

20.09.2017



eficode

VOLVO GROUP IT: XE BPM

DEVOPS DEVELOPMENT REPORT

INDEX

- 1. INTRODUCTION**

- 2. SUMMARY**

- 3. KEY OBSERVATIONS**

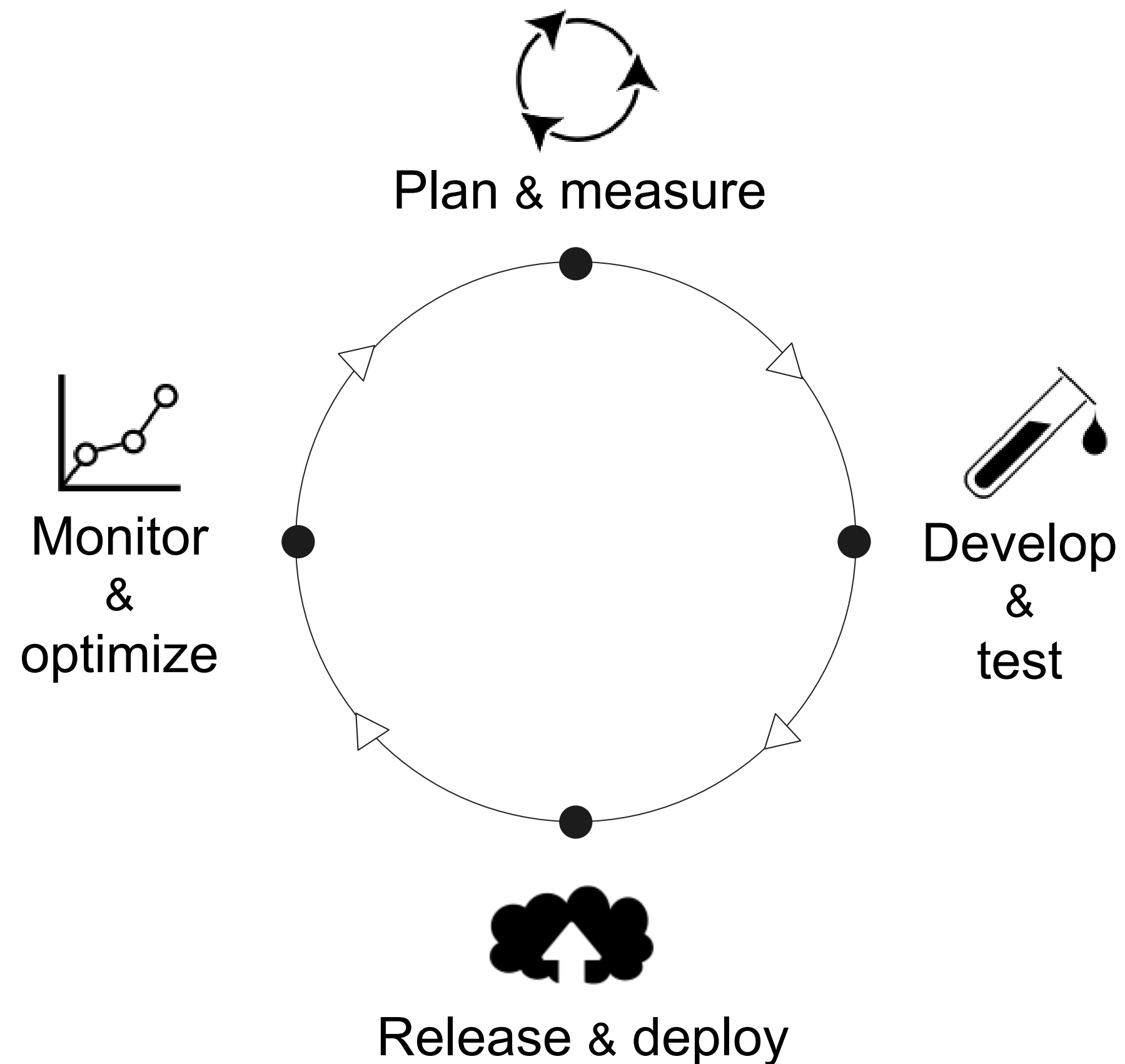
- 4. IMPROVEMENT SUGGESTIONS,
BENEFITS & ROADMAP**

