Stona Automotive

AutoPLM System Documentation

Release 1.0.0

Sergei Antipov

Vu Minh Duc

Aleksandr Nekrashevich

Sarim Jalal

Matheus Cornejo

Andrey Tutchev

Moscow, Feb 28, 2024

Table of Contents

1. Introduction

1.1. Goal

With the creation of AutoPLM, a state-of-the-art software system tailored to the changing needs of the automotive industry, Stona Automobile wants to revolutionize the way automotive companies handle the challenges of product development and management by digitizing operations through AutoPLM. With the launch of AutoPLM, the automotive industry will take a big step toward increased efficiency and digitization, and the standards for automotive excellence will likely be redefined.

Furthermore, from conceptual design to end-of-life management, every stage of the product life cycle is optimized by the AutoPLM system's engineering. AutoPLM aims to decrease time-to-market, optimize workflows, and guarantee that products meet the highest standards of quality and innovation by increasing the digitalization of the automotive industry. This all-encompassing method of managing the product life cycle is evidence of Stona Automobile's dedication to adopting digital transformation and raising the bar for automotive excellence.

1.2. Objectives

- Implement a system that improves data management, guaranteeing the integrity and accessibility of information throughout the organization.
- Develop a system that optimizes each phase of the product life cycle. making automotive company more digitized
- Enhance collaboration across departments by facilitating seamless communication and data sharing, thereby breaking down silos within the organization.
- Reduce time-to-market for new vehicle models by streamlining design, manufacturing, and testing processes through integrated workflows and real-time analytics.
- Increase operational efficiency by automating routine tasks and employing predictive analytics to anticipate and mitigate potential issues in the production line.

2. Description of the Model

The AutoPLM system, developed by Stona Automobile, is a sophisticated information system designed to encompass several critical aspects that are essential for every car producer in the automotive industry. This comprehensive system is built to streamline operations, enhance efficiency, and foster innovation across the entire organization. The model includes the following key components:

- **Supply Chain Management**

AutoPLM integrates advanced supply chain management features, designed to optimize the flow of materials, information, and finances as they move from supplier to manufacturer to wholesaler to retailer to consumer. This component facilitates real-time tracking of parts and materials, enabling proactive management of supply chain risks and ensuring timely delivery of components, which is crucial for maintaining production schedules and reducing costs.

- **Production Planning for Current and New Models**

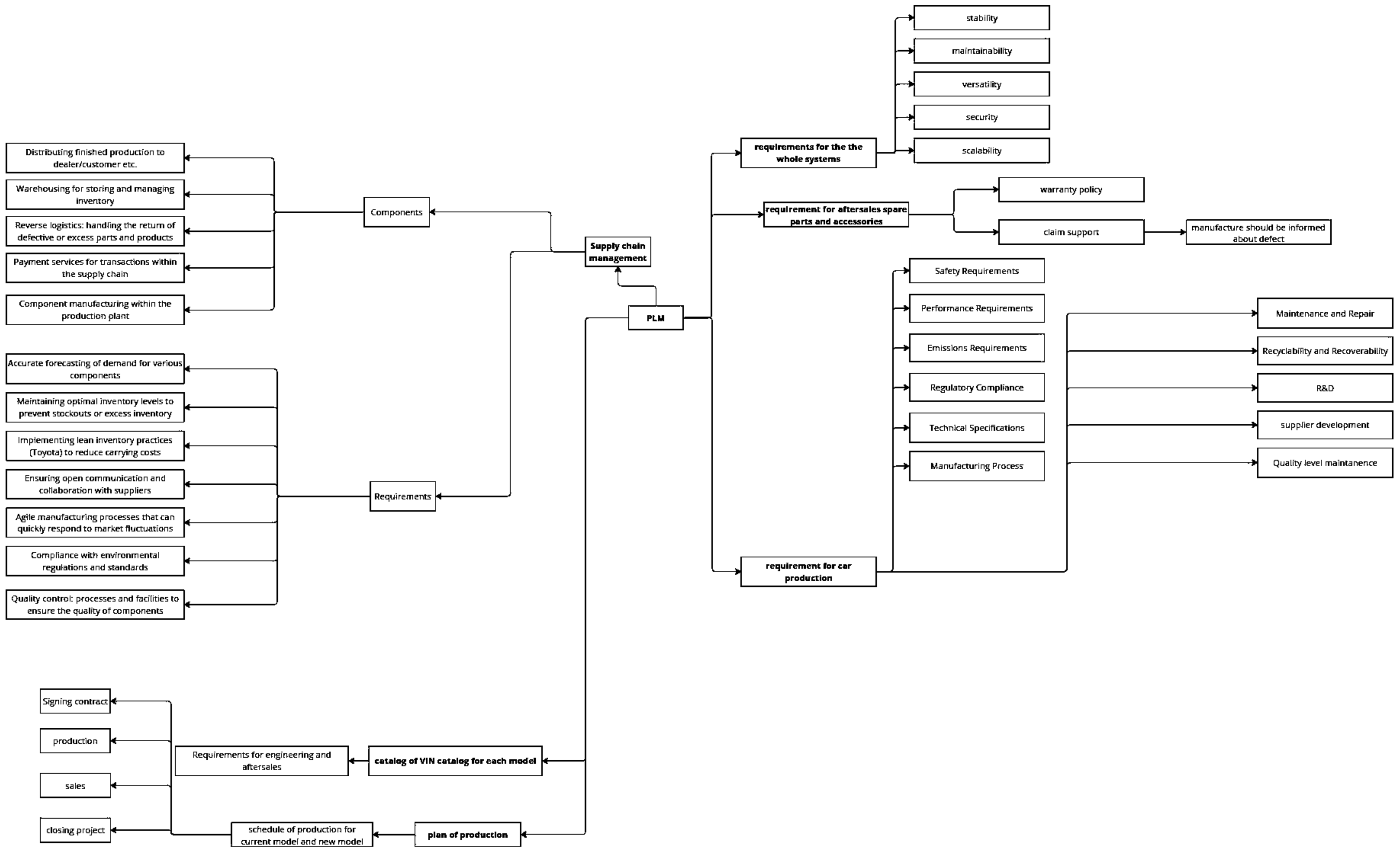
The system provides robust tools for production planning, accommodating both current and future vehicle models. This includes scheduling production runs, managing inventory levels, and allocating resources efficiently to meet production targets. By optimizing production planning, AutoPLM helps in reducing lead times and improving the flexibility to adapt to market demands or changes in production requirements.

- **Technical Requirements for the Information System**

AutoPLM is engineered to meet high technical standards, ensuring reliability, scalability, and security. The system's architecture is designed to support large volumes of data and complex operations, while providing a user-friendly interface for employees across the organization.

- **Requirements for After-Sales Maintenance**

Recognizing the importance of after-sales service, AutoPLM includes features to support maintenance and customer service operations. This involves tracking warranty claims, scheduling service appointments, managing spare parts inventory, and providing support for service technicians. By enhancing after-sales maintenance, the system aims to improve customer satisfaction and loyalty.


