was around 20% of total effort
- Single defect may have to get fixed in each branch
- Projects often break features
- Developers did not have to maintain their code

Now

- Only one mainline
- Ownership 100% clear

- Negligible refactoring or waste
- Each product variant is built from the same code line so it benefits from the entire install base
- Developers maintain their own code and control tech debt

Integration

2012

- Infrequent & near the end

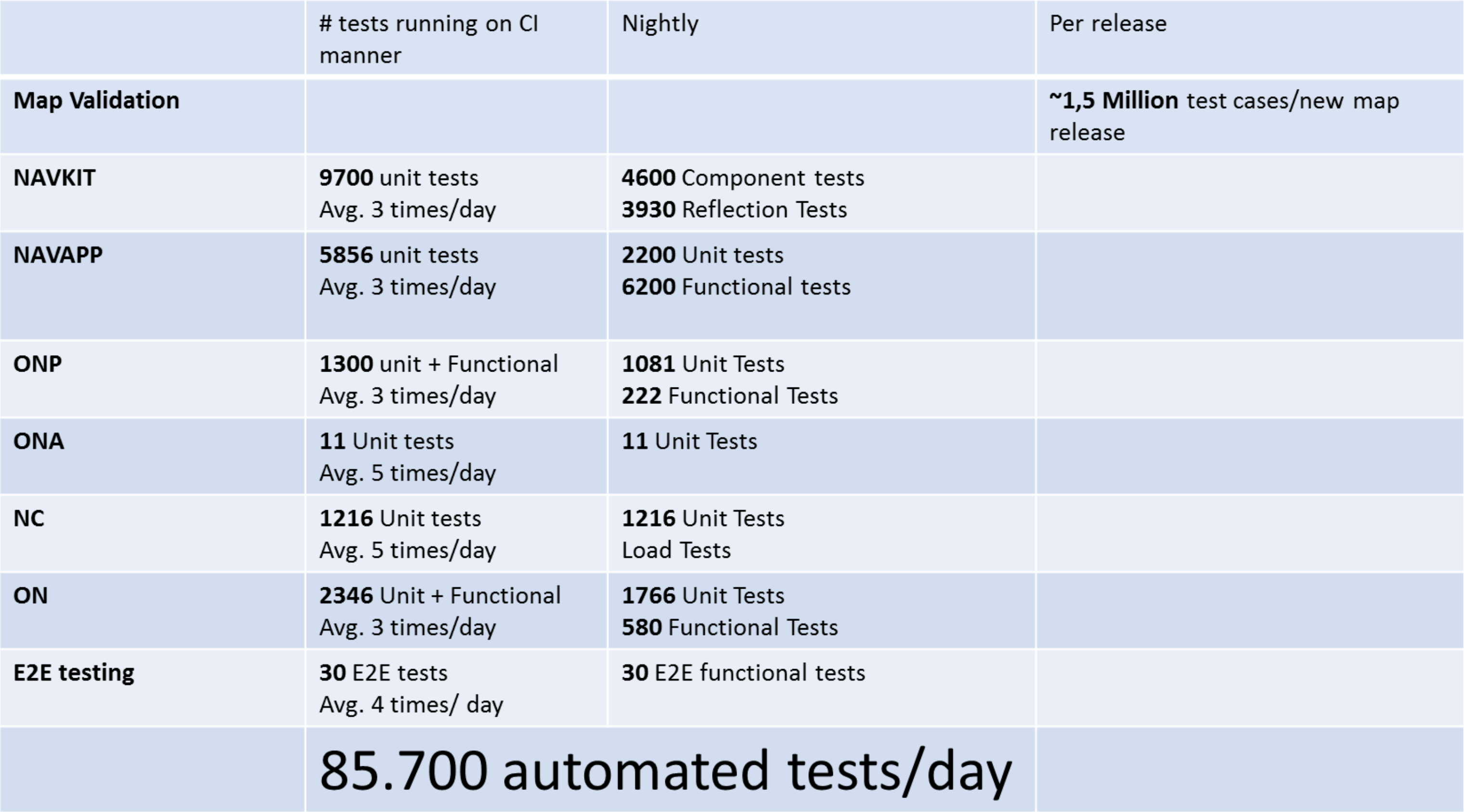
- effort to schedule integrations
- significant effort to conduct integration months after branching
- Each integration would trigger significant corrective actions
- Catching up could take 20% of the total effort

Now

- Continuous and ongoing

- Each developer submits their code several times/day
- Each submission requires success test automation results
- Whoever breaks it fixes it without delay
- Daily full regression suite
- Daily integration with downstream systems

Test Automation & Continuous Integration



	# tests running on CI manner	Nightly	Per release
Map Validation			**~1,5 Million** test cases/new map release
NAVKIT	**9700** unit tests Avg. 3 times/day	**4600** Component tests **3930** Reflection Tests	
NAVAPP	**5856** unit tests Avg. 3 times/day	**2200** Unit tests **6200** Functional tests	
ONP	**1300** unit + Functional Avg. 3 times/day	**1081** Unit Tests **222** Functional Tests	
ONA	**11** Unit tests Avg. 5 times/day	**11** Unit Tests	
NC	**1216** Unit tests Avg. 5 times/day	**1216** Unit Tests Load Tests	
ON	**2346** Unit + Functional Avg. 3 times/day	**1766** Unit Tests **580** Functional Tests	
E2E testing	**30** E2E tests Avg. 4 times/ day	**30** E2E functional tests	
	85.700 automated tests/day		

Summary of “The Good”

Observation	Impact
Always release on fixed schedule	• Reliable and predictable releases of production code • Establishes fixed rhythm
Release **quicker** and more **often**	• Fail fast (<2 weeks) is better than after 6 months • Validate and adapt sooner • Adapt to change/learnings
Run automated tests suite **per** submission & **per** day	• Detect/prevent issues with each new submission • Mainline is always able to run • No bottleneck at the end • Reduces waste as others stay up to date
Single **shared** backlog available to **all** to view	• Improved transparency and info sharing • Done means working capability not task complete
Perpetual teams	• Teams establish ways of working & esprit du corps • Improves estimating by allowing historical comparisons • Enables estimation accuracy analysis • Team controls their own commitments • Sustainable development

Summary of the "Bad"

Observation	Impact
Teaches the business that Agile = 100% predictable	- What happened to iterative development - What happened to incremental development
Enables the business to change direction/ strategy/ priorities often	- Let's face it, too much Agility is just an inability to make choices and decisions i.e. chaos
Everyone can see everyone's backlog, priority, throughput…	- Everyone can second guess your prioritisation - Everyone can second guess your estimates
Shows week teams	- Teams that normally staid below the radar which no one new what they did are suddenly very exposed to the daylight
Goes on forever without a break (HIP downtime)	- Projects used to have a nice slow start up and shut down phase, so cyclical rhythm - Now work is harder and does not let up

Summary of the “Ugly”

Observation	Impact
Some teams will resist	• Reject the need to be part of an ART • Reject the need to have common ways of working…
Some people loose power	• Project managers loose scope, resourcing, and budget • Resource managers are no longer needed
Centralised decision making shifts to decentralised	• Evolving decisions to the lowest level threatens central portfolio level experts like architects and makes guiding independent teams hard
Fully defined transforms to barely sufficient	• Architects still love to make future proof architectural designs and plans • UX want to make pixel perfect designs for all use cases

Thank You

Any questions?

http://nl.linkedin.com/in/jamesjanisse