- 5. APPENDIXES**

INTRODUCTION

DEVOPS IN A NUTSHELL

Devops enables continuous software delivery

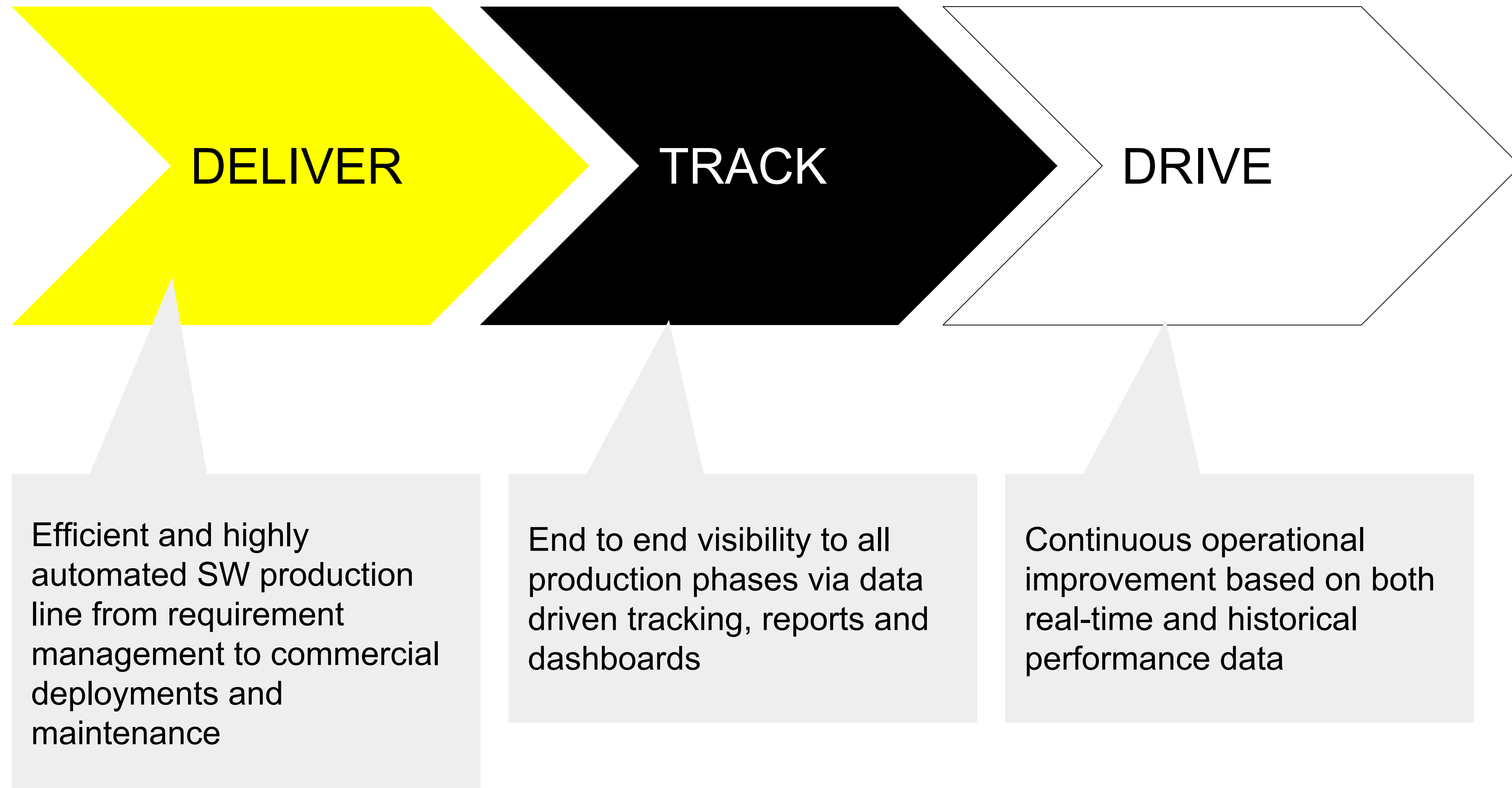


Faster time-to-value

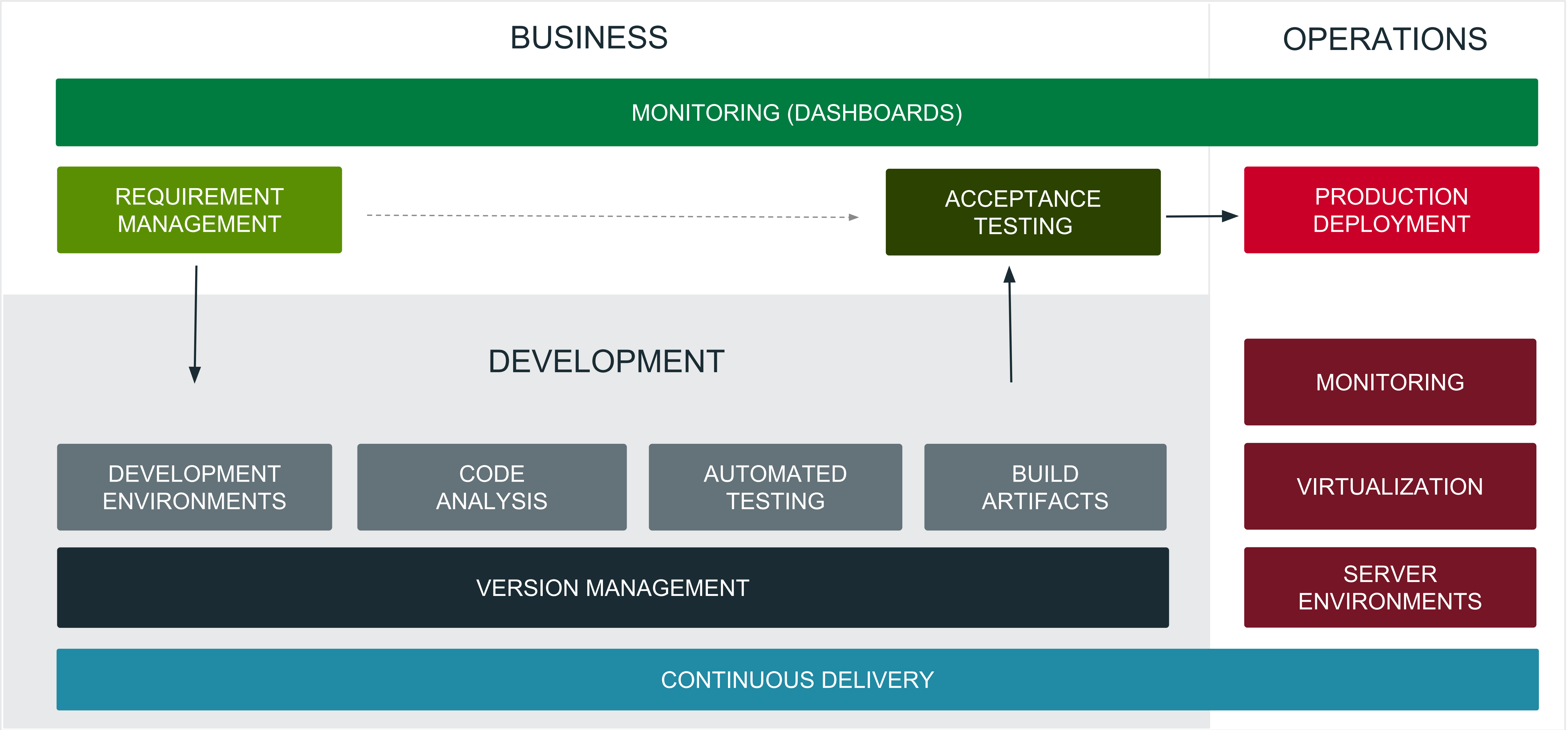
More time to innovate and improve

Improved customer experience

DEVOPS TRANSFORMATION JOURNEY



MODERN SOFTWARE DEVELOPMENT MODEL



SUMMARY

BACKGROUND

- Eficode performed a Devops pre-study with Volvo Group IT's XE BPM team
 - 10 people with varying backgrounds and different job descriptions were interviewed.
 - A person from the DRS team was interviewed to learn about the concepts of Volvo Production System for IT (VPS4IT) project.
 - Background material and XE BPM application architecture was reviewed.
- Target was to assess the current state of Devops implementation and identify areas for improvement
 - Focus on how to increase the product's quality, enhance development and optimize development practices within the Team.
- The evaluation is based on Eficode's maturity model which includes:
 - Devops maturity
 - Automation maturity
 - Test automation maturity


BACKGROUND

- About the Volvo Group IT XE BPM software product:
 - The product has a relatively long history.
 - Architecture is monolithic with dependency to third party software (MEGA).
 - Serving internal customers valuable information about the complex application structure Volvo has.
- The Team is located in two locations
 - Main development activities done in Bangalore, business analyst in Gothenburg.



VOLVO GROUP IT KPIs:

Reduce Lead-time
Increase Quality



DEVOPS MATURITY - 29 / 100

FOR XE BPM



	001	002	003	004
LEADERSHIP	Development operations have been separated from the business knowledge. Starting a new development project is laborious.	Starting new development projects is agile, and there are practices in place for steering the project.	New projects can be connected to organization's strategic targets. Starting a new pilot project is easy.	Real-time metrics supporting decision making and tracking the completion of strategic targets are available.
ORGANIZATION AND CULTURE	Design, development and quality assurance are separate from each other. Communication is primarily in writing.	Work is conducted in teams but development and quality assurance are separate from each other.	The teams work independently. They have total liability for the development and quality assurance of features.	The teams communicate with each other regularly and work together to improve their practices. Communication with the IT operations is continuous.
