Tampere University

Lecture 7: continuous deployment – part 2

Tampereen yliopisto
Tampere University

Main principles (https://continuousdelivery.com/principles/)

- Build quality in
- Work in small batches
- Computers perform repetitive tasks, people solve problems
- Relentlessly pursue continuous improvement
- Everyone is responsible

Sound familiar from somewhere?

CI – essential practices (according to Humbley and Farley)

- Don’t check in on a broken code
- Always run all commits tests locally before committing, or get your CI server to do it for you
- Wait for commit tests to pass before moving on
- Never go home on a broken build
- Always be prepared to revert to the previous revisions
- Time-box fixing before reverting
- Don’t comment out failing tests
- Take responsible for all breakages that result from your changes
- Test-driven development

Deployment essential pract.
(according to Humbley and Farley)

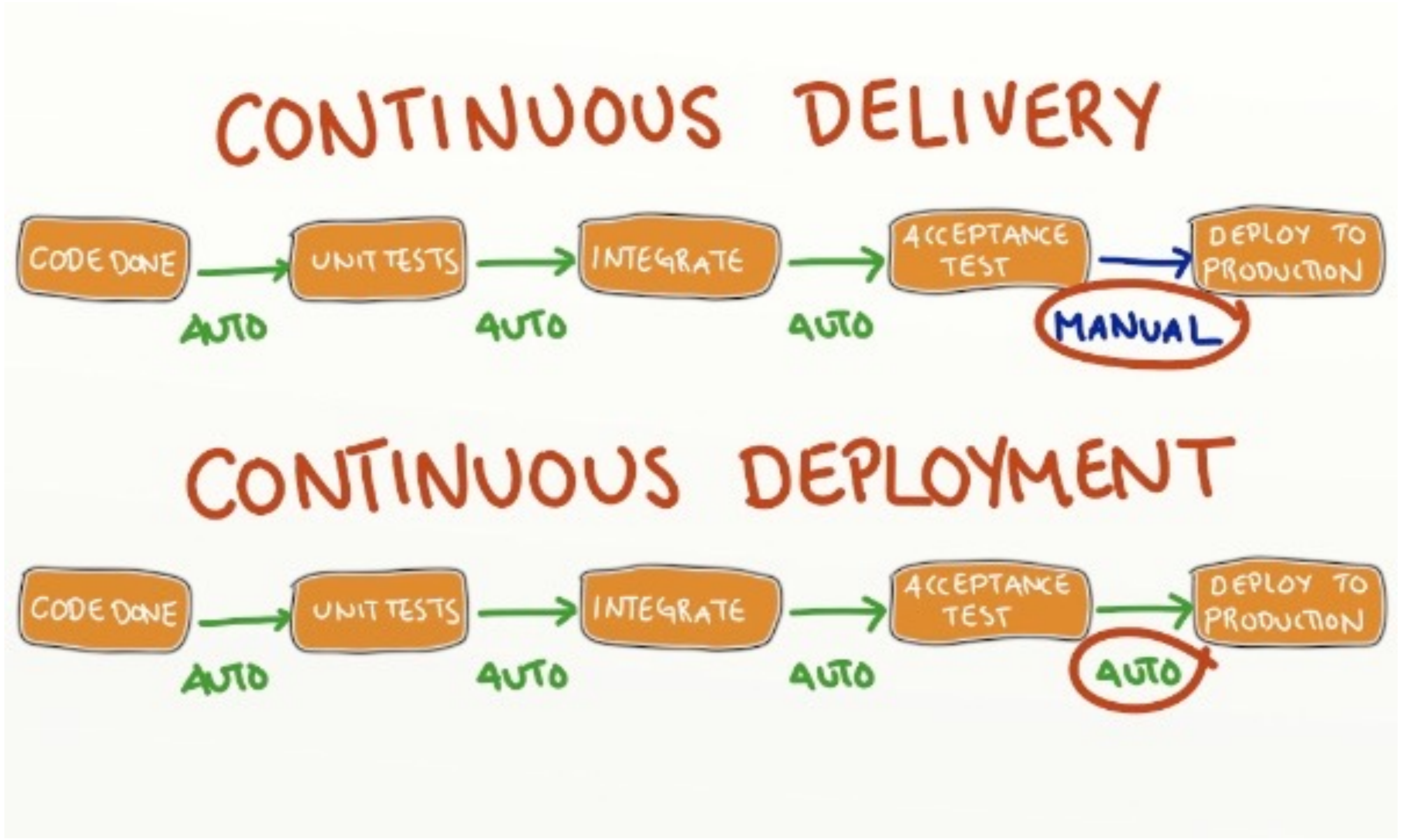
- Only build your binaries once
- Deploy the same way to every environment
- Smoke-test your deployments
- Deploy to copy of production
- Each change should propagate through the pipeline instantly
- If any part of pipeline fails, stop the line

Tampereen yliopisto
Tampere University

Back to CD

Continuous delivery and deployment

(http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment)



Tampereen yliopisto
Tampere University

Perceived benefits

- **Improved delivery speed** of software changes Improved speed in the development and deployment of software changes to production environment.
- **Improved productivity in operations** work. Decreased communication problems, bureaucracy, waiting overhead due to removal of manual deployment hand-offs and organisational boundaries; Lowered human error in deployment due to automation and making explicit knowledge of operation-related tasks to software development
- **Improvements in quality**. Increased confidence in deployments and reduction of deployment risk and stress; Improved code quality; Improved product value to customer resulting from production feedback about users and usage.
- **Improvements in organisational-wide culture** and mind-set. Enrichment and wider dissemination of DevOps in the company through discussions and dedicated training groups ‘communities of practice’

Maturity models in software engineering

- The first welknown was
  - Capability Maturity Model developed by Software Engineering Institute at Carnegie Mellon University in 1986
  - Five levels:
    - *Initial* (chaotic, ad hoc, individual heroics) - the starting point for use of a new or undocumented repeat process.
    - *Repeatable* - the process is at least documented sufficiently such that repeating the same steps may be attempted.
    - *Defined* - the process is defined/confirmed as a standard business process
    - *Capable* - the process is quantitatively managed in accordance with agreed-upon metrics.
    - *Efficient* - process management includes deliberate process optimization/improvement.
- Practical meaning may be questioned, but there has been many followers.

Tampereen yliopisto
Tampere University

Maturity model for CD

(https://www.urbancode.com/resource/continuous-delivery-maturity-model/ was at developer.ibm.com)

- **Base**: The base level is enough to "be on the model". The team has left fully manual processes behind.
- **Beginner**: At the beginner level, the team is trying to adopt some ECD practices in earnest but is still performing them at a rudimentary level.
- **Intermediate**: Practices are somewhat mature and are delivering fewer errors and more efficiency. For many teams, Intermediate practices may be sufficient.
- **Advanced**: doing something well beyond what most of the rest of the industry and is seeing a great deal of efficiency and error prevention as a result.
- **Extreme**: Elements within the Extreme category are ones that are expensive to achieve but for some teams should be their target. Put another way, most organizations would be crazy to implement them, while this minority would be crazy to not implement them.

Tampereen yliopisto
Tampere University

Another
(https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model/)

Base … started to prioritize work in backlogs, have some process defined which is rudimentarily documented and developers are practicing frequent commits into version control.

Beginner … teams stabilize over projects and the organization has typically begun to remove boundaries by including test with development. Multiple backlogs are naturally consolidated into one per team and basic agile methods are adopted ….