PLM
Supply chain management
Components
Distributing finished production to dealer/customer etc.
Warehousing for storing and managing inventory
Reverse logistics: handling the return of defective or excess parts and products
Payment services for transactions within the supply chain
Component manufacturing within the production plant
Requirements
Accurate forecasting of demand for various components
Maintaining optimal inventory levels to prevent stockouts or excess inventory
Implementing lean inventory practices (Toyota) to reduce carrying costs
Ensuring open communication and collaboration with suppliers
Agile manufacturing processes that can quickly respond to market fluctuations
Compliance with environmental regulations and standards
Quality control: processes and facilities to ensure the quality of components
catalog of VIN catalog for each model
Requirements for engineering and aftersales
plan of production
schedule of production for current model and new model
Signing contract
production
sales
closing project
requirements for the the whole systems
stability
maintainability
versatility
security
scalability
requirement for aftersales spare parts and accessories
warranty policy
claim support
manufacture should be informed about defect
requirement for car production
Safety Requirements
Performance Requirements
Emissions Requirements
Regulatory Compliance
Technical Specifications
Manufacturing Process
Maintenance and Repair
Recyclability and Recoverability
R&D
supplier development
Quality level maintanence

3. Approach: Spiral Model

3.1. Comparison

	Time	Quality	Cost	Documentation	Extensibility	Score
Code-and-fix model	**6**	**3**	**2**	**0**	**2**	**2.6**
Waterfall model	**2**	**4**	**3**	**3**	**3**	**3**
Rapid prototyping model	**6**	**3**	**3**	**2**	**4**	**3.6**
Incremental model	**8**	**6**	**5**	**6**	**2**	**5.4**
Synch-stabilization model	**2**	**5**	**3**	**2**	**3**	**3**
Spiral model	**3**	**9**	**7**	**9**	**8**	**7.2**
OO model	**4**	**6**	**5**	**6**	**9**	**6**

3.2. Benefits of Spiral Model

The implementation approach for the AutoPLM system adopts the Spiral Model of software development. This model is particularly suited to complex projects that entail various aspects such as supply chain management, production, distribution, and after-sales service, allowing for iterative development, risk management, and incorporating feedback at each stage.

- **Iterative Development**

The Spiral Model facilitates iterative development, enabling the project team to refine and expand the AutoPLM system through multiple cycles. This approach allows for the gradual integration of each critical aspect of the company's operations, starting from the initial concept to the final deployment. By breaking down the project into smaller segments, the team can focus on high-priority features first, ensuring that the most impactful areas of the business benefit from early improvements.

- **Risk Management**

A key advantage of using the Spiral Model is its inherent focus on risk management. At each iteration, potential risks are identified, assessed, and mitigated. This is particularly important for a large company like Stona Automobile, where changes in one area, such as the supply chain, can have significant repercussions on production, distribution, and after-sales service. The model allows the company to address these risks proactively, ensuring that the system remains resilient and adaptable.

- **Incorporating Feedback**

The iterative nature of the Spiral Model also provides ample opportunity to incorporate feedback from various stakeholders, including management, employees, suppliers, and customers. This feedback is critical for ensuring that the AutoPLM system meets the actual needs of the organization and its ecosystem. By engaging stakeholders at each stage of development, Stona Automobile can fine-tune the system to better support its operations and strategic objectives.

4. Methodologies

4.1. Comparison

The development and implementation of the AutoPLM system at Stona Automobile incorporate several software development methodologies, each chosen for its unique strengths and suitability to different aspects of the project. These methodologies are ranked by their relevance to the project as follows:

IBM Rational Unified Process (RUP)

Parameter	Score (1 to 5)	Justification
Flexibility and Adaptability	**3**	RUP is somewhat flexible but can be perceived as more process-heavy and structured, which might limit adaptability in certain contexts.
Documentation Management	**5**	RUP provides detailed documentation, which can be beneficial for formal projects with high documentation requirements.
Handling Regulatory Requirements	**5**	RUP is well-suited for projects with stringent regulatory requirements due to its structured and controlled approach, which facilitates compliance.
Team Collaboration	**4**	RUP emphasizes collaboration and communication through phases, fostering good team interaction.

Microsoft Solution Framework (MSF)

Parameter	Score (1 to 5)	Justification
Flexibility and Adaptability	**2**	MSF tends to be more prescriptive and may not be as flexible as Agile approaches.
Documentation Management	**3**	MSF also promotes documentation, but it may not be as extensive as RUP, striking a balance between documentation and agility.
Handling Regulatory Requirements	**4**	MSF also emphasizes structure and control, making it suitable for handling regulatory requirements, although it might be perceived as slightly less accommodating than RUP.

Team Collaboration	**3**	MSF also promotes collaboration but might be perceived as less collaborative than Agile due to its structured nature.

Agile Methodologies (Scrum, XP)

Parameter	Score (1 to 5)	Justification
Flexibility and Adaptability	**5**	Agile is known for its high adaptability and flexibility due to its iterative and incremental nature.
Documentation Management	**2**	Agile, while valuing working software over comprehensive documentation, encourages just-enough documentation.
Handling Regulatory Requirements	**3**	Agile is adaptable but may face challenges in highly regulated industries where extensive upfront planning and documentation are required.
Team Collaboration	**5**	Agile places a strong emphasis on continuous collaboration, communication, and cross-functional teams, making it highly effective for team collaboration.

Overall rating

Methodology	Pros	Cons	Grade
IBM Rational Unified Process (RUP)	- Detailed documentation - Quality control	- Complexity and heaviness - Costs associated with change management	**4.25**
Microsoft Solution Framework (MSF)	- Integration with Microsoft tools	- May be less suitable for large projects - May not offer the same degree of flexibility as Agile	**3**
Agile Methodologies (Scrum, XP)	- Flexibility and adaptability	- May not be suitable for formal projects - Requires high team discipline and involvement	**3.75**

4.2. Benefits of RUP

1. RUP emphasizes detailed documentation throughout the development process, which can be advantageous in projects with strict regulatory requirements, like AutoPLM project.

2. RUP includes a strong focus on risk management and project control, helping teams identify and mitigate potential issues early in the development process, as well as monitoring and managing project progress effectively.

3. RUP, like Agile, is highly customizable, and also supports an iterative development approach, allowing for incremental progress and the ability to adapt to changing requirements.

4. Overall, RUP, while also being not tied to Microsoft technologies, is well-suited for any large-scale enterprise projects with complex requirements and a need for a structured development approach.

5. Constraints

Business:

- Complex and global supply chain networks may introduce challenges in tracking changes and workflow across suppliers and partners
- Resistance to change within the organization may impede the adoption of new software interfaces and workflows
- Adherence to specific (while also constantly changing) industry regulations and standards

Technical:

- Existing legacy systems may lack standardized interfaces, making integration more challenging.
- Stringent data security and compliance requirements may restrict the types of data that can be exchanged with external systems.
- The sheer volume and complexity of data generated during the lifecycle and workflow.

6. Quality Attributes

6.1. Key Attributes

- **Extensibility**

Extensibility is a critical quality attribute for the AutoPLM system, ensuring that the software platform can evolve to accommodate new features or capabilities as the automotive industry advances. This attribute allows Stona Automobile to adapt to future technological innovations or market demands, enhancing the system's longevity and relevance.

- **Maintainability**

Maintainability is essential for the ongoing success and operational efficiency of the AutoPLM system. It enables timely and cost-effective modifications for error corrections and minor functional adjustments, without necessitating major overhauls. This attribute ensures that the system can be kept up-to-date with minimal disruption, supporting continuous improvement.

- **Intellectual Property Protection**

Intellectual Property Protection (IPP) is paramount in safeguarding the proprietary technologies and processes embedded within the AutoPLM system. Through effective licensing management systems, Stona Automobile can protect its intellectual property, ensuring that its innovations remain secure from unauthorized use or replication. This protection is vital for maintaining competitive advantage and fostering trust with partners and customers.

6.2. DevOps

The implementation of DevOps practices is integral to ensuring that the AutoPLM system embodies its core quality attributes and remains reliable under any circumstances. Through the adoption of several key instruments, the system's development, deployment, and maintenance processes are optimized for efficiency, security, and scalability.

- **CI/CD Pipelines with Auto-Testing Stages**

Continuous Integration/Continuous Deployment (CI/CD) pipelines equipped with auto-testing stages are fundamental for achieving rapid iteration cycles. This approach enables flexible maintainability and swift extensibility by allowing for automatic testing and deployment of code changes. It ensures that the AutoPLM system can evolve quickly to meet new requirements or incorporate enhancements without compromising on quality or performance.

- **Version Control and Monitoring Software**

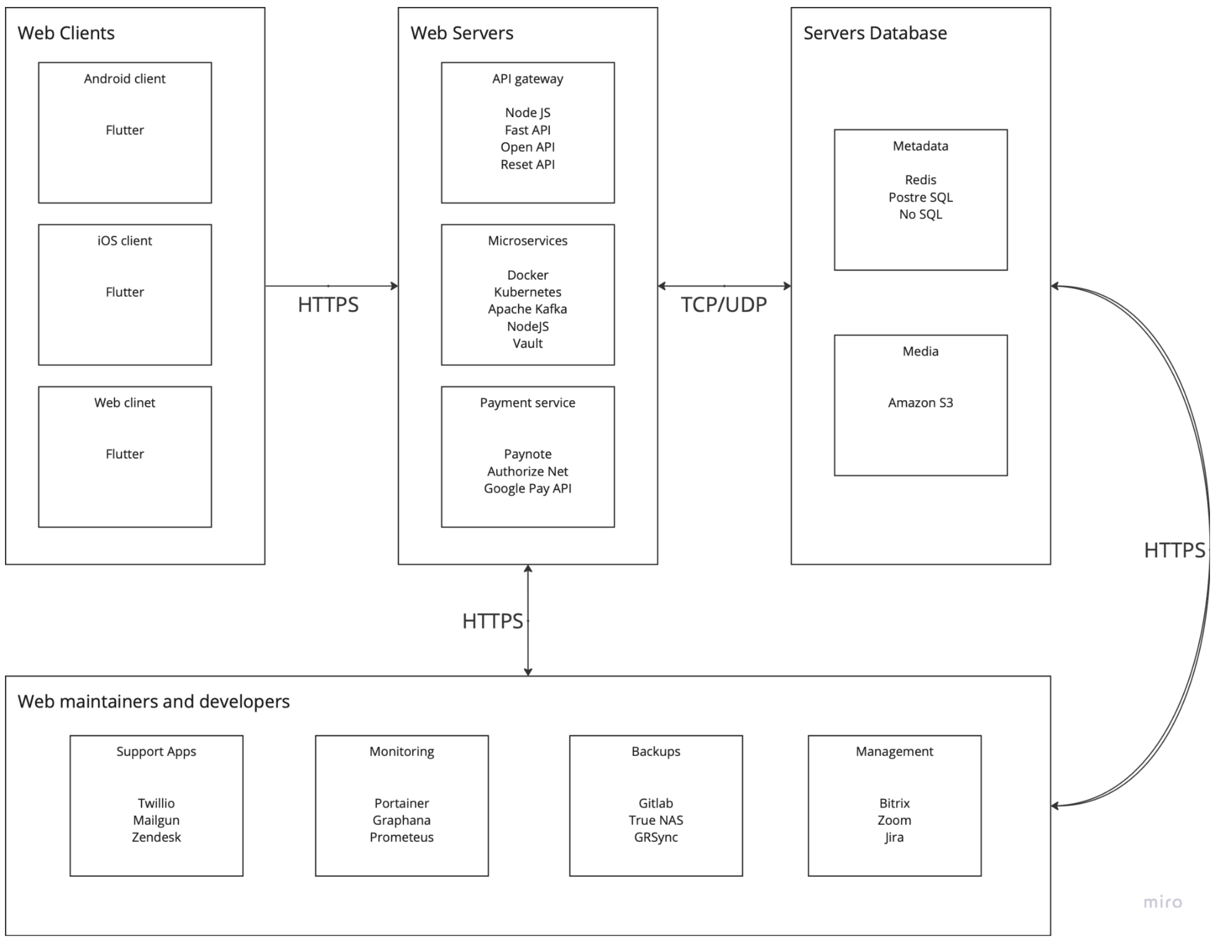
Version control systems are critical for maintaining a comprehensive record of all changes made to the AutoPLM codebase, enhancing the traceability of modifications and facilitating easier identification of issues at early stages. Monitoring software complements this by continuously scanning the system's operations to detect anomalies or performance deviations, enabling prompt corrective actions and ensuring system stability.

- **Security Scanning (OWASP) and Reliable Artifact Repository**

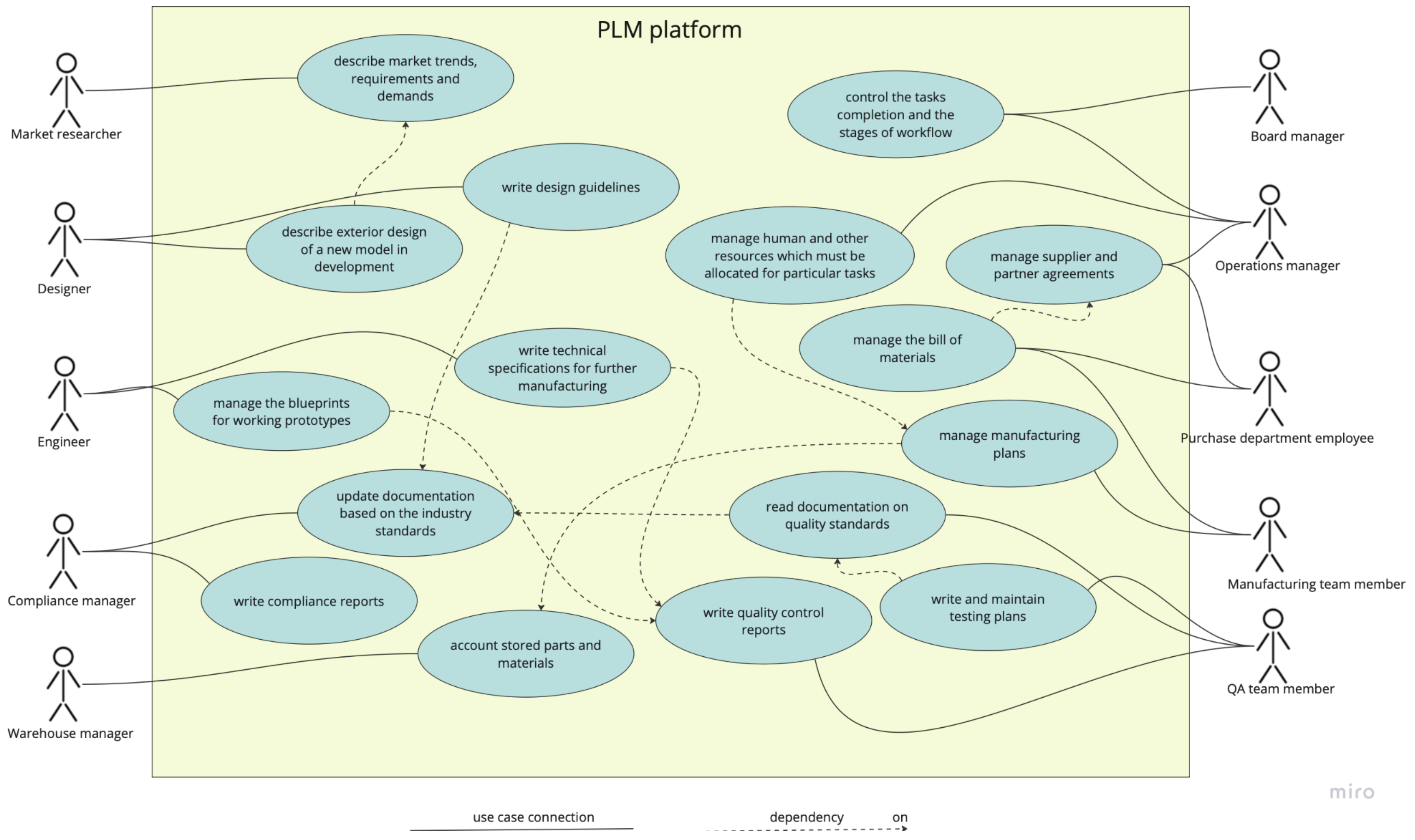
Security is paramount in the development and operation of the AutoPLM system. Implementing security scanning tools based on the Open Web Application Security Project (OWASP) guidelines helps in identifying and mitigating vulnerabilities, protecting the system against potential threats. Additionally, utilizing a reliable artifact repository for managing software artifacts ensures that all components of the system are securely stored and managed, safeguarding sensitive data, credentials, and intellectual property from unauthorized access.

7. Diagrams

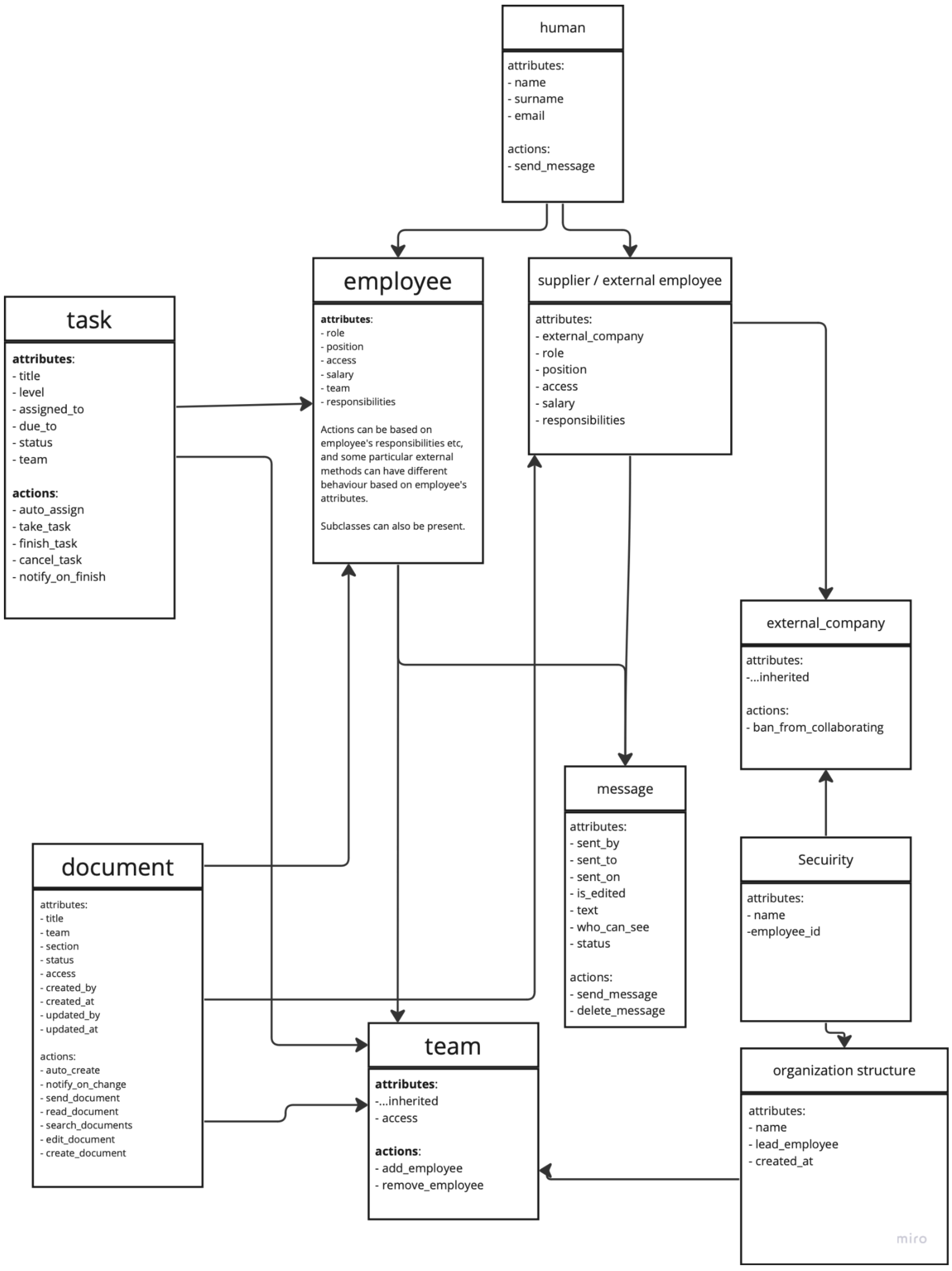
7.1. Architecture Diagram



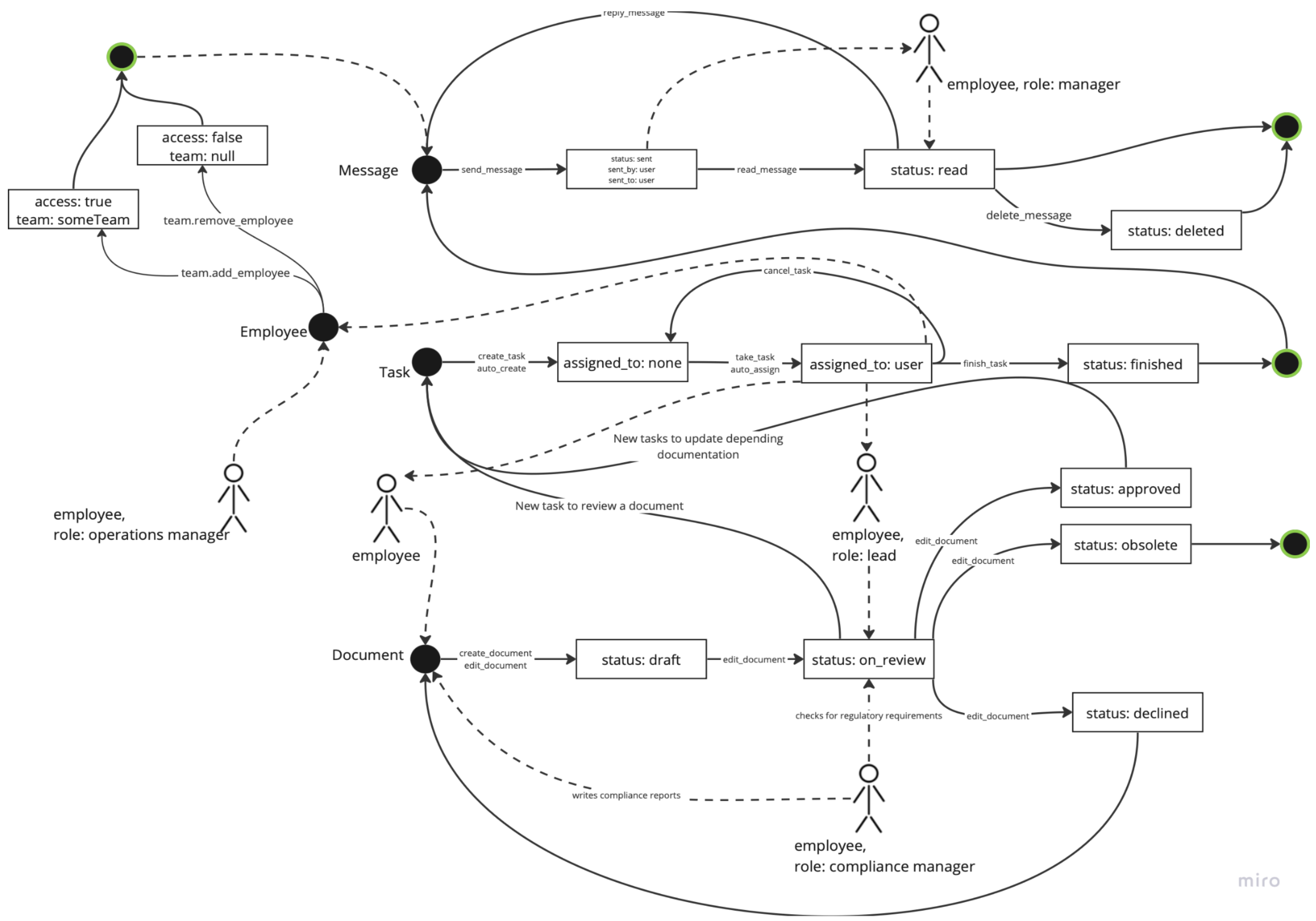
7.2. Use Case Diagram


PLM platform
Market researcher
Designer
Engineer
Compliance manager
Warehouse manager
Board manager
Operations manager
Purchase department employee
Manufacturing team member
QA team member
describe market trends, requirements and demands
write design guidelines
describe exterior design of a new model in development
control the tasks completion and the stages of workflow
manage human and other resources which must be allocated for particular tasks
manage supplier and partner agreements
manage the bill of materials
write technical specifications for further manufacturing
manage the blueprints for working prototypes
manage manufacturing plans
update documentation based on the industry standards
read documentation on quality standards
write compliance reports
write and maintain testing plans
write quality control reports
account stored parts and materials
miro
use case connection
dependency on

7.3. Class Diagram



7.4. State Transition Diagram



8. Plans for AutoPLM Implementation

8.1. Implementation

Goal: develop and implement a comprehensive Product Lifecycle Management (PLM) software solution that meets the specific needs of the automotive industry.

Objectives:

1. Build a scalable architecture
2. Incorporate security measures
3. Develop user-friendly UI that satisfies end-users' needs across different departments
4. Focus on task-oriented functionalities to maintain the production workflow
5. Implement features that ensure compliance with industry standards and regulations
6. Foster collaboration with current vendors and partners for seamless integration

Scope: end-to-end development, deployment and ongoing maintenance of a PLM software solution

Human resources (overall 40+ people at least):

1. Development teams: backend/frontend, software architects, DB administrators (15-20 total)
2. QA team: performance testers, e2e testers etc (7-10 total)
3. Management team: project managers, product owners, business analysts (5-7 total)
4. External consultants: for example, industry partners

Other resources (by groups):

1. Hardware infrastructure (servers etc)
2. Development environments (CI/CD platform etc)

Approach: iterative modular development cycles for a platform system, with monthly releases of new functionalities, improvements and integrations, which allows to quickly adapt to continuously changing industry requirements and also maintain sustainability and security of the software with CI/CD methodologies

Milestones:

1. Building system architecture (*1 month*), meaning that the technology stack selection is validated beforehand, while also used for further planning.
2. Creating a working prototype (*2 months*), which facilitates improvement based on initial feedback.
3. Initial development (*9 months*).

4. Commercial release and user training (*2 months*)
5. Continuous optimization and improvements of the software (*ongoing*)

At the same time, several **activities** need to be done simultaneously during the whole process, such as testing, documentation and maintenance.

8.2. Testing

Goal: Implementing methodologies of testing the software using proposed tools to ensure the correctness of individual software components by isolating them and validating their behavior against expected outcomes.

Objectives:

1. Ensure the PLM software meets the specified requirements and works as intended.
2. Early detecting and fixing errors in the whole development cycle - Black box technologies
3. Guarantee the proper reliability, performance, and security for all PLM systems.
4. Comply with automotive industry regulations and standards
5. Additional software audit for quality validation of the whole system

Scope: full end-to-end testing from beginning to maintenance the whole life-cycle and inheritance approach for the new versions of software

Testing Approach:

1. Test planning: defining the components and functions which has to be tested
2. Functional and Integration testing: to check that all functions and system works correctly with the whole enterprise system
3. Performance testing: checking system under various loads using DDoS methods.
4. Security testing: checking for vulnerabilities and security.
5. Compliance testing: verify that system meets the requirements with automotive industry regulations and standards
6. User Acceptance Testing (UAT): checking the system for proper work by employees

General principles:

1. Each business function has to be tested using types of testing by each individual modules
2. Programmers have to make general testing of the whole system
3. QA team - has to check aspects of each business function separately: VIN catalog, BOM, PLM, SCM

Chosen testing technologies:

1. “Black box technology” is appropriate for the system, due to it covers all the selection and give maximum profitability

2. “Failure statistics, reliability analysis” allows to collect statistical information in case of data analysis for all errors which appear in the system. Based on this data, programmers and QA can determine next steps- continue testing or recode the module.
3. “Cleanroom” can be additional testing due to it combining numbers of software technologies which can be applied somewhere in the system.
4. “Technologies without code execution” does not fit due to it less formal and more simpler, and does not correspond with compliance and regulation in automotive industry

8.3. Documentation

Goal: Establish a robust documentation framework to support the comprehensive PLM solution.

Objectives:

1. Develop a comprehensive set of documentation covering system architecture, design, and functionalities.
2. Ensure documentation aligns with automotive industry standards and regulatory requirements.
3. Facilitate knowledge transfer among development teams, QA, and end-users.
4. Create documentation that supports future system maintenance and updates.
5. Implement version control for all documentation to track changes and updates.

Scope: Documentation encompasses the entire development lifecycle, from initial planning and architecture to ongoing maintenance. It includes both technical documentation for developers and end-user documentation for training and support.

Human Resources:

1. Documentation team: technical writers, content editors, illustrators (3-5 total).
2. Collaboration with Development and QA teams for accurate and timely information.
3. Subject Matter Experts (SMEs) from different departments for specialized content.

Other Resources, by usecases (prioritized):

1. Documentation management infrastructure (Confluence) for collaboration and version control of user- information, as well as comprehensive description of all the modules in the platform + guidelines for every team. (**No. 1 priority**: knowledge bank for everyone)
2. Inline documentation: (JSDoc / JavaDoc / Doxygen) for structurised description of code methods, variables and contracts (**No. 2 priority**: code transfer, clear understanding, seamless maintenance and troubleshooting)
3. Testing CASE: (Allure TestOps) for containing crucial test cases and results of testing overall (**No. 3 priority**: overall quality assurance).
4. API descriptors: (OpenAPI) for providing and maintaining clear and seamless integration.
5. Design software: (Lucidchart, Miro, Figma) graphics software for creating visual aids in documentation

6. Training environments: simulated environments for creating user guides and training materials.

Approaches (both are essential during the whole lifecycle:

1. Technical documentation: crucial for maintenance and sustainability, from low-level code comments to comprehensive descriptions of all the modules and classes separately.
2. User-centric documentation: prioritize user guides, FAQs, and troubleshooting documentation for end-users, who pay attention to business logic as the whole result of module integration.

Milestones:

1. Initial documentation framework *(1 month)*: establish documentation tools, templates, and guidelines.
2. Architecture and design documentation *(2 months)*: capture system architecture, design decisions, and data models.
3. User manuals and training materials *(4 months):* develop end-user documentation, training manuals, and videos.
4. Integration and API documentation *(2 months):* document integration points and APIs for external collaboration.
5. Ongoing updates and review *(ongoing):* regularly update documentation with each development cycle.

Other activities:

1. Review and feedback: engage stakeholders for documentation feedback and updates.
2. Continuous improvement: regularly review and enhance documentation guidelines.
3. Version control: for keeping track of changes.
4. Training material development: create materials for user training sessions.

8.4. Integration (development + testing)

Goal: ensure the correct and seamless interaction between modules within the platform, as well as between our system and existing solutions, maintaining data consistency and overall stability.

Objectives:

1. Achieve seamless connection with existing key business systems of our clients.
2. Ensure interoperability with supplier and partner systems for collaboration.
3. Incorporate technologies that facilitate communication and data transfer between platforms.

Scope: development of integration capabilities and ongoing maintenance to ensure connectivity throughout the product life cycle.

Human Resources:

1. Development Teams: engineers specialized in integration, software architects (10-15 in total).
2. Project Management Team: integration leaders, team coordinators (3-5 total).
3. QA Team: integration testers (4-5 total).
4. External Consultants: experts in systems integration, if necessary.

Other resources:

1. Hardware Infrastructure: integration servers for testing interconnected systems.
2. Integration Software: message brokers (Kafka), REST API compatible frameworks (Express / Springboot), loadbalancers (Nginx) etc.
3. CASE that allow simulating real environments for exhaustive testing: Postman, Selenium, Citrus.

Development approach: develop custom adapters and connectors to facilitate interaction between PLM software and other systems with the help of API and asynchronous instruments, such as message brokers.

Testing approaches:

1. Descending (top-down) integration testing: focusing only on logical layer (*e2e use-cases of user interaction with the production lifecycle management*) can overcomplicate the testing and make the LC iterations slower
2. Ascending (down-top) integration testing (**better, but still low regard to business logic**):
   - Low-level core operating classes (*document management / task management*) are meant to be very reusable as the platform expands, while being maintainable.

- Less time required for troubleshooting, which provides faster iterations of a LC
- Better for teams collaboration, as only changed and/or available modules are tested.

3. Combined integration testing (**chosen**):
   - While core modules are essential for the platform, they are connected to each other in a way that the number of their business applications continuously increase over time.
   - Being enriched with decisive class properties and decorators, core classes usually don't need to be accompanied with new entities.
   - Count of pairs and groups of modules will usually stay the same.
   - However, for both levels the constant increase of testing cases is expected, because all of them are responsible for countless business scenarios and functionalities.

Milestones:

1. Connector design and development (3 months).
2. Integration with existing systems (4 months).
3. Integration tests and adjustments (3 months).
4. Pilot implementation in production Environment (2 months).
5. Continuous evaluation and optimization (in progress).

Other activities:

1. Analysis of integration requirements
2. Identification of connection points
3. Development of custom APIs
4. Deep integration with existing systems, considering security and authentication protocols.
5. Constant monitoring of performance metrics
6. Staff training on new integrated processes

8.5. Maintenance

Goal: Accelerate automotive product development and management by integrating a unified PLM software platform to streamline processes from design to manufacturing.

Objectives*:*

- Maintain system stability and security.
- Regularly update and monitor the system.
- Address issues swiftly and continuously refine the system.

Scope: The plan encompasses system audits, performance monitoring, user support, and vendor collaboration, covering all stages from software updates to data recovery.

Human Resources: (overall 15+**)**

- System analysts (3-5 total)
- IT support staff (10-15 total)
- Training coordinators (2-4 total)
- Vendor liaison officer (1-2 total)

Other Resources:

- Hardware: servers for data backup, user workstations.
- Software: real-time monitoring tools, helpdesk software, online training platforms.

Maintenance Approaches, by usecases*:*

- Corrective: quick response to immediate problems, **relevant right after first release**, while also crucial for supporting the platform sustainability at every phase (No. 1 priority).
- Adaptive: updates to meet changing needs, enhancements based on user feedback, are the core of the **whole LC and product development**. (No. 2 priority, as improving business metrics and stakeholders' expectancies are relied mostly on that)
- Perfective and proactive: regular system checks to prevent issues, suitable for the **later iterations** of LC, when the main functionality is already released at the time (No. 3 priority, backlog) .

Activities*:*

- Perform system audits for security and compliance.
- Manage updates according to schedules.
- Monitor performance and user experience.
- Implement robust data backup and recovery plans.
- Perform regression testing on every release.

- Offer user support and training.

Tools and Resources:

- Real-time monitoring tools: Grafana, Prometheus, Kibana, Moira.
- Helpdesk software: Zendesk
- Backup solutions, native to database.
- Testing tools with all the documented cases for regression testing: Allure TestOps
- Incident / bug report / documentation management software: Jira, Confluence

9. OOP Design Patterns in AutoPLM implementation

Patterns, prioritized by the importance:

1. **Factory Pattern**:
   a. As our system deals with numerous instances/objects of each class (e.g. documents of a single type, employee of a single team etc), their creation in the scalable system must be handled seamlessly

   b. Instances/objects can be enriched with database integration methods, which are responsible for managing a particular entry in the mentioned database (e.g. team123.addEmployee(Employee456))

   c. Thus, the factory may also work as a GET query handler, which wraps the results from a database in a logical layer

2. **Abstract Factory Pattern (combined with the pattern above)**:
   a. As our system deals with few abstract classes, their subclasses fullfill the core business logic in the end, ensuring scalability of a system (e.g. subclasses of a class "Document" can be "Bill of Materials", "Design Blueprint" etc)

   b. Such subclasses may vary in methods, attributes and business logic overall, so we definitely need separate factories for them

   c. However, they must vary only in differentiating implementation (*Interface segregation principle in SOLID*), while the common logic needs to be contained in one place (*Substitution principle in SOLID*)

   d. Thus, the chain of inheritance might be the following: *AbstractFactory -> AbstractDocumentFactory -> DesignDocumentFactory (Open-closed principle in SOLID)*

3. **Composite Pattern**:
   a. While all of the subclasses must be encapsulated, the dependency injections will be useful to handle complex connections between related subclasses of the same level, without causing circular dependencies *(Dependency inversion principle in SOLID)*

   b. The pattern may be used to represent hierarchical structures such as departments containing employees, tasks containing subtasks, or documents containing sections, when the changes in one class inevitably lead to the changes in another

4. **Singleton Pattern**:
   a. The business logic scenarios (named as “usecases/services/etc”), must represent a single instance in the end, initialized only once to be used in the controllers or message consumers (*Single responsibility principle in SOLID*)

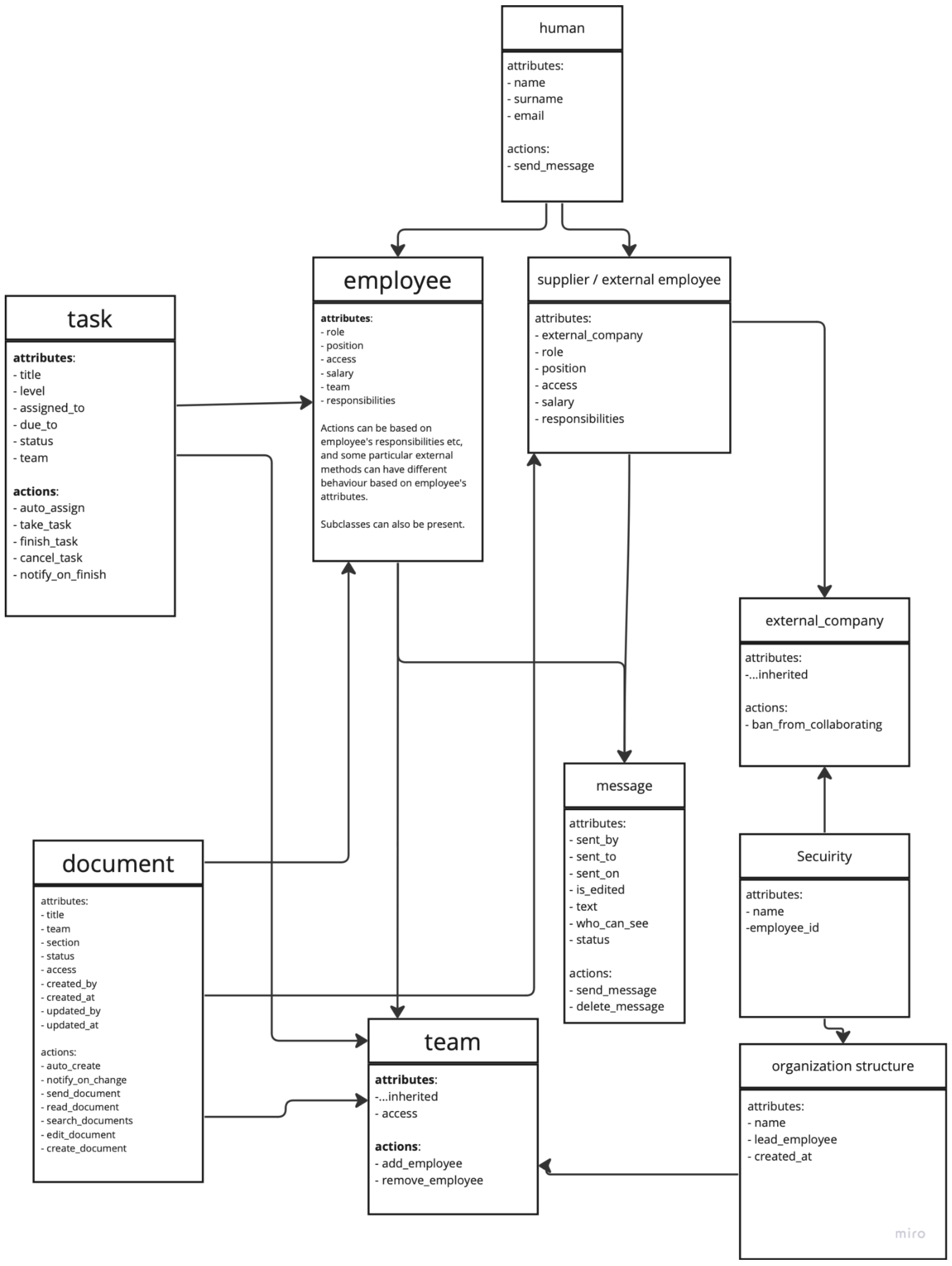
   b. Nevertheless, they also need to maintain the inheritance from abstract classes (*e.g. AbstractService* -> *AbstractEditService* -> *EditMessageService*)

5. **Builder Pattern**:
   a. In our project, this is a utility pattern, designed to simplify dynamic queries and transformations

   b. For a particular business logic, e.g., we may need to get all the draft documents for an Operations team: *DocumentFactory.addTeamFilter(‘operations’).addStatusFilter(‘draft)...*

   c. This approach assures that the lower-level operations are encapsulated, while the higher-level logic manages only the logical conditions for queries

10. Strategic Reuse in AutoPLM implementation



Pic.1 AutoPLM class diagram

Our automotive company's reuse strategy is well thought out, focusing on both internal and external resources to enhance system reusability:

10.1. Code Modules

Abstract Design: Our code modules are designed with abstraction in mind, allowing them to be easily reused across different parts of the system. This abstraction ensures that modules can be adapted to various scenarios without modification, which is a key aspect of code reuse. We utilize a `Document` class for creating instances of `TextDocument`, `SecureDocument`, and other similar classes. This design adheres to the Open Closed Principle (OCP) of SOLID principles, as new factory subclasses can be introduced without modifying the base factory, thereby promoting reusability and extensibility.

```
// Single Responsibility Principle
class Person {
public:  virtual void send_message() = 0;};

class Employee : public Person {
public:  void send_message() override {  }};

// Open-Closed Principle
class Document {
public:
  virtual void send_message() = 0;
  virtual void delete_message() = 0;
};

class TextDocument : public Document {
public:
  void send_message() override {  }
  void delete_message() override {  }};

class SecureDocument : public Document {
public:
  void send_message() override { }
  void delete_message() override { }};

// Liskov Substitution Principle
class Task {
public:
  virtual void assign_to_employee() = 0;
  virtual void cancel_task() = 0;
};

class DevelopmentTask : public Task {
public:
  void assign_to_employee() override {  }
  void cancel_task() override {  }};

class MarketingTask : public Task {
public:
  void assign_to_employee() override {  }
  void cancel_task() override {  }};

// Interface Segregation Principle
class TeamLead {
public:
  virtual void add_employee() = 0;
};

class Manager : public TeamLead {
public:
  void add_employee() override {  }
};

// Dependency Inversion Principle
class MessageSender {
public:  virtual void send_message() = 0;};

class EmailSender : public MessageSender {
public:  void send_message() override { }};

class Employee {
private:
public:  void send_message() {  }};
```

10.2. External Libraries

- **Custom snippets**: We create custom snippet modules that encapsulate common functionality. These modules can be shared across projects, reducing redundancy and enhancing maintainability. An example could be a utility module for handling date and time operations in our ***"task"*** modules.

- **Third-Party Packages**: Our company integrates third-party packages that meet specific requirements. This allows for rapid development and leveraging community contributions, thus reducing the time spent on building foundational components from scratch.

  a) **Java Libraries:**
  Hibernate*:* For Object-Relational Mapping (ORM), simplifying SQL database access.
  Spring Framework*:* Provides comprehensive infrastructure support for developing Java applications.
  Apache Commons*:* Offers utility classes and functions to simplify common programming tasks.
  Log4j*:* For logging purposes.

  b) **JavaScript Libraries:**

  React: Popular for building user interfaces, especially single-page applications.
  Express.js: A minimal and flexible Node.js web application framework. ***Socket.IO:*** Enables real-time, bidirectional, and event-based communication between the browser and the server.

  c) **C/C++ Libraries:**
  Boost*:* A set of libraries that extend the functionality of C++.
  Qt: A cross-platform application framework for desktop, embedded, and mobile applications.
  OpenCV: A computer vision library for our security systems.
  Eigen: A high-level C++ library of template headers for linear algebra, matrix and vector operations, numerical solvers, and related algorithms.

  d) **Python Libraries:**
  NumPy*:* For numerical computations and data manipulation.
  Pandas*:* For data manipulation and analysis of our databases.
  TensorFlow and PyTorch*:* For machine learning and deep learning applications of the AutoPLM.
  Requests*:* For making HTTP requests.

- **Forked Packages**: Occasionally, we fork and adapt open-source packages from GitHub to suit our specific requirements. This approach allows us to modify the package while benefiting from the original author's work and community contributions.

10.3. Documentation

- **Reusable Skeletons and Templates**: By providing documentation with reusable skeletons, we facilitate consistency and ease of maintenance. These templates ensure that new developers follow established patterns and practices, which is crucial for maintaining reusability.

- **Module Descriptions**: Clear descriptions of reusable modules are essential for future developers. They provide context and understanding of the module's purpose, which is vital for effective reuse.

  Person (Abstract):
  Handles basic personal information (name, surname, email).
  Declares a virtual `send_message(message)` for concrete implementations.
  Document (Abstract):
  Holds document content and collaborator list.
  Declares virtual methods for `send_message(message)` and `delete_message(message)`
  Task (Abstract):
  Contains task description and status
  Declares virtual methods for `assign_to_employee(employee)` and `cancel_task()`
  MessageSender (Interface):
  Declares a virtual `send_message(message)` for uniform message sending

- **Guidelines**: We establish guidelines that emphasize key principles. These guidelines serve as a compass during both development and code review processes, ensuring that the code remains clean, maintainable, and reusable:

  1. **SOLID principles**
  2. **DRY principle**
  3. **“The Power of 10” Rules**

10.4. Version Control

- **Git**: Using Git for version control enables seamless navigation and searching of modules. It provides a robust history tracking feature, which is beneficial for identifying when and why changes were made, aiding in the reuse of stable versions of code.

10.5. Integrated Development Environment (IDE)

- **VS Code:** With VS Code, we can implement interfaces and apply class design patterns. Its built-in error detection helps catch issues early, ensuring that implementations remain consistent and reusable.

10.6. Database Structure

- **Components:** abstract entities, such as *employees* or *tasks*, can be multi-purpose and vary in particular properties, which are responsible for specific business logic and subentities (e.g. *QA Tester* is a subentity of *employee*, and info about it is stored in abstract table)
- **Types and enums**: types/enums, such as *status* (active/disabled/archived), can be reused throughout the database schema implementation.
- **Stored procedures:** responsible for avoiding SQL queries' duplication.
- **Normalization:** starting from third normal form, it insures the modularity of tables and, therefore, the reusability.

10.7. Other Artifacts

- **Code Snippets / Helpers:** small pieces of code that solve specific problems and tasks
- **Configuration Files**: for CI/CD pipelines across multiple repositories with less lines of deployment/integration/testing code.
- **User Profiles or User Personas:** to describe each role, position, chosen function for each employees related on theirs responsibilities
- **Data Dictionary**: to outline the data architecture and structure of our system, ensuring that all necessary data is accounted for before development begins.
- **Meeting Notes**: Notes from meeting between devops team to inform about drawbacks for platform development in a more efficient way
- **Risk Assessments**: to prevent risks which are related to development and implementation of the system and providing guidelines on what should do, must do, etc.
- **The Compiled Application**: final artifact, representing the fully developed and tested application that is ready for deployment. It includes all versions from early prototypes to the final version of the system.

  Reusing Artifacts: To save time and resources by avoiding the need to develop the same functionality from scratch.
- **Ignoring Artifacts**: to ignore certain artifacts that do not align with the project's goals or

requirements.

- **Neglecting Artifacts**: to avoid neglecting artifacts for establishment a systematic approach to artifact management that includes regular reviews, updates, and maintenance.

11. Notes on Flaws and Improvements of this Documentation

1. A mentioned overemphasis on Supply Chain Management in the Model Description was compensated with more focus on another aspects of the model.
2. A justification for choosing the approach (Spiral Model) was enriched and extended.
3. Methodologies were also graded and examined deeply in order to choose one of them and to justify the selection.
4. Hardware and software architecture of AutoPLM was additionally documented and presented as multiple diagrams, also provided with DevOps practices.
5. Use Case diagram was reworked with appropriate schema, while State Transition Diagram now include several references to particular classes of the system.
6. Plans were also provided with class examples.
7. Reuse strategy was extended with some more artifacts that are helpful, such as: database structure or configuration files.