

ENGINEERING PROGRESS MANUAL





Phase 4: Programming and Electronics

CONTENTS

Purpose, structure and goals

- 1.1 Purpose of the area
- 1.2 Roles and responsibilities
- 1.3 Philosophy

Electronics and wiring protocol

- 2.1 Electrical layout design and planning
- 2.2 Wiring and connection standards
- 2.3 Connection checks and safe power-on

Development protocol

- 3.1 Development platforms and environments
- 3.2 Version control and collaboration
- 3.3 Code structure standards

Testing, calibration and debugging protocol

- 4.1 Testing pyramid
- 4.2 Tuning
- 4.3 Autonomous testing and calibration
- 4.4 Fine tuning

Competition handoff and pit support

- 5.1 Code preparation
- 5.2 Pit checklist

Documentation

- 6.1 Purpose of documentation
- 6.2 Engineering Notebook
- 6.3 Team technical log
- 6.4 Autonomous and routine documentation
- 6.5 Code and system documentation
- 6.6 Tech Binder (optional)
- 6.7 Backups and information control





PURPOSE, STRUCTURE AND GOALS

1.1 Purpose of the Area

- Design, build and integrate the robot's control system (hardware + software) to ensure:
- Precise subsystem control
- Stable communication between devices
- Ability to adapt at competition