Intermediate … extended team collaboration when e.g. DBA, CM and Operations are beginning to be a part of the team or at least frequently consulted by the team. Multiple processes are consolidated and all changes, bugs, new features, emergency fixes, etc, follow the same path to production. Decisions are decentralized to the team and component ownership…

Advanced … team will have the competence and confidence it needs to be responsible for changes all the way to production. Continuous improvement mechanisms are in place … releases of functionality can be disconnected from the actual deployment, which gives the projects a somewhat different role. A project can focus on producing requirements for one or multiple teams and when all or enough of those have been verified and deployed to production the project can plan and organize the actual release to users separately.

Expert …some organizations choose to make a bigger effort and form complete cross functional teams that can be completely autonomous. With extremely short cycle time and a mature delivery pipeline, such organizations have the confidence to adopt a strict roll-forward only strategy to production failures.

Another
(https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model/)

Base … one or more legacy systems of monolithic nature in terms of development, build and release. Many organizations at the base maturity level will have a diversified technology stack but have started to consolidate … to get best value from the effort spent on automation.

Beginner … the monolithic structure of the system is addressed by splitting the system into modules … this will also naturally drive an API managed approach to describe internal dependencies and also influence applying a structured approach to manage 3rd party libraries … importance of applying version control to database changes will also reveal itself.

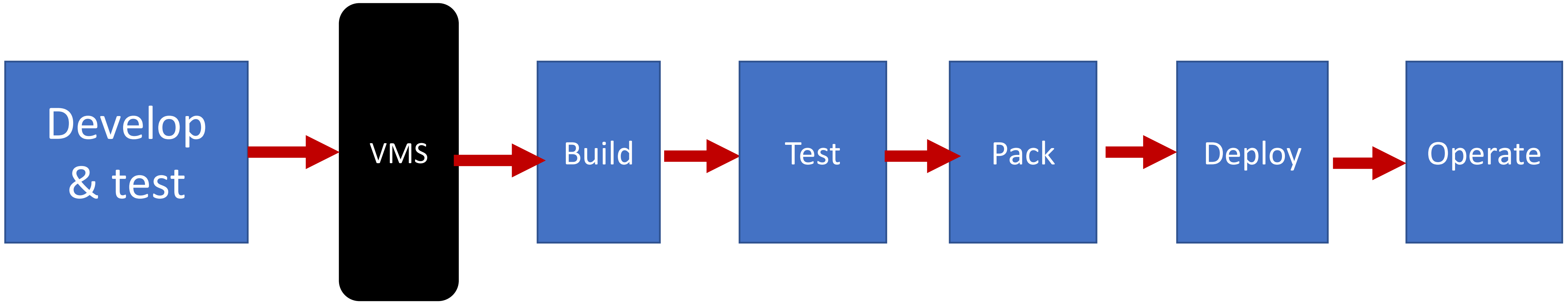
Intermediate. … a solid architectural base for continuous delivery … feature hiding for the purpose of minimizing repository branching to enable true continuous integration. … modularization will evolve into identifying and breaking out modules into components that are self-contained and separately deployed. … start migrating scattered and ad-hoc managed application and runtime configuration into version control and treat it as part of the application just like any other code.

Advanced. … split the entire system into self contained components and adopted a strict api-based approach to inter-communication so that each component can be deployed and released individually … every component is a self-contained releasable unit with business value, you can achieve small and frequent releases and extremely short release cycles..

Expert … some organizations will evolve the component based architecture further and value the perfection of reducing as much shared infrastructure as possible by also treating infrastructure as code and tie it to application components. The result is a system that is totally reproducible from source control, from the O/S and all the way up to application. …

Tampereen yliopisto
Tampere University

Simplified pipeline



C++

Python

Tampereen yliopisto
Tampere University

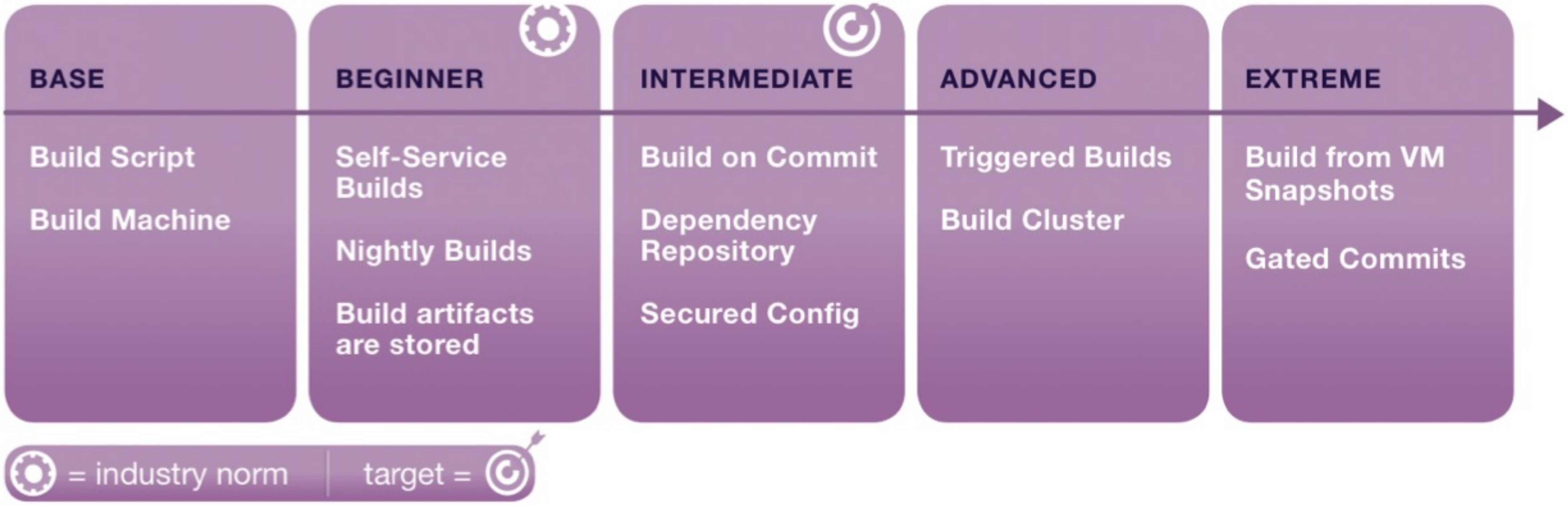
Build – which tools you know ?

- Make
  - Old
  - Declarative
  - Hard to debug
- Ant
  - Designed for Java
  - Based on XML-based configuration language
- Maven