ENVIRONMENTS AND RELEASE	Products are environment-specific and they are compiled manually. Environments are installed and configured manually.	The system is divided into parts and the compiling environment is known. Some releases are automated.	Environments can be installed and configured automatically. Build and release processes are automated.	Releases may be conducted automatically and continuously. Migration and recovery processes work as expected.
BUILDS AND CONTINUOUS INTEGRATION	Product integration is automatic, but configuration and deployment are controlled manually. No artifact or change logs management.	The process starts team-specifically after every change. Tools are shared. Integration does not involve testing.	Integration covers the entire product and it is connected to acceptance testing. Dependencies are known and managed.	Build and integration processes are continuously improved based on collected metrics with aim to speed up the feedback cycle and improve visibility.
QUALITY ASSURANCE	Quality assurance is conducted completely by hand and primarily after development.	Unit testing or static code analysis is in place for some parts of the product.	Features visible to the end users are covered with automatic tests. Testers participate in the development process.	Acceptance tests present system requirements clearly and guide the development of the system as much as possible.
VISIBILITY AND REPORTING	Reports are made by hand when necessary.	Code integration, unit testing and code analysis are visible to the team.	The status of requirements can be monitored in real time in relation to tests and released features.	Real-time metrics are automatically collected from the product development process and used as a basis for improvement.
TECHNOLOGIES AND ARCHITECTURE	Technologies and tools are obsolete or are not fit for current requirements.	Technologies are growing old and the architecture is only partially adaptive or the interfaces are lacking.	Technologies are modern or well supported. The interfaces are well documented and exist for all key functionalities.	The architecture and technologies are optimal and enable reaching business targets efficiently.