1.2 Roles and Responsibilities

- Software Lead

Defines the code architecture

Approves pull requests

Owns the final deployment

- Electronics Lead

Designs the electrical system

Oversees wiring and safety

Approves power-ons

- Subsystem Programmers

Write the logic for each mechanism

Run unit tests

- Integration Technician

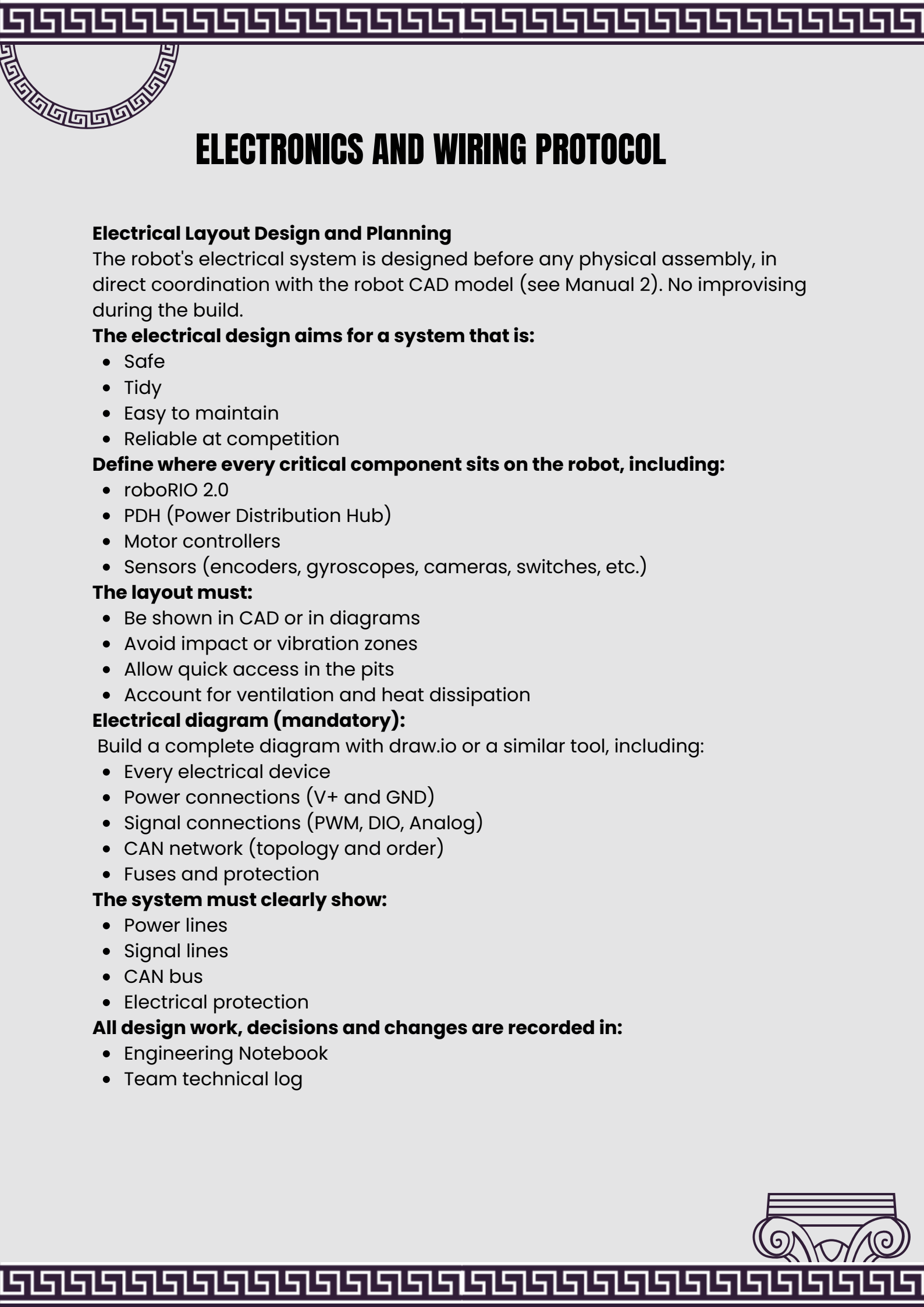
Coordinates hardware/software testing

Finds failures in how systems interact

1.3 Working Philosophy

- Every change must be reproducible
- Every system must be explainable
- Every component must be documented
- Nothing gets connected without a plan





ELECTRONICS AND WIRING PROTOCOL

Electrical Layout Design and Planning

The robot's electrical system is designed before any physical assembly, in direct coordination with the robot CAD model (see Manual 2). No improvising during the build.

The electrical design aims for a system that is:

- Safe
- Tidy
- Easy to maintain
- Reliable at competition

Define where every critical component sits on the robot, including:

- roboRIO 2.0
- PDH (Power Distribution Hub)
- Motor controllers
- Sensors (encoders, gyroscopes, cameras, switches, etc.)

The layout must:

- Be shown in CAD or in diagrams
- Avoid impact or vibration zones
- Allow quick access in the pits
- Account for ventilation and heat dissipation

Electrical diagram (mandatory):

Build a complete diagram with draw.io or a similar tool, including:

- Every electrical device
- Power connections (V+ and GND)
- Signal connections (PWM, DIO, Analog)
- CAN network (topology and order)
- Fuses and protection

The system must clearly show:

- Power lines
- Signal lines
- CAN bus
- Electrical protection

All design work, decisions and changes are recorded in:

- Engineering Notebook
- Team technical log





Wiring and Connection Standards

Robot wiring follows strict standards for reliability, easy maintenance and fast troubleshooting.

Color code (mandatory):

- Red: positive voltage (V+)
- Black: ground (GND)
- Yellow / Blue / Green: signals

Good wiring practices:

- **Routing:**
 - Keep wires away from moving parts
 - Avoid sharp edges
 - Separate power from signal when possible
- **Management:**
 - Use zip ties or velcro
 - Never force or pull on wires
 - Keep wiring organized by subsystem
- **Connectors:**
 - Fully inserted
 - Use retention
 - No loose connections
- **Labeling (mandatory):**
 - Label both ends
 - Include name and function
 - Example: INTAKE_MOTOR_LEFT

Minimum wire gauges (FRC):

Aplicación	Calibre mínimo (AWG)
Batería principal	6 AWG
Motores principales	12 AWG
Señales y sensores	18–24 AWG

The wrong gauge can cause overheating, voltage drop and intermittent faults.





Connection Checks and Safe Power-On

Before powering the robot, run a full check of the electrical system. This is mandatory on first assembly and after any change.

PRE-POWER-ON checklist:

Verificación	Estado (✓ / ✗)	Notas
No hay corto V+ / GND		
Polaridad correcta		
Sin cables expuestos		
Fusibles correctos		
Componentes firmemente montados		

First power-on procedure:

1. Use a current-limited power supply (if possible)
2. Power up with no active mechanical load
3. Check the LEDs on the controllers and roboRIO
4. Confirm there are no:
 - Burning smells
 - Overheating
 - Unexpected reboots
5. Connect the main battery
6. Run basic tests from the Driver Station

Safety rules:

- Never leave the robot powered on unattended
- Keep the power disconnect within immediate reach
- Avoid working on a powered robot without control





DEVELOPMENT PROTOCOL

Development Platforms and Environments

Robot software is developed in a standard environment that keeps everyone compatible, stable and able to collaborate.