Tampereen yliopisto
Tampere University

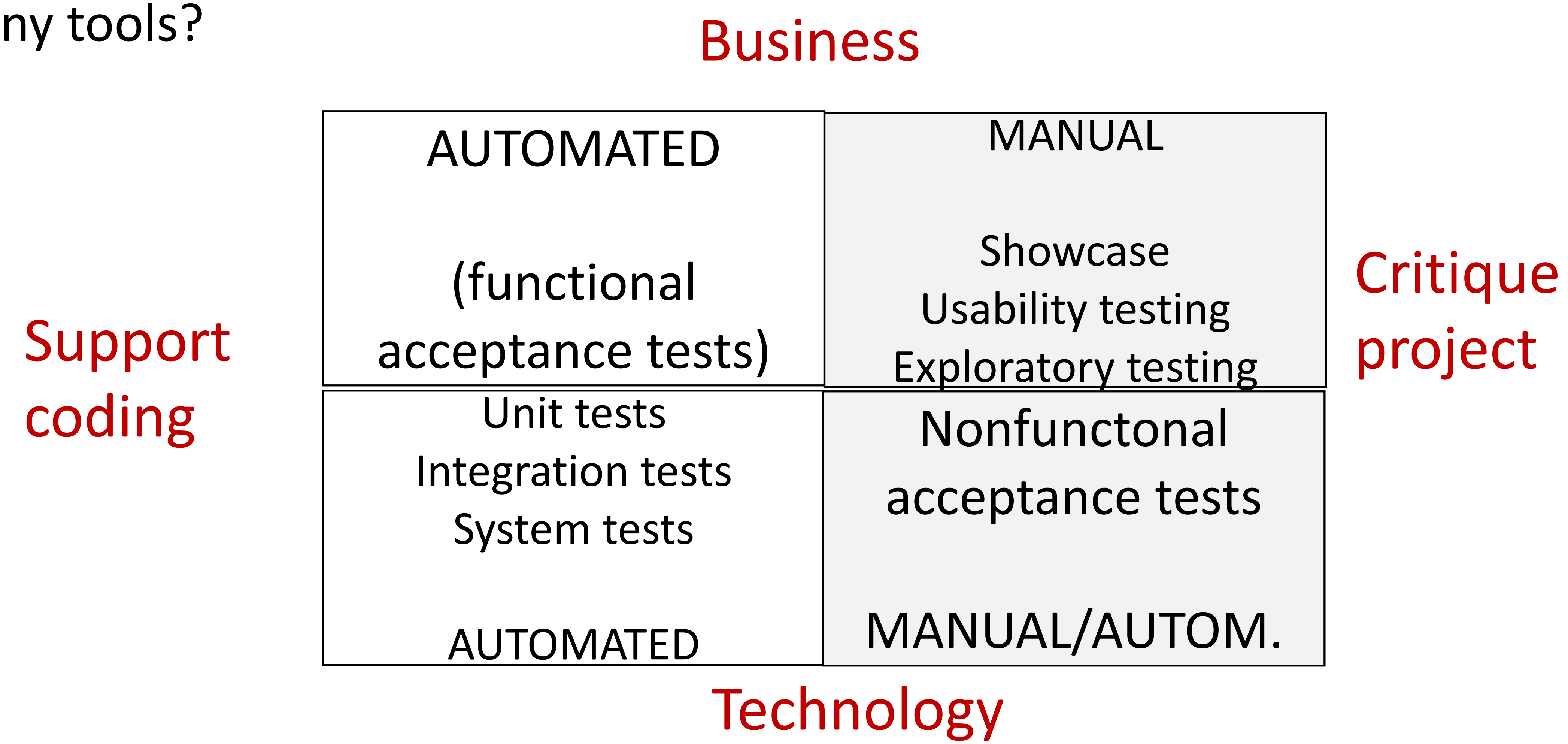
(https://www.urbancode.com/resource/continuous-delivery-maturity-model/
was at developer.ibm.com)

BUILDING



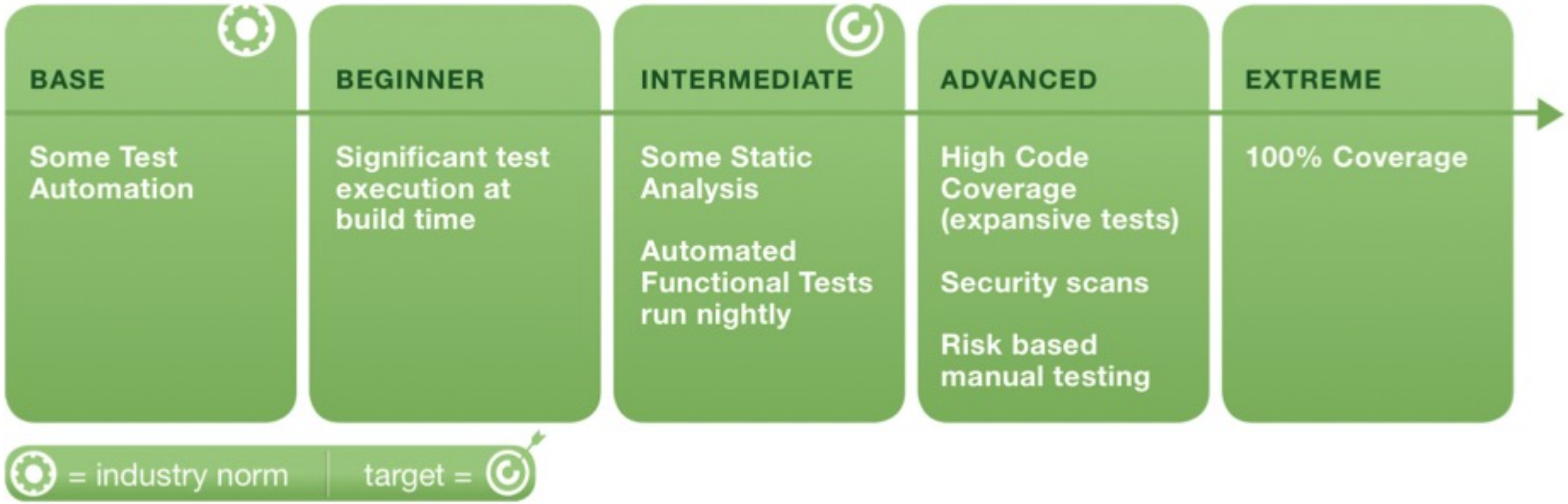
Testing

- Automate, automate, automate
- Know any tools?



(https://www.urbancode.com/resource/continuous-delivery-maturity-model/ was at developer.ibm.com)

TESTING


BASE
Some Test Automation
BEGINNER
Significant test execution at build time
INTERMEDIATE
Some Static Analysis
Automated Functional Tests run nightly
ADVANCED
High Code Coverage (expansive tests)
Security scans
Risk based manual testing
EXTREME
100% Coverage
= industry norm
target =

Tampereen yliopisto
Tampere University

Pack

- Binaries
- Required Libraries
- Runtime (e.g. Python)
- Manifest file
- Help files
- Localization stuff
- …

- Examples
  - Windows install shield
  - Java JAR
  - Android APK

What else comes to mind?

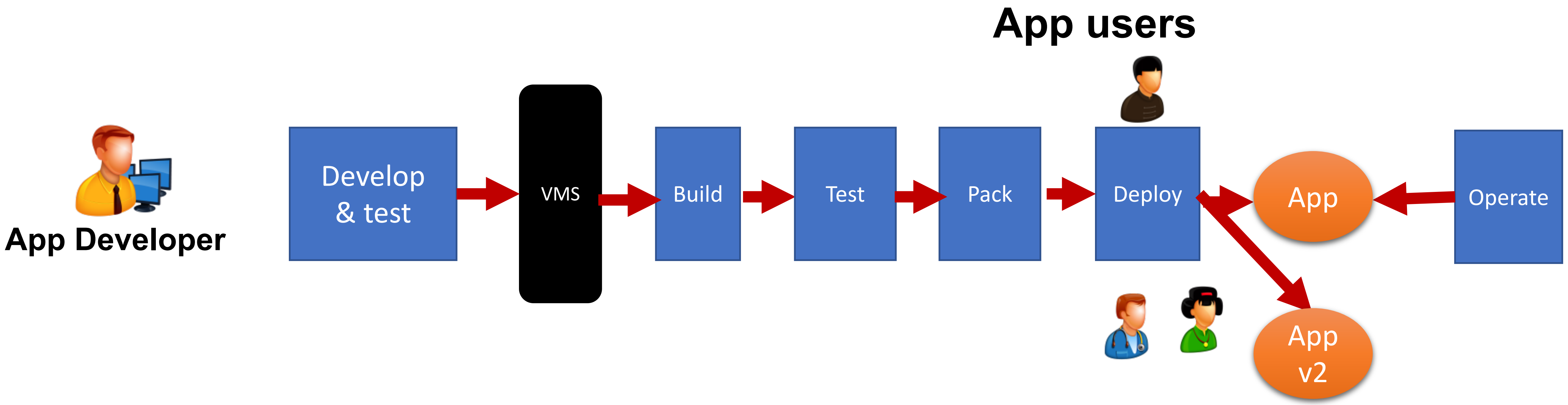
Tampereen yliopisto
Tampere University

Deployment/Delivery

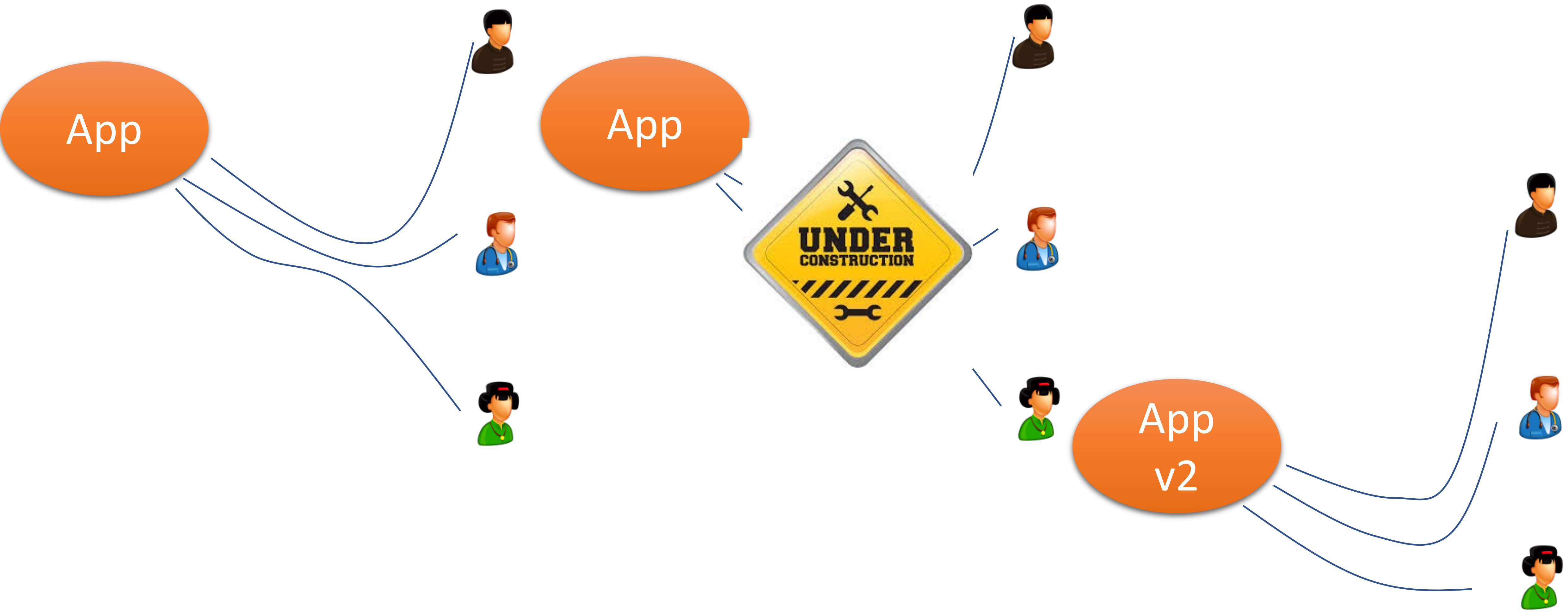
- Humble and Farley write
  - Creating the infrastructure (hardware, networking, middleware, …)
  - Installing correct version of the application
  - Configuring the application with its data

- Sounds a bit difficult?
- Text written before 2011
- First Docker release 2013

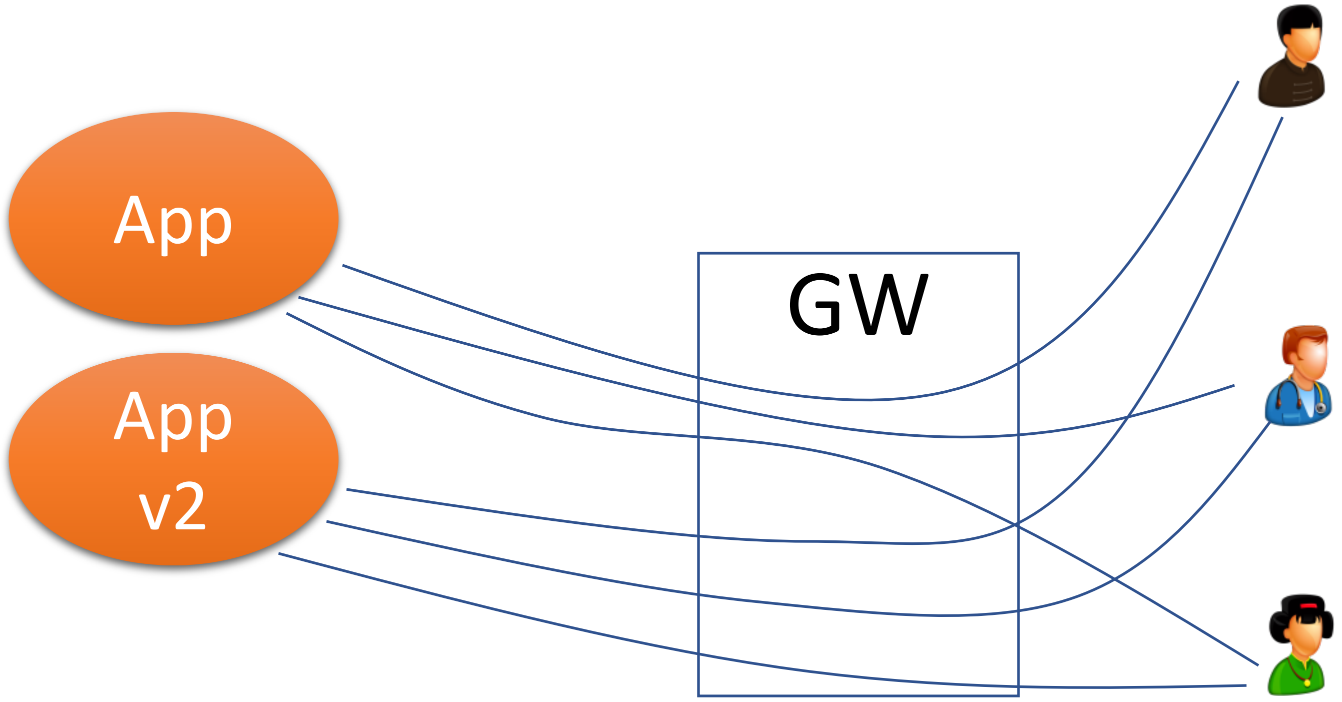
But when we have users



A possible strategy to deploy a new version?



Tampereen yliopisto
Tampere University

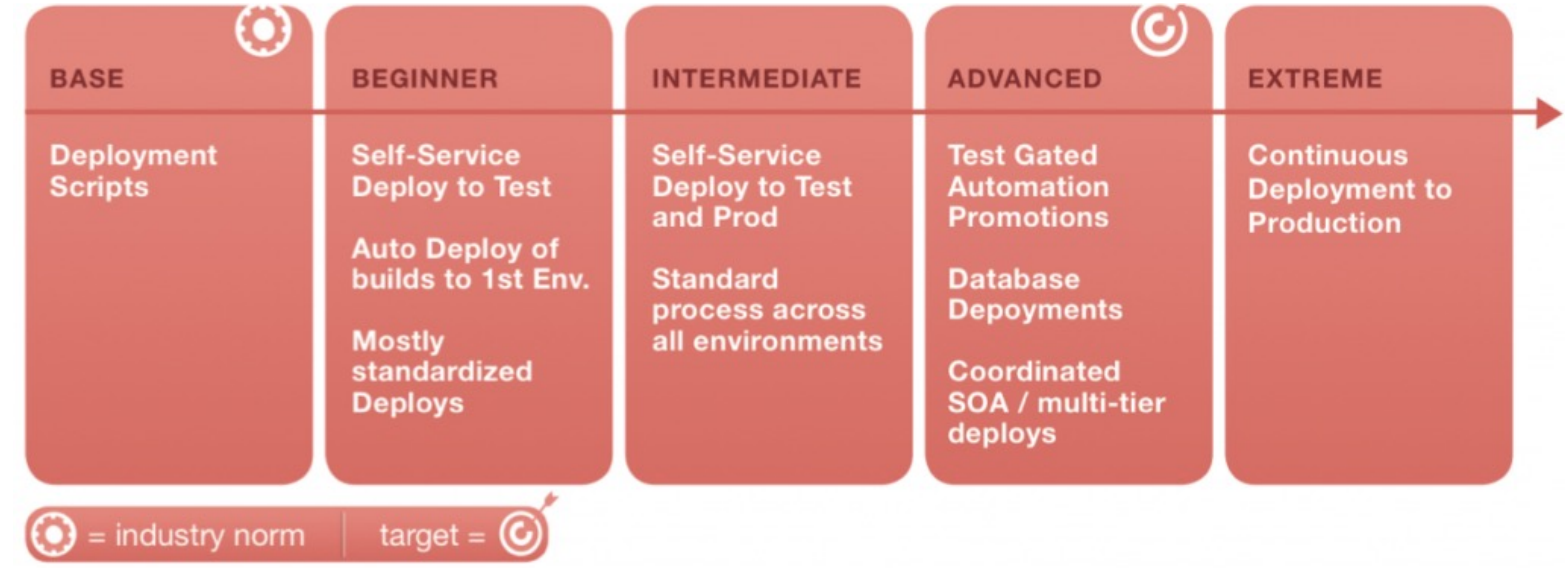


Problems & issues?

Tampereen yliopisto
Tampere University

(https://www.urbancode.com/resource/continuous-delivery-maturity-model/ was at developer.ibm.com) >

DEPLOYING

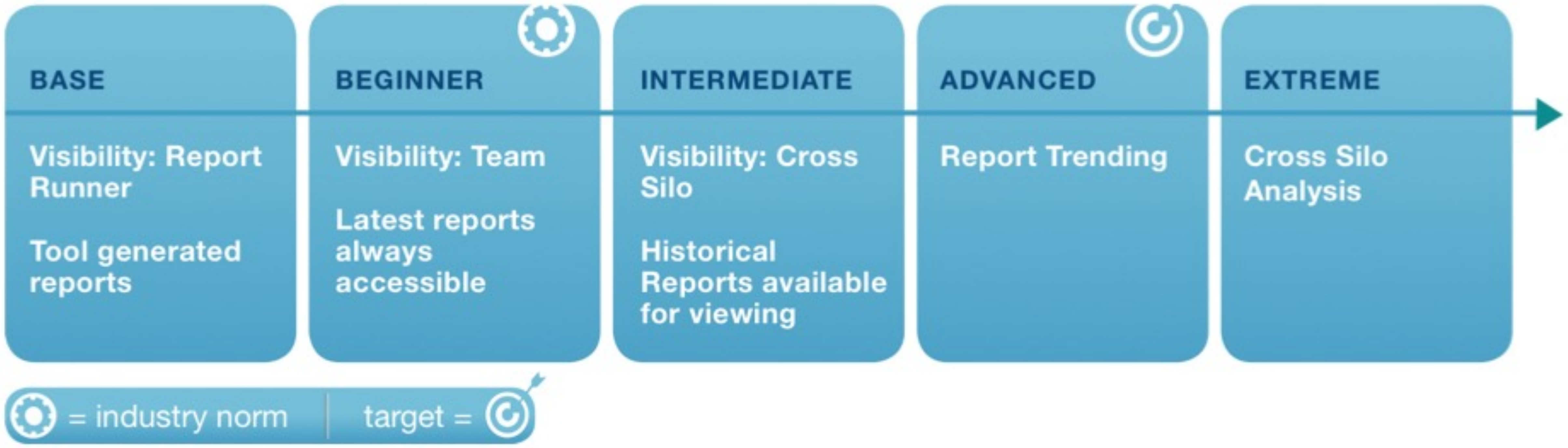


BASE	BEGINNER	INTERMEDIATE	ADVANCED	EXTREME
Deployment Scripts	Self-Service Deploy to Test Auto Deploy of builds to 1st Env. Mostly standardized Deploys	Self-Service Deploy to Test and Prod Standard process across all environments	Test Gated Automation Promotions Database Depoyments Coordinated SOA / multi-tier deploys	Continuous Deployment to Production

⚙ = industry norm | target = ◎

(https://www.urbancode.com/resource/continuous-delivery-maturity-model/ was at developer.ibm.com)