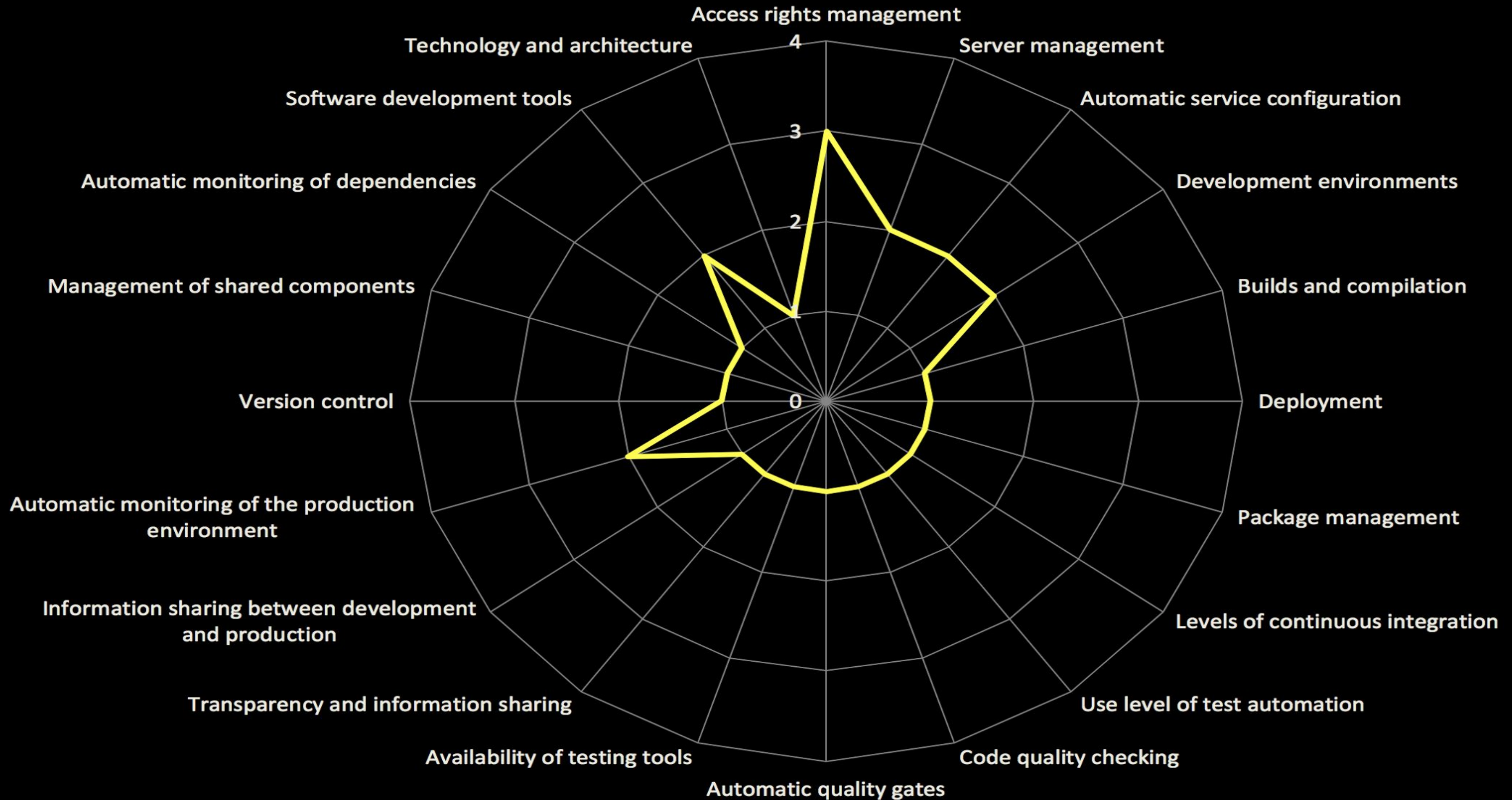
DEVOPS MATURITY - 32 / 100



AVERAGE FOR HEAVY INDUSTRIES

	001	002	003	004
LEADERSHIP	Development operations have been separated from the business knowledge. Starting a new development project is laborious.	Starting new development projects is agile, and there are practices in place for steering the project.	New projects can be connected to organization's strategic targets. Starting a new pilot project is easy.	Real-time metrics supporting decision making and tracking the completion of strategic targets are available.
ORGANIZATION AND CULTURE	Design, development and quality assurance are separate from each other. Communication is primarily in writing.	Work is conducted in teams but development and quality assurance are separate from each other.	The teams work independently. They have total liability for the development and quality assurance of features.	The teams communicate with each other regularly and work together to improve their practices. Communication with the IT operations is continuous.
ENVIRONMENTS AND RELEASE	Products are environment-specific and they are compiled manually. Environments are installed and configured manually.	The system is divided into parts and the compiling environment is known. Some releases are automated.	Environments can be installed and configured automatically. Build and release processes are automated.	Releases may be conducted automatically and continuously. Migration and recovery processes work as expected.
BUILDS AND CONTINUOUS INTEGRATION	Product integration is automatic, but configuration and deployment are controlled manually. No artifact or change logs management.	The process starts team-specifically after every change. Tools are shared. Integration does not involve testing.	Integration covers the entire product and it is connected to acceptance testing. Dependencies are known and managed.	Build and integration processes are continuously improved based on collected metrics with aim to speed up the feedback cycle and improve visibility.
QUALITY ASSURANCE	Quality assurance is conducted completely by hand and primarily after development.	Unit testing or static code analysis is in place for some parts of the product.	Features visible to the end users are covered with automatic tests. Testers participate in the development process.	Acceptance tests present system requirements clearly and guide the development of the system as much as possible.
VISIBILITY AND REPORTING	Reports are made by hand when necessary.	Code integration, unit testing and code analysis are visible to the team.	The status of requirements can be monitored in real time in relation to tests and released features.	Real-time metrics are automatically collected from the product development process and used as a basis for improvement.
TECHNOLOGIES AND ARCHITECTURE	Technologies and tools are obsolete or are not fit for current requirements.	Technologies are growing old and the architecture is only partially adaptive or the interfaces are lacking.	Technologies are modern or well supported. The interfaces are well documented and exist for all key functionalities.	The architecture and technologies are optimal and enable reaching business targets efficiently.