Official team standard:

- Language: Java
- IDE: Visual Studio Code + WPILib extension
- Version control: GitHub

Required tools:

- GitHub Desktop (visual management of repositories and branches)
- WPILib installed and up to date
- roboRIO drivers and tools
- Access to the team's official repository

Minimum setup per member:

Every programmer needs:

- A working WPILib environment
- The ability to clone, commit and push to the repository
- Access to the project documentation
- A working connection to the robot (USB/Ethernet/WiFi depending on stage)

Core rule:

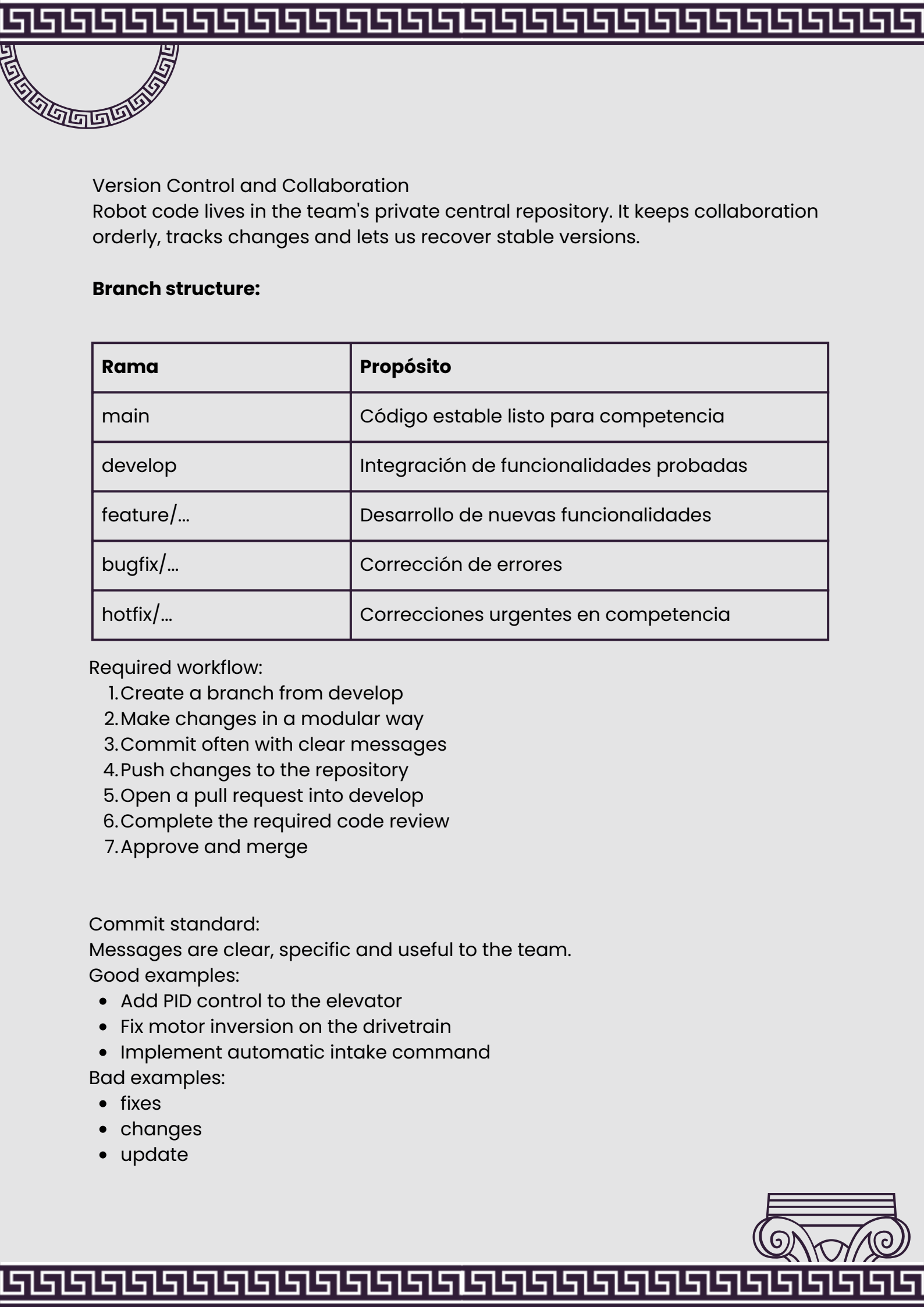
All development uses official FRC tools.

Environments that are not compatible and could compromise deployment or integration are not allowed.

Good environment practices:

- Keep WPILib up to date (without breaking compatibility mid-season)
- Make sure the code builds before pushing changes
- Never work directly on main
- Keep versions consistent across team members





Version Control and Collaboration

Robot code lives in the team's private central repository. It keeps collaboration orderly, tracks changes and lets us recover stable versions.

Branch structure:

Rama	Propósito
main	Código estable listo para competencia
develop	Integración de funcionalidades probadas
feature/...	Desarrollo de nuevas funcionalidades
bugfix/...	Corrección de errores
hotfix/...	Correcciones urