19.06.2017

VOLVO IT: NAMS

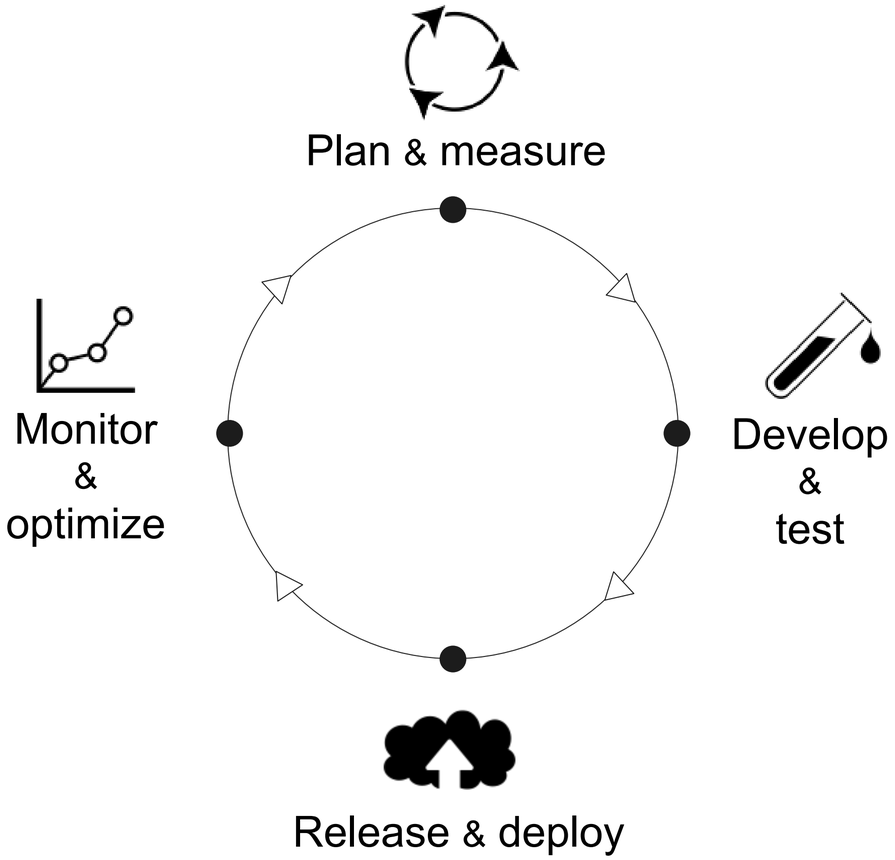
DEVOPS DEVELOPMENT REPORT

INDEX

INTRODUCTION

DEVOPS IN A NUTSHELL

DevOps enables continuous software delivery



- Faster time-to-value
- More time to innovate and improve
- Improved customer experience

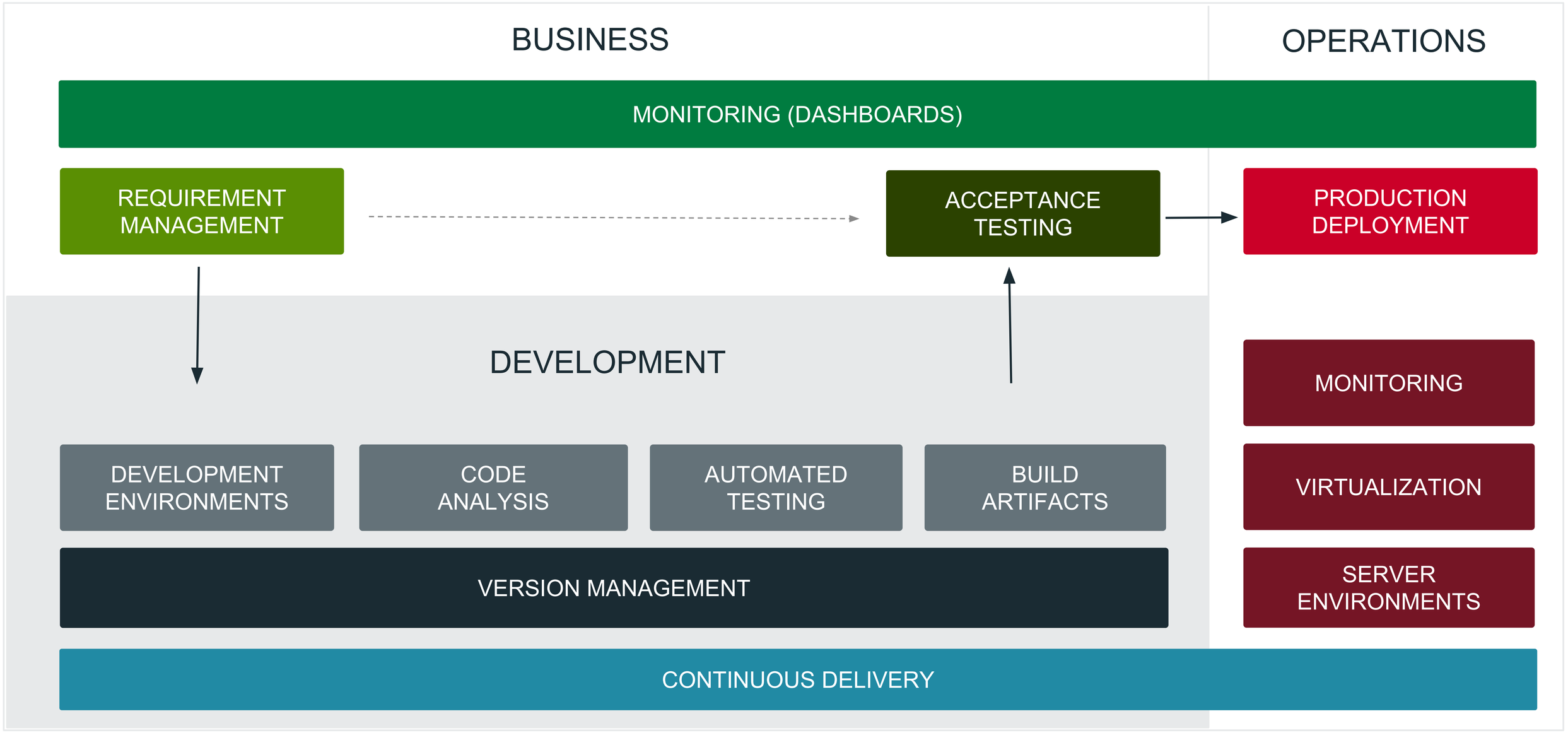
DEVOPS TRANSFORMATION JOURNEY

Efficient and highly automated SW production line from requirement management to commercial deployments and maintenance

End to end visibility to all production phases via data driven tracking, reports and dashboards

Continuous operational improvement based on both real-time and historical performance data

MODERN SOFTWARE DEVELOPMENT MODEL



SUMMARY

BACKGROUND

- Eficode performed a Devops pre-study with Volvo IT’s NAMS team
  - 7 persons with varying backgrounds and different job descriptions were interviewed.
  - In addition, background material and NAMS application architecture was reviewed.
  - The target was to assess the state of the current DevOps implementation and identify areas for improvement.
  - Focus on how to the 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 IT NAMS software production environment
  - The product has a relatively long history and a large code base that can be described as legacy.
  - The product uses a variety of client technologies.
  - There is plenty of system-specific product variation that enables integrations to different back-end systems.
  - Customization capabilities give competitive advantage but also create complexity as no two customer services are alike.
    - Recently much of the development focus has been on a individual customer specific implementation projects, which has taken bandwidth from the generic NAMS development and further slowed down the development effort.

eficode

VOLVO IT KPIs:

Reduce Lead-time
Increase Quality

DEVOPS MATURITY MODEL

FOR NAMS

36/100

WWW.EFICODE.COM

	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.	There are real-time metrics available from the development, supporting decision making and tracking the completion of strategic targets.
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.	The development organization meets regularly. Collected metrics 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.	Metrics are 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 appropriate and enable reaching business targets efficiently.

DEVOPS MATURITY MODEL

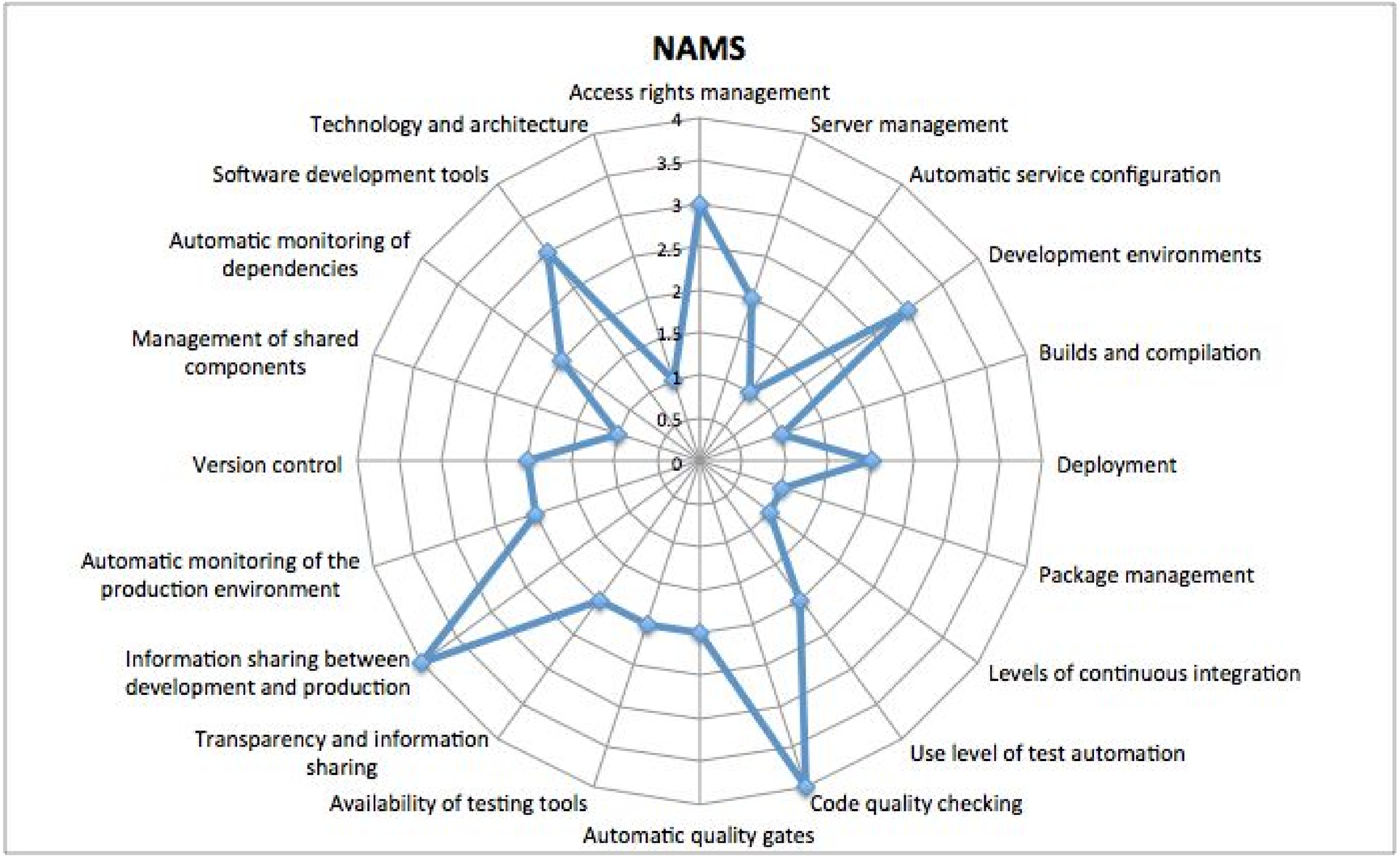
AVERAGE FOR HEAVY INDUSTRIES

32/100

WWW.EFICODE.COM

	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.	There are real-time metrics available from the development, supporting decision making and tracking the completion of strategic targets.
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.	The development organization meets regularly. Collected metrics 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.	Metrics are 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 appropriate and enable reaching business targets efficiently.

NAMS' Automation maturity: 52/100



Scale 0 → 4

KEY OBSERVATIONS

CURRENT STATE

LEADERSHIP

- Organizational structure and responsibilities are unclear
  - Key question to ask is how to put emphasis on communication between business and NAMS team?
  - There has been long-going effort to convince leadership to appoint someone from business to be responsible of NAMS.

- Mission is unclear
  - Who decides what is the purpose of NAMS?
  - Clear mission will also help in prioritizing different development project tasks more correctly from the beginning.
  - Also intra-project scheduling conflicts may decrease.

- Business perspective is missing
  - Nobody in leadership considers NAMS their primary business.
  - These results in multiple issues for the team affecting eg. task prioritization, information sharing between different services and development schedules.
  - This is one of the largest time consuming factors of key personnel which makes those persons knowledge silos.

CURRENT STATE

ORGANIZATION & CULTURE

- The team’s way of working has some attributes of known Agile process models:
  - Kanban, but without the ability to constantly release ready items.
  - Scrum, but without the ability to lock scope.

- The team has observed a few improvement ideas in respect to the current way of working.
  - Excellent spirit and proactive communication has been **most important factor for keeping the product up and running**.
  - Although the team is spread into two continents, active communication between the different parts of the team is daily activity.
    - Both locations hold knowledge from various activities and are thus to some extent self-organized and cross-functional.
  - Team understands deeply the business criticality of their domain.

CURRENT STATE

ORGANIZATION & CULTURE

- In the current organizational co-operation, the interconnected services do not know of each other’s statuses.
  - NAMS relies on the information given by the other services and thus needs to access other services without interruptions.
  - Operations maintenance isn’t the first priority for any team.
  - Frequently NAMS has been the first team to notice whether a service hasn’t been working correctly.

- Team is located within two locations, Gothenburg (UTC+01:00) and Curitiba (UTC-03:00).
  - Daily and weekly meetings are nevertheless utilized for discussions and sharing information.
  - Communication between Gothenburg and Curitiba is available mainly via different tools and thus a dedicated collaboration tool would be useful.
    - Skype and email are used for daily communication. Missing a communication tool for the whole team.
      - Itself Skype is a usable real-time communication tool, but it’s main focus is on 1-to-1 communication.
    - Face-to-face communication appears to be quite easy within the team.
      - It is fast, but as discussions are not written to TFS as tasks or other documentation, information is lost.

CURRENT STATE

ORGANIZATION & CULTURE, RELEASE

- Issue tracking and requirement management is handled with TFS.

- Release cycle is long.
  - Varying development periods.
    - Dependent on other teams of Volvo IT organization; no visibility to what those teams are doing.

  - About four releases a year.

  - Ability to release is the biggest key factor to improve current state.

CURRENT STATE

ENVIRONMENTS & RELEASE

- NAMS has fairly intense infrastructural needs for the environments.
  - Since it is business-critical application, high availability is a must.
  - NAMS needs to store audit logging resulting in need for terabytes of storage.

- Here is an example what infrastructure of NAMS would cost in public cloud. What level of service are You paying of?
  - Servers
    - Get servers immediately when you need them.
    - Infinite scaling, multiple availability zones and regions ensure high availability.
  - Databases
    - Fully managed databases with high availability and read replicas as services.
    - Automatic security patches, engine upgrades available whenever you want them
  - Long-term storage
    - Different levels of storage solutions available to optimize cost
  - Price
    - NAMS total infrastructure budget in a public cloud (AWS) is 10 - 20 k€ / year.
    - **NAMS' current budget is in order of magnitude greater.**

CURRENT STATE

BUILDS & CONTINUOUS INTEGRATION

- To be able to speed up releasing, You should aim to:
  - Create a build per new commit
  - Create multiple builds per day
  - Automated tests are executed for each build
  - A successfully tested build is deployed automatically to the test environment.
  - Release to QA and Production environments should be one-click-operation from CI.

- Situation today:
  - TFS is able to build NAMS package and partly deploy it to different environments.
  - There are no automatic build triggers in use.
  - Environments are unique and different in multiple ways.
  - Production deployment has a long manual checklist where as it should be fully automatic procedure.
  - Scheduled time frame required for deployments.
  - Test automation is part of the development process
    - It is not clear what is the overall state of the test automation within various development and test phases.
    - Usage of test automation as a driver for development is low.

CURRENT STATE

BUILD & CONTINUOUS INTEGRATION: RECOMMENDATIONS

- Try to agree with HCL that You can deploy to production environment whenever You want, unless HCL requests for a maintenance window.
  - Initially, there does not seem to be any reason why the way of cooperation could not be achieved.
    - NAMS is already working outside of the normal co-operation boundaries, as agreed.

- Automate environment and application configuration, and store them in version control.
  - Minimizing the amount of manual work done for configuration tasks is vital. Doing so You can decrease the probability of having environments misconfigured and not working.

- Configure environments automatically at the beginning of each deployment as it’s defined in version control.
  - Doing so will decrease the probability of having different versions in use without Your knowledge. Additionally, this practice drives You to always update the latest working configuration into version control.

CURRENT STATE

BUILD & CONTINUOUS INTEGRATION: RECOMMENDATIONS CONTINUE

- Configure the application automatically at each deployment as defined in version control.
  - Having an automation to configure the environment after each deployment is also needed to increase the level of test automation, as system needs to be easily configurable from tests.

- Make sure all team members can see Your work in the version control.
  - Visibility throughout the development process is very helpful on many cases as “given enough eyeballs, all bugs are shallow” (Linus’s Law)

- Ensure manual deployment checklists are obsoleted by previous steps.
  - In most cases there’s a way to handle the manual steps automatically. You may need to work on the solution for some time, but while working on it you may notice that the manual checklist is, for example, incomplete, has duplicated issues, or has mistakes.

CURRENT STATE

QUALITY ASSURANCE

- Currently:
  - Testing is done mostly after development activities
    - Take the advantage of executing automated tests after each build. Target for fixing problems right after a build fails.
  - Testing activities take days or even weeks to be done
    - Product’s overall status is unknown for a long time.
- Visibility of testing is questionable
  - Acceptance criteria, test cases and test reports should be visible and understandable for everyone.
  - Test cases should be stored in version control. Additionally, if a test case needs, for example, some pre-configuration on the environment side or additional checks, the prerequisites should be defined in the test automation.

CURRENT STATE

QUALITY ASSURANCE

- Testing should be taken into account as early as possible, meaning already at requirement definition stage.
  - Testers should usually be (after business) the persons who understand the requirement the best.

- There should be a documentation that one can easily refer to while testing and creating test cases.
  - Documentation can be accumulated by writing it when discussion is prompted.

- Possible major risks because of human errors.
  - Wrong environments used in testing.
  - Used test tool is incorrectly configured or old version.
    - Increasing the level of automation in all used testing would result in giving fast feedback that drive further development. Eg. unwanted environment changes can be seen faster and be corrected.

- Testers main responsibility should be to update automated test sets, not to “test that stuff still works”.
  - There seems to be no areas that are “impossible to automate” in NAMS.
  - Automated tests suffer from false results caused by environment or application configuration.

CURRENT STATE

QUALITY ASSURANCE, VISIBILITY & REPORTING

- Aim to 100% test automation level
  - First create a roadmap for increasing the level of test automation.
  - Execute tests automatically after each deployment in the CI server. They should be an automated quality gate in the CI pipeline.
  - Fully automate all current cases and then start increasing coverage.
    - Start with simple, but necessary tests (“smoke set”) and continue until you have a good feeling about the coverage of automated tests executed after a deployment. The next step is not to stop test development activities, instead continue to improve the tests in sync with the product’s development activities.

- Currently, test reporting are created manually and distributed via email.
  - Syncing different reports and analyzing trends is difficult.
    - Each test execution must generate a report that accurately reflects the current state of the NAMS version used in testing activities.
    - Usually test automation tools provide a good and easy way to produce test reports that are similar and easily comparable.
    - If Your current test framework can’t generate reports, try some modern test framework, for example Robot Framework.

CURRENT STATE

TECHNOLOGIES & ARCHITECTURE

- The architecture is legacy at this point.
  - This is no surprise for a project that has been on-going as long as NAMS.

- Refactoring and architectural renovation need to be taken into consideration, but this is recognized in the team.
  - Effective roadmap in place:
    - Plans about branching model in version control.
    - Architecture refactoring to microservice-like.
    - Increasing the level of test automation.

- Improvement suggestions give our vision of the priority order.

WIDER ORGANIZATIONAL ISSUES

- The following issues need larger discussion inside Volvo IT organization and are as such something NAMS team cannot improve by themselves.
  - We propose NAMS team to be the spearheads in these discussions.
  - Tools for development.
    - Right now, it is difficult to employ new tools because of heavy processes.
    - This is hampering the daily work greatly.
  - Organizational responsibilities are hard to understand.
    - What is the larger mission?
  - Visibility between projects.
  - Communication with HCL.
    - Slow response time
    - Inflexible processes

IMPROVEMENT SUGGESTIONS & HIGH-LEVEL ROADMAP

Development proposal

#	Title	Difficulty to implement	Description
1	Automate configuration	L	Define environment and application configuration in version control. Implement a way to automatically apply configuration especially for test automation. Promote it to become the only way to alter configuration.
2	Define completely new concept for “a release”	S	Employ Blue-Green deployment model and feature toggles (part of application configuration) to separate the concepts of pushing new versions to production and when those are turned live, freeing you from intra-organizational scheduling constraints. Work with HCL to forego the release window.
3	Test automation: tool evaluation and incorporation	S	Evaluate used test tooling with open source alternatives, especially concentrating on interface testing.
4	Collaboration tool: evaluation and incorporation	S	Nowadays there are multiple communication tools (eg. Slack, Flowdock) that are used for secure team communication. A team communication tool would raise quick communication and improve visibility.
5	Test automation: smoke set	M	First create a roadmap for increasing the level of test automation. Fully automate all current cases (“smoke set”) and then start increasing coverage. Execute tests automatically after each deployment in the CI server.

Development proposal

#	Title	Difficulty to implement	Description
6	Continuous Integration	S	Apply configuration (proposal #1) automatically upon every build.
7	Continuous deployment	M	Deploy automatically to all environments with a click of a button. Deploy automatically to Test-environment upon every build. Use test automation as an automated quality gate in the CD pipeline.
8	Employ Kanban fully	M	After #2, you should be able to work without intra-organizational dependencies affecting you that much. Start following Kanban metrics and work to improve your process based on those.
9	Evaluate possibilities of moving to public cloud	S	Evaluate, if you could employ at least partly, public cloud services especially for storage. This step should feed into proposal #10.
10	Change application architecture to microservice-like	L	Also devise a deprecation strategy, so you can remove dead code.

HIGH LEVEL ROADMAP

Phase 1

- Automate configuration
- Define completely new concept for “a release”
- Test automation: tool evaluation and incorporation
- Collaboration tool: evaluation and incorporation
- Test automation: smoke set
- Continuous Integration

Phase 2

- Continuous deployment
- Employ Kanban fully
- Evaluate possibilities of moving to public cloud

Phase 3

- Change application architecture to employ micro-services.

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	*Version control and installation is not connected without manual steps.*	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	Installation is done manually and few people are able to do them.	*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	*No package management or differing conventions.*	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	*No continuous integration or loose conventions.*	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.

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.

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.

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	No system tests.	*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

Tatu Kairi
tatu.kairi@eficode.com

Markus Suonto
markus.suonto@eficode.com