Automation maturity - 34/100



KEY OBSERVATIONS

CURRENT STATE

LEADERSHIP

- Mission is quite clear
 - To deliver business' needs with good quality
 - Could be more ambitious. For example, you could ask the possible questions: What are we aiming to become, what are we working for in the bigger picture?
 - Clear mission is one of the cornerstones for motivation and helpful for a project in maintenance mode.
 - Mission is also important when prioritizing work.
- Some level of mastery and autonomy
 - Management trusts that the Team will succeed
 - Good domain knowledge and insight knowledge of the 3rd party software (MEGA), working together to solve issues
 - Funding process limits enhancement possibilities
 - Business/Sweden prioritizes the work

CURRENT STATE

ORGANIZATION & CULTURE

- Operational model reflects the organizational model
 - Organization divides teams to developers, testers and analysts
 - Culture encourages responsibility division according to organizational model
 - Sub-teams within the Team take responsibility from part of the flow but not for the entire project
- Culture of fast gains
 - Work/task prioritization: new features and manual testing over automated testing, refactoring and code reviews
 - Trying out improvements/new things requires a big effort
- Mixed culture of Agile and Waterfall
 - Ways of working come from top to down
 - Release cycle defined by the management
 - Team utilizes some attributes of Scrum and VPS4IT

CURRENT STATE

ORGANIZATION & CULTURE

- Information shared mainly in discussions
 - Essentials are shared in daily huddle meetings
 - On-demand discussion in person or by utilizing Skype
 - Communication is efficient but information only accessible to participants in discussions
- High level of trust in each other - maybe even blind trust
 - Code reviews are a manual action executed by one person
 - No sharing of responsibility - either he did it or I did - there is no We in doing
- Experimentation culture is not available - comfort zone activities
 - From team side weak activeness to push for experiments
- Open communication also with management

CURRENT STATE

PROCESSES

- Segregated flow
 - Requirements' definition, development and testing activities are segregated
 - After development you wait for production
 - Always in this order
 - Handovers between flow
 - Individual responsibility of CRs
- Mixed culture of Agile and Waterfall
 - Daily standups with physical board to visualize the current state of software development activities
 - Quite long meeting for requirement breakdown
 - Separate meetings to solve why someone is stuck with a problem
 - “Lessons learned” meeting after releases

CURRENT STATE

PROCESSES

- Definition of done is unclear
 - The process is not written in details anywhere from it could be utilized by everyone.
- Requirement process is not visible
 - You have the people for requirement breakdown
 - Currently utilized Teamplace does not support TRACK-phase data collection
 - Requirement enhancements/new features do not affect test cases' content
 - CR acceptance criteria and details are not written down
- Testing is not identified as a first class citizen
 - Acceptance criteria is not clear and customers have different views to CRs
 - Definition of Done does not include definitions of testing.

CURRENT STATE

ENVIRONMENTS & RELEASES

- Mutable servers managed by HCL
 - Fragile approach by design.
- Process for updating Windows servers
 - HCL has the maintenance responsibility.
- Test environment deployed with a weekly cycle
 - Technical ability to deploy separately after every CR.
- CI server is missing from the environment
 - Essential in an CI/CD environment
- The QA environment is really slow and it does not provide good user experience.
 - Requires root cause analysis and maybe some environment changes.

CURRENT STATE

ENVIRONMENTS & RELEASES

- Big and slow releases
 - 4 major releases per year
 - Deadline driven releases
 - CRs stay in “ready for prod” state for long time - has a big impact on lead times
 - Business is happy with the slow release pace - hard to improve if there is no will for improvement
- Inflexible and slow process
 - Post release emergency additions are possible, but not often
 - Maintenance window required by HCL
 - 2 weeks UAT before production
- Well defined release notes are missing
 - Comparison examples:
 - Git’s release note for version 2.15.0: <https://github.com/git/git/blob/master/Documentation/RelNotes/2.15.0.txt>
 - GitKraken’s release note for the latest release: <https://support.gitkraken.com/release-notes/current>

CURRENT STATE

BUILDS & CONTINUOUS INTEGRATION

- Requirements for efficient continuous integration:
 - Must have production like environment with production like data
 - Must be able to deploy all new changes and verify quality of new and existing features
 - Standardized, automatic build and deployment processes
 - Quality check needs to be swift - defines feedback loop length
- Current status:
 - There is no dedicated CI server
 - Scheduled import and export scripts
 - Current environment backup process from PROD takes several hours
 - Deployments to PROD environment happens 4 times a year, to TEST once a week
 - Slow, manual verification of new software versions

CURRENT STATE

QUALITY ASSURANCE

- CR acceptance criteria is discussed during analyzation meetings but details are not very well documented
 - Visibility to testing activities is questionable.
- Possible major risks because of human errors
 - Test automation is not utilized to give fast feedback and drive development activities.
- Last minute corrections to the software application make the overall status vague
- Root cause investigation activity is missing
- Testers' main responsibility should be to update automated test sets, not to “test that stuff still works”.
 - The value of automated tests suffer from culture, ways of working and tools' utilization.

CURRENT STATE

QUALITY ASSURANCE

- Testing is done mostly after development activities are finalized
 - CI environment and test automation are not utilized to verify each build.
- Test cases are not versioned in the same repository with the application software.
- Testing activities are not documented accurately
 - Some test cases exist, but they are not updated and maintained
 - Details missing, human errors are possible
 - Separate tests suites are missing
 - Smoke, regression, release testing, ..
 - Product's overall status is unknown for a long time
 - Test reports are missing

CURRENT STATE

QUALITY ASSURANCE, VISIBILITY & REPORTING

- Level of test automation is low/no test automation
 - Hard to define what kind of overall changes each new CR introduces.
 - Few automated test cases exists, but they are not utilized.
- The value of high-quality test reporting is not utilized
 - Each test execution activity must generate a report that accurately describes the current state of the XE BPM's application that used in testing activities.
 - Test report should be the single source of truth in quality assurance.
 - Usually test automation tools provide a good and easy way to produce test reports that are similar and easily comparable.

CURRENT STATE

QUALITY ASSURANCE, VISIBILITY & REPORTING

- No specific data collected from the Team's performance
 - Currently utilized Teamplace does not provide an easy way to analyze performance.
 - Data crunching is possible, but it does not tell the whole truth
- Monitoring relies completely on HCL
 - Application software does not include monitoring capabilities
- Working together with MEGA to analyze possible bottlenecks

CURRENT STATE

TECHNOLOGIES & ARCHITECTURE

- For version control, except for MEGA, SVN is in use
 - All code is not stored in version control.
- Disharmony concerning MEGA
 - Business is happy with MEGA
 - Release publication time takes 7-8 hours, dependent on MEGA
 - Performance issues with MEGA
 - Root cause investigation is hard without proper logging

WIDER ORGANIZATIONAL ISSUES

- Project thinking in the leadership
 - Most of the time software is a product, not a project
 - Considering a new product or a new version as a “project” is harmful
 - New version (project) might have a new manager
 - Old manager drops responsibility and moves on
 - Even if the manager is the same, pressure towards delivering well in the project scope
 - Encourages towards not dealing with any technical debt - instead just deliver obligatory items fast
- Results are wanted in the way of releases, but quality is not tracked at the same time through well documented data
 - Encourages deadline driven development and value determination
- Big and slow release process is a comfort activity

WIDER ORGANIZATIONAL ISSUES

- The UAT activities do not provide additional value to the release process
 - Increases lead time
 - Comfort operation for business
 - Could utilize frequently deployed releases to QA environment also for business' testing activities
- Separation of developers, testers and analyst
 - No end-to-end process - working as a team to have a CR moved to Done-column.
- All code should be in version control.
- Monolith deployments

IMPROVEMENT SUGGESTIONS, BENEFITS AND HIGH-LEVEL ROADMAP

SUGGESTIONS

#	Title	Difficulty to implement	Description
1	CI / Jenkins server	*	Widely known and well supported CI-server that can be used for various tasks and it integrates easily with different environments. Essential investment for building up a CI/CD pipeline.
2	SonarQube	*	A code quality analyzer tool that helps to improve and maintain the code quality. Integrate it to the CI/CD pipeline.
3	Take linters and other appropriate plugins into use	*	Helper tools used in software development.
4	Automate daily checkup of latest build state	*	There's a daily task to verify that the build was successful. It is required to check some files. This simple tasks should be automated to prevent human errors and bring visibility to the daily status.
5	Evaluate and take Jira into use	**	Get your tasks visualized and have team performance indicators/statistics.
6	Move to Kanban	**	A software delivery process that encourages to make tasks smaller, take the tasks quickly under development and release features when they become ready unless business specifically insists that they're not released until a specific date.

SUGGESTIONS

#	Title	Difficulty to implement	Description
7	Enable monitoring/tracking of services	***	Start collecting data in order to investigate root causes. This is the key to identifying and overcoming persistent performance issues.
8	Evaluate and take Git into use	**	Subversion does not support effective branching. Move your code to Git.
9	Define a workflow process	**	Take feature branching into use. All code and tests should be in version control, also the MEGA related code. Trigger SonarQube for every change you make in Git.
10	Definition of Done	*	Definition of done is an important tool that can help you support building good culture. If you define DoD as a team and agree when you're done with certain tasks, then there is clarity. Without Definition of Done people might have different perceptions of when they're done and process get's siloed into different stages.
11	Evaluate test automation frameworks	**	There are various test automation frameworks that can be used for automated testing. Define your needs and evaluate couple of different frameworks. Create a smoke test set with the frameworks you choose to evaluate.
12	Create automated test cases	**	Create automated tests cases with the chosen test framework(s) for different kinds of tests and test sets. Integrate it to Jenkins.

SUGGESTIONS

#	Title	Difficulty to implement	Description
13	Increase test coverage for automated acceptance testing	***	By utilizing the previously designed smoke and regression test sets you can go far. Aim for a 100% test coverage and fast feedback of the software quality with high-quality reporting.
14	Release notes	**	Release notes bring better visibility to the Team and customers.
15	Remove UAT from the release process	**	The UAT comfort testing does not bring value with automated acceptance testing. Instead, you should aim towards faster release cycle and if business are unhappy with some features they'll get fixes faster. In the current process they don't get anything until an agreed date.

BENEFITS

#	Title	Description
1	CI / Jenkins server	There are multiple tasks that can be automated. By automating the tasks and enabling Jenkins to execute the tasks you get as well reporting facilities and visibility to the current state.
2	SonarQube	Currently code quality checks are done mostly manually and randomly. SonarQube analyzes the code base and provides visibility to the code quality. It supports various programming languages and is usable also in this project.
3	Take linters and other appropriate plugins into use	Easy to take tools that improve ways of working and provide valuable information of the software being created.
4	Automate daily checkup of latest build state	Having the daily checkup tasks automated will fasten the process, provide accurate results and bring visibility to the whole team.
5	Evaluate and take Jira into use	Teamplace isn't very user-friendly providing information about the different statuses requirements are in. Additionally it doesn't bring out of the box various statistics and reports from the project. Jira provides improved visibility and various statistics out of the box.
6	Move to Kanban	The team has many CRs, some small some big. With Kanban they would be able to create and release CRs once the team has implemented and tested it.

BENEFITS

#	Title	Description
7	Enable monitoring/tracking of services	End users have reported of performance issues with the tool/site. Proper logging would enable tracking of the root cause. In addition utilizing HCL's monitoring services would show whether the problem is in the application or the environment being used.
8	Evaluate and take Git into use	Git supports lightweight branches, allowing you to efficiently work on feature branches. Apply a modern tool to store your code, for example Bitbucket, Github or Gitlab.
9	Define a workflow process	When everything is in version control you can manage and deploy software reliably. Having test and application source code in the same repository simplifies tracking of different versions, makes fixing bugs in test and application side easier. Additionally you will have the same version of tests and application side by side making debugging problems easier. With proper branching you can use pull requests as review points and triggers for CI. Having also the MEGA related code in your version control you are not dependent only on MEGA's version control and you can track reliably and in the same manner as all the other source code what kind of changes have been made.
10	Definition of Done	Well implemented and followed Definition of Done helps to track the changes and bring insight to the current state of the software. Everyone knows what needs to be done in different stages of the development process.
11	Evaluate test automation frameworks	Once you have an understanding of the various test frameworks that are available you can choose the one(s) that suite your the best. Things you need to consider are, for example, difficulty of implementation, test coverage, easiness to update/maintain test cases and suites and whether the framework(s) provide good reports that are easily usable/understandable also outside of the Team.
12	Create automated test cases	Once you have created multiple test sets you can, for example, get faster feedback of the software quality, improve visibility, have defined test sets for defined test activities. Start by improving the smoke test set and create, for example, a regression test set.

BENEFITS

#	Title	Description
13	Increase test coverage for automated acceptance testing	A good test coverage gives the Team and its customers a good overview of the changes happening with software. The Team can make quick fixes and release smaller changes with acceptable confidence faster. The Team can design appropriate test sets for different testing activities, for example, fast and long lasting test sets.
14	Release notes	Utilizing the well designed CI/CD-pipeline the Team will have valuable information about the changes in the software application, for example, number of CRs, the areas where Team has been working on, performance statistics etc. By utilizing the tracked information the Team can produce well prepared release notes that are valuable for the customers and bring visibility to the whole release process.
15	Remove UAT from the release process	As the Team is working in an fast and efficient way and can provide detailed information of its work for each release the UAT process becomes unnecessary. The Team can provide fast fixes to the problems and change its development direction as per requests. Testing activities provide well documented test reports and they are readable and understandable by the customer and the Team itself.

HIGH LEVEL ROADMAP

2017

2018

09

10

...

01

02

...

07

08

Phase 1

- CI / Jenkins server
- SonarQube
- Take linters and other appropriate plugins into use
- Automate daily checkup of latest build state
- Evaluate and take Jira into use
- Move to Kanban
- Enable monitoring/tracking of services

Phase 2

- Evaluate and take Git into use
- Define a workflow process
- Definition of Done
- Evaluate test automation frameworks
- Create automated test cases

Phase 3

- Increase coverage for automated acceptance testing
- Release notes
- Remove UAT from the release process

APPENDIX 1: AUTOMATION MATURITY MATRIX

Environments and builds				
Access rights management	Every tool has separate user management or actual access rights management is not in use.	Each tool has a separate user management, but persons in charge are documented or the channel for requests is common.	Shared tools are connected to the company's centralized access management. Tool specific access rights are handled with group management. The system enables centralized creation of non-personal integration accounts.	Group management is project specific and projects themselves can manage their groups. The system enables creation of light R&D accounts in under an hour.
Server management	Servers are requested by mail or a tool several weeks or months earlier. Access management is manual.	Acquiring servers is done by a tool, but getting them takes weeks. Acquisition is centered to specific persons. There's a process for documenting environments.	Servers can be requested instantly with an electronic tool.	Servers acquired for development work are destroyed automatically. Access management on development is connected to group management and is connected to other development tools.
Automatic service configuration	External services, such as firewalls, load balancers and data warehouses as well as servers are configured manually.	Servers are configured manually, but external services automatically	Services are mainly configured automatically	All components of the service are automated. The whole infrastructure of the service is described as code.
Development environments	Developers can choose their own tools, if they are free. Everyone installs their own tools and are responsible for their functioning themselves.	Every developer is given a specific tool set. The company pays for commercial tools, if they are used.	Every developer is given a specific tool set. Installation is documented and IT support gives full support for the installation. Based on their needs, developers can also use their own tools, if they're free. The company pays for commercial tools, if there's basis on purchasing the tool.	Creation of development environments is automated. Building a new environment to an empty machine is easy, fast and reliable.

Build and continuous integration				
Builds and compilation	<i>Version control and installation is not connected without manual steps.</i>	A compilation and or quality assurance is automatically started from version control, but not the packaging or the installation.	Distributable packages are developed automatically from version control to the artifact storage.	Version control changes passing quality assurance are automatically taken to installation.
Deployment	<i>Installation is done manually and few people are able to do them.</i>	Installation is documented and documentation is updated regularly.	Application update is automated and both utility program and configuration needs are documented.	Application and the infrastructure it needs are completely automated.
Package management	<i>No package management or differing conventions.</i>	Packages are stored in version control.	Package management has a separate storage system, where packages are stored versioned. Requirement relations are documented.	Packages are handled with system that is specialized to package management, which can be used to make queries about requirement relations.
Levels of continuous integration	<i>No continuous integration or loose conventions.</i>	Continuous integration is team specific and doesn't cover the whole product pipe.	Results of integration process of teams is available for other teams, but integration of the next level is not started automatically.	Team level integration, when successful, starts integration of the next level automatically and relevant teams get information about the integration results related to components they are responsible.