REPORTING

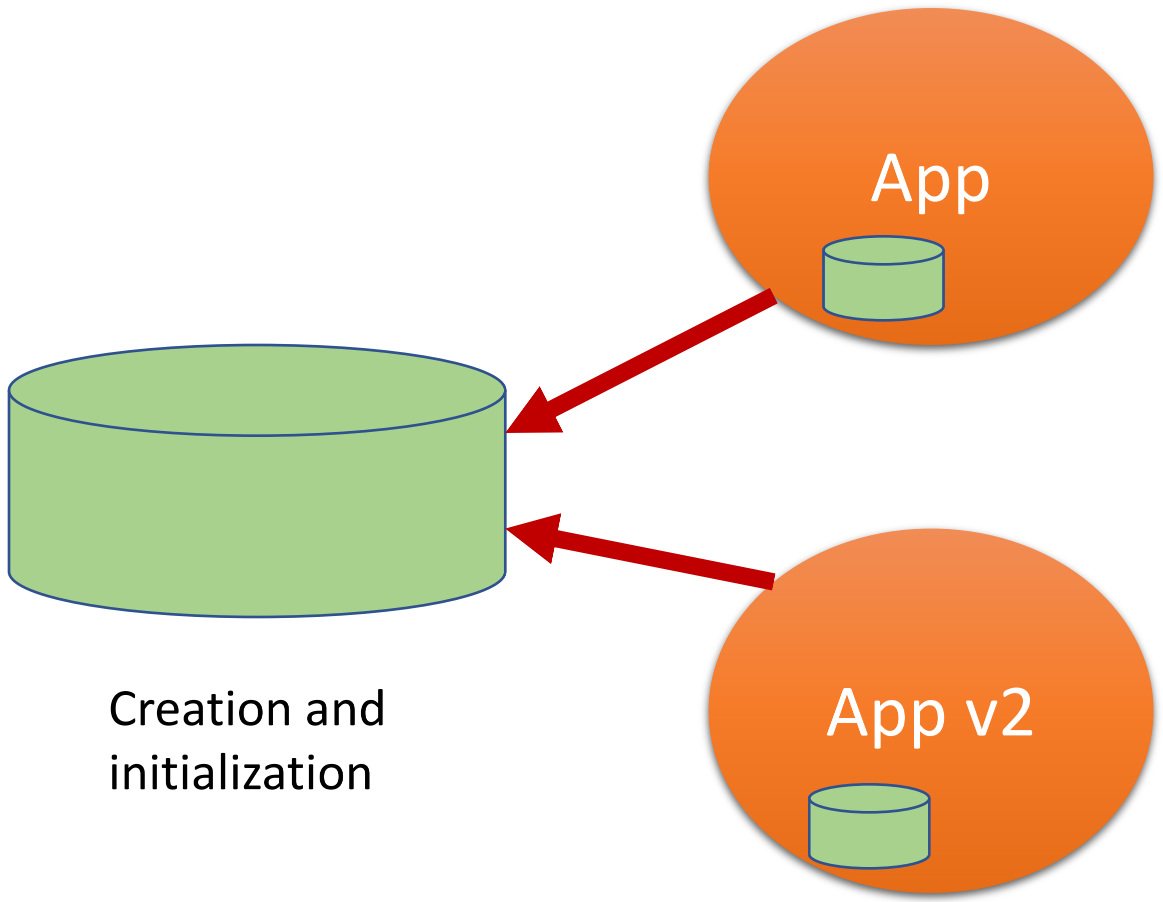


Deployment strategies

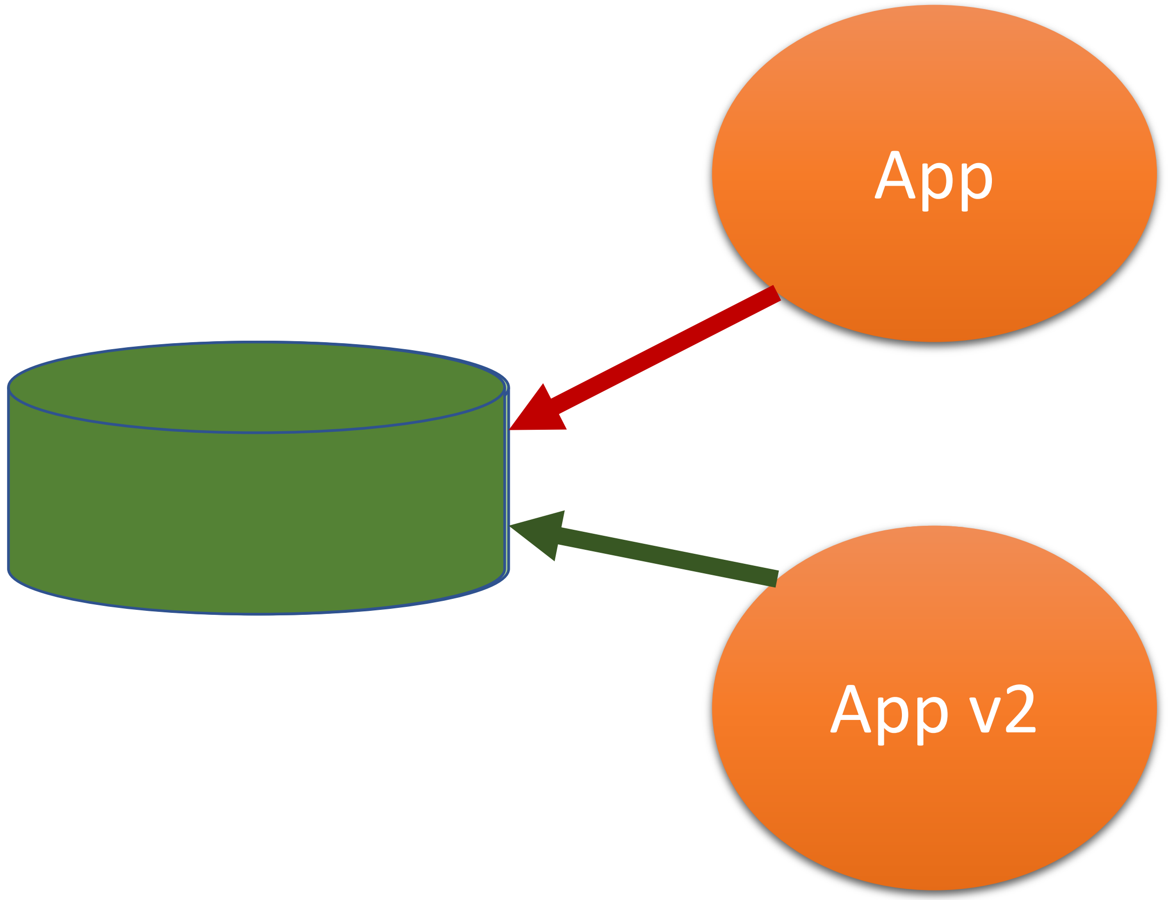
- Basic Deployment (aka Suicide) (https://harness.io/2018/02/deployment-strategies-continuous-delivery/) all nodes are updated at the same time
- Rolling Deployment (https://harness.io/2018/02/deployment-strategies-continuous-delivery/) nodes are updated incrementally,
- BlueGreenDeployment (http://martinfowler.com/bliki/BlueGreenDeployment.html) uses a router of incoming traffic as the tool. In this approach the new version (called green) is set up in parallel with the current (blue). When new (green) is ready, the router is switched to new (green) and blue is left as a backup. If something goes wrong with new, the router can be switched back to old - that means easy "rollback".
- Canary Releases (http://martinfowler.com/bliki/CanaryRelease.html) implements the deployment incrementally. In this case the router first directs only part of the customers to the new version. If feedback is is good, the other customers are moved to new version, too

Tampereen yliopisto
Tampere University

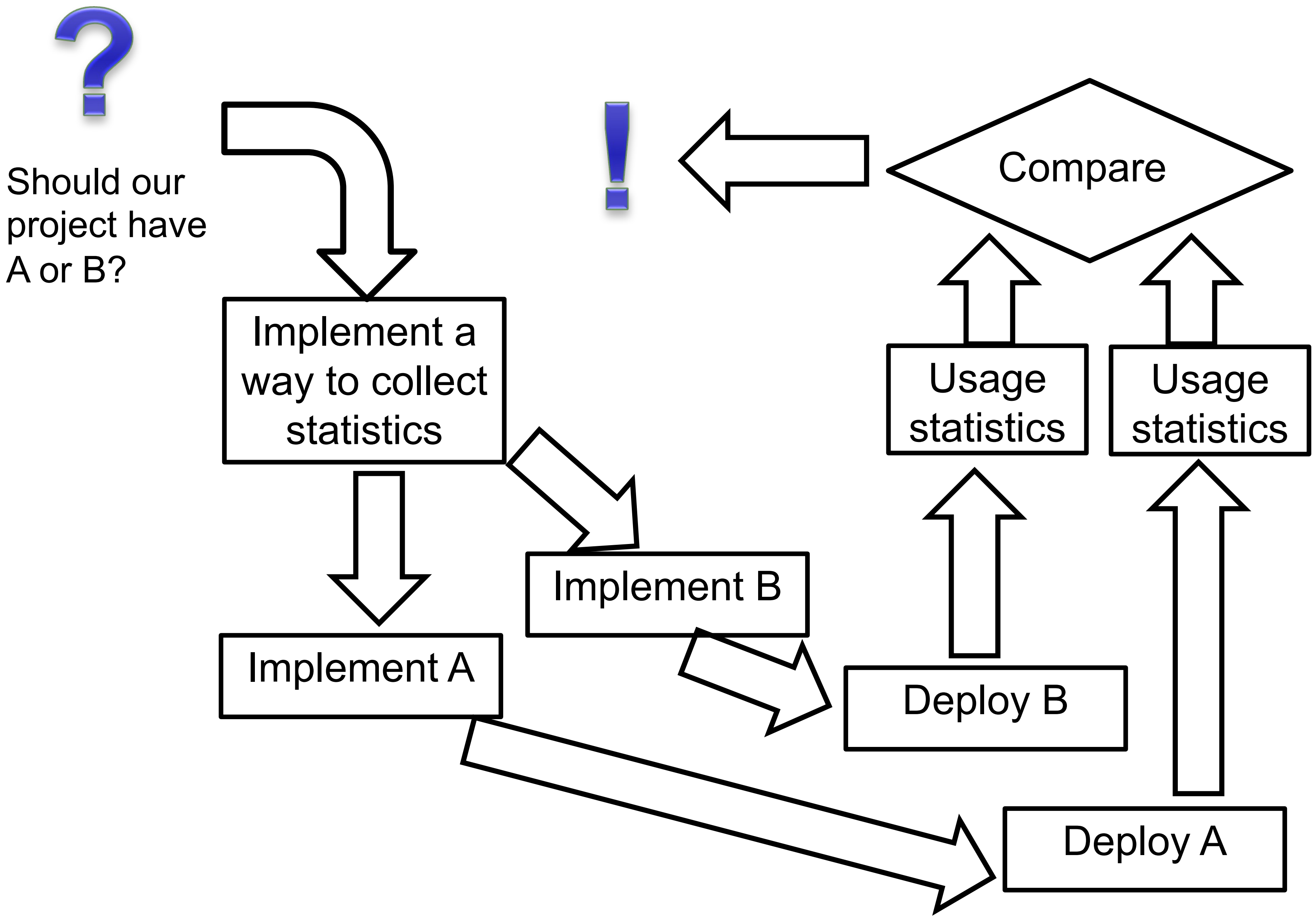
How about the data?



Creation and
initialization

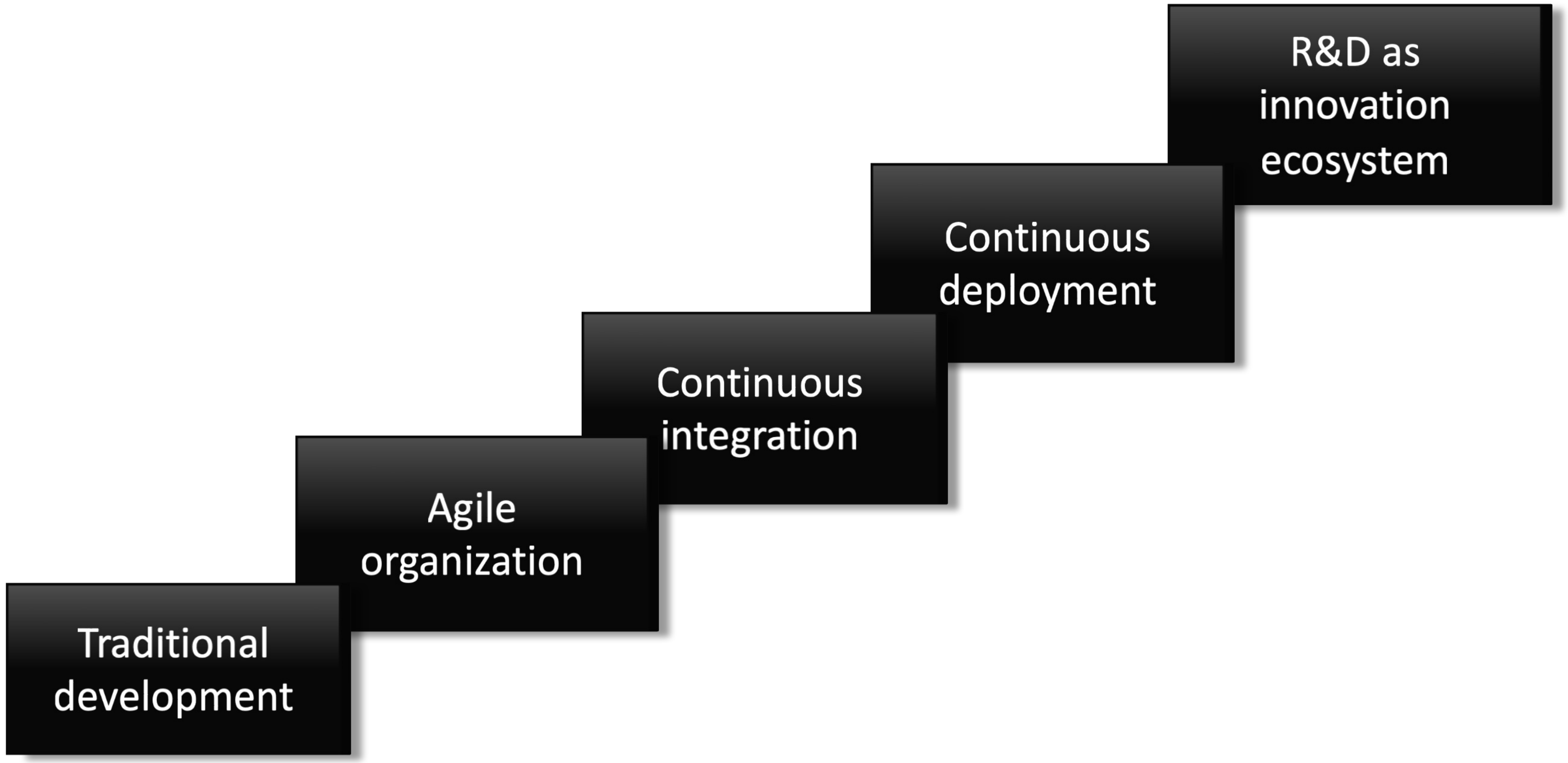


A/B Testing


?
Should our project have A or B?
!
Compare
Implement a way to collect statistics
Implement B
Implement A
Usage statistics
Usage statistics
Deploy B
Deploy A

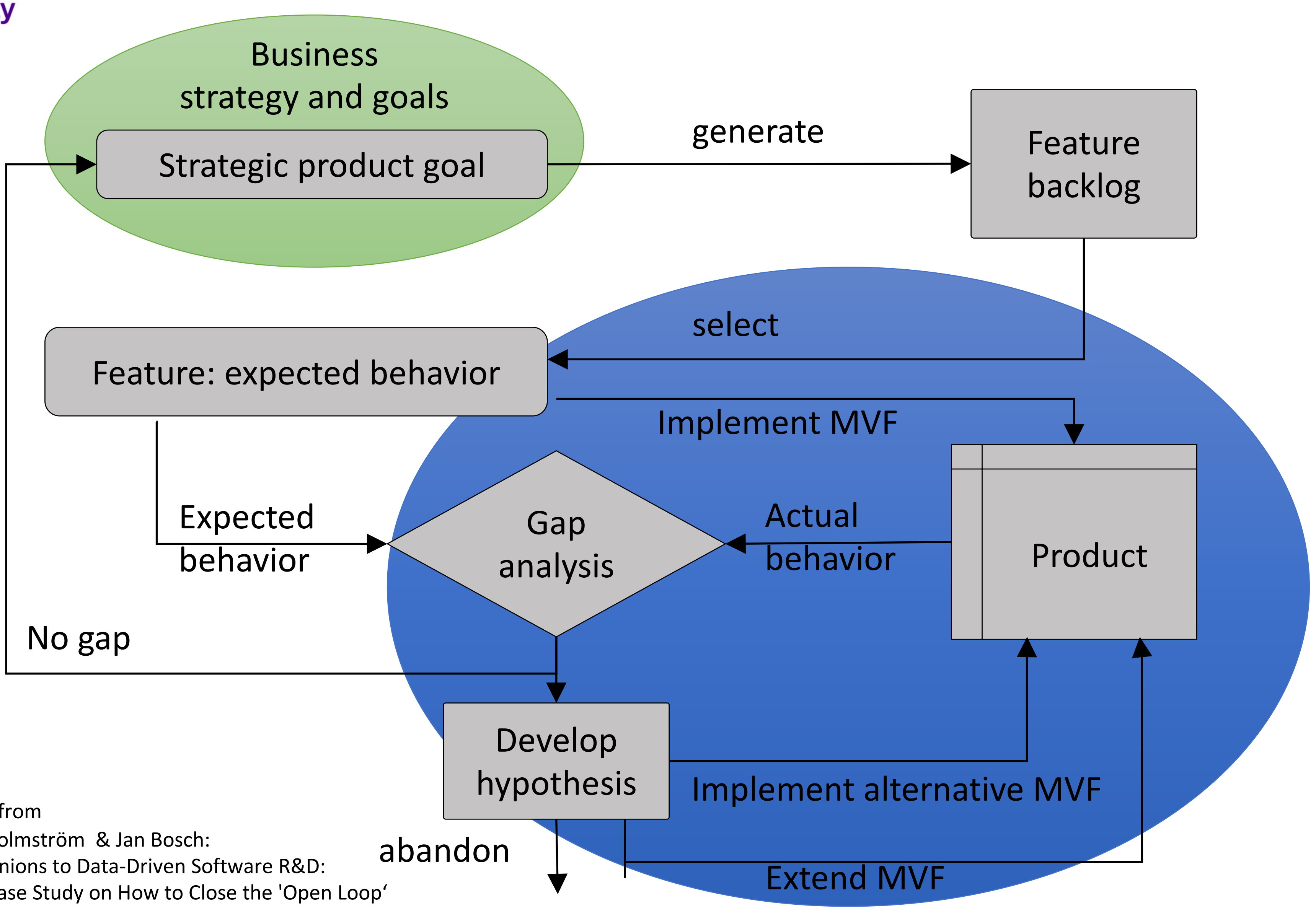
Tampereen yliopisto
Tampere University

Stairway to Heaven (As described by Jan Bosch)



Tampereen yliopisto
Tampere University

The HYPEX model (**Hypothesis Experiment Data-Driven Development)**

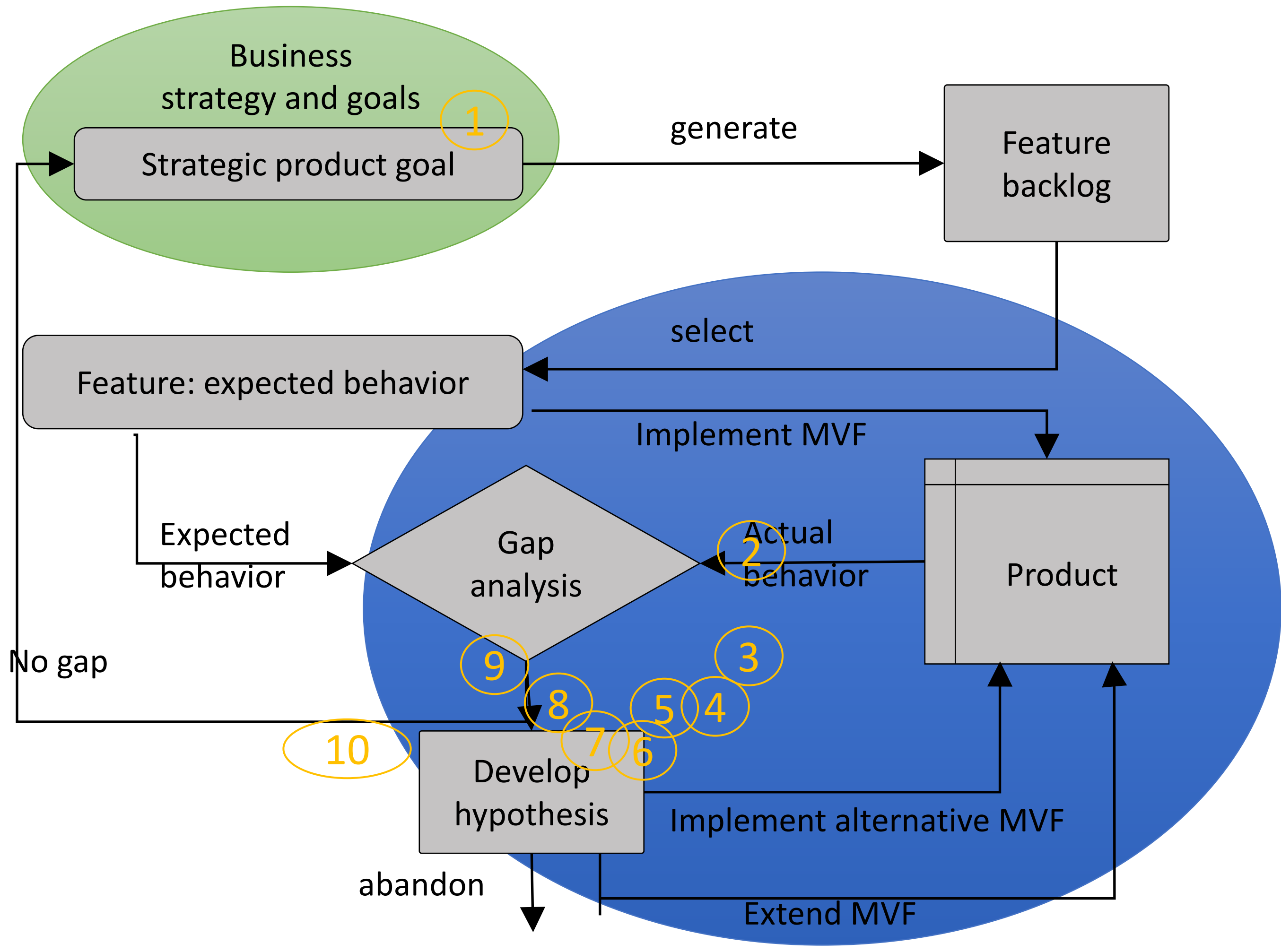


Adopted from
Helena Holmström & Jan Bosch:
From Opinions to Data-Driven Software R&D:
A Multi-case Study on How to Close the 'Open Loop'
Problem

Tampereen yliopisto
Tampere University

Data-driven software development

1. Planning of the data collection
2. Deployment of data collection
3. Monitoring of the applications
4. Picking up the relevant data
5. Pre-processing – filtering and formatting – the data
6. Sending and/or saving the data
7. Cleaning and unification of the data
8. Storing the data
9. Visualizations and analysis
10. Decision making


Business strategy and goals
Strategic product goal
1
generate
Feature backlog
select
Feature: expected behavior
Implement MVF
Expected behavior
Gap analysis
Actual behavior
2
Product
No gap
9
3
8
5
4
7
6
10
Develop hypothesis
Implement alternative MVF
abandon
Extend MVF