Quality assurance				
Use level of test automation	No automated tests (manual/no tests)	Smoke tests / regression tests for main features.	Large part of the application is tested.	Over 85% test coverage.
Code quality checking	No special code quality monitoring.	One of three in use: Static analysis, pair programming, code review	Two of three in use: Static analysis, pair programming, code review	All three in use: Static analysis, pair programming, code review
Automatic quality gates	No automatic quality gates.	Qualities are in the process definitions: code reviews done by people and reports done by testers act as quality gates. Coding style conventions are defined.	There's a review tool for code review and test automation produces automatic reports.	In addition to automated tests and possible code review, software is checked with static analysis with every version control commit.
Availability of testing tools	All testing is done manually by testers or automatic tests can't be run locally.	Smallish part of tests can be run on developers' machines or tests requiring external dependencies are not separated.	Most tests can be run on developers' machines. Tests are organized so that tests requiring externals dependencies can be easily left not run. A test environment is available for all external dependencies.	All tests can be run on developers' machines. External dependencies are either replaced with imitations or can be run locally.

Visibility and reporting				
Transparency and information sharing	Information about development and production is scattered and formed manually.	Some teams produce information about their actions automatically. Information might be centralized, but is not available for everyone and no combined analysis is done.	Fault situations of the most critical environments are shown in the workspaces of the responsible teams. Reports of the development process is automatically generated at least monthly.	The status of all systems is available for both development and productions teams and their managers on all organization levels. Development produces portfolio level information to upper management and it is easily available and almost real time.
Information sharing between development and production	Development and production is separated. Production can't access development documentation and developers can't access metrics produced in production.	Production and development is separate, but reports are shared as needed. A process exists for sharing information.	A part of tools used by production and development are common and information is shared between the team regularly.	Development team is able to follow problems occurring in production at any time. Production can forward tickets they have accumulated to development, if the problem can't be fixed without the help of the development team. Production and development can communicate fluently or the teams are combined.
Automatic monitoring of the production environment	The status of the service is not automatically monitored.	Some components of the service are monitored automatically or manual monitoring (e.g. disk space) has a well documented process.	All components of the service are monitored automatically, but total availability is not calculated or reacting to fault situations is not done automatically.	The metrics of different components and servers of the service are constantly monitored and they leave history information. Availability of the distributed infrastructure is calculated. Reacting to fault situations is done automatically.
Version control	Version control is not used or the use is sporadic.	Version control has a documented process, but the process is followed randomly or it does not serve good practises.	Version control process is followed extensively and procedures cover special situations.	Developers work with systematic version control. Problems caused by inadequate use of version control practically don't appear. Versions deployed to production can be easily linked back to version control.

Technology and architecture				
Management of shared components	Sharing of components relies on tacit knowledge.	Information about shared components is maintained in documentation.	Installation packages of shared components are usually available.	Copies of the components can be installed from a centralized place and notification of their updates are available.
Automatic monitoring of dependencies	Dependencies are not followed structurally.	Dependencies are managed with manual documentation.	Component dependencies are followed automatically, but centralized dependency information is not available or in use for those developing internal dependencies.	Component dependencies are followed automatically. Information is available for other teams as well. The information contains version and other information of the dependencies.
Software development tools	Every team is responsible for their own development tools.	Teams are responsible for their own tools, but the tool management is centralised. Backups and monitoring works in device level.	Some of the tools are centralised, but they do not offer different workflows for different teams or tools are managed manually.	All shared tools are centralised to common environment. Teams can manage automated project creation inside the team. Tools enable different workflows for each team.
Technology and architecture	Technology is old and/or does not meet the requirements of the architecture in scalability, integration and security aspects.	Technology is about to be too old and architecture is flexible only in some parts or interfaces are inadequate.	Technology is quite new and well supported. Architecture is separated and all frontend functionality can be done also using interfaces.	Technology is new or well supported. System structure is lead as a whole and is divided into appropriate microservices.

APPENDIX 2: TEST AUTOMATION MATURITY MATRIX

Test Automation Maturity				
Unit Testing	No unit tests	Unit tests are executed locally on the developer's machine.	Unit tests are executed automatically on the CI-server. Failed tests prevent from releasing. There is something to be desired in respect of test coverage.	Unit tests are fully automated and their test coverage is over 85%.
Integration Testing	No integration tests	Integration tests are executed manually.	Integration tests are executed automatically on the CI-server. Failed tests prevent from releasing. There is something to be desired in respect of test coverage.	Integration tests are fully automated and their test coverage is over 85%.
System Testing		System tests are executed manually by the tester.	System tests are executed automatically on the CI-server. Failed tests prevent from releasing. There is something to be desired in respect of test coverage.	System testing is fully automated and its test coverage is over 85%.
Performance Testing	No performance testing	Performance tests are executed manually.	Performance tests are executed automatically on the CI-server. Failed tests prevent from releasing. There is something to be desired in respect of test coverage.	Performance testing is fully automated and it simulates well user load and actions.
Security Testing	No security testing	Security tests are executed manually.	Security tests are executed automatically on the CI-server. Failed tests prevent from releasing. There is something to be desired in respect of test coverage.	Security tests are fully automated and their test coverage is on a good level.

THANK YOU!

Mikko Drocan

mikko.drocan@eficode.com

Markus Suonto

markus.suonto@eficode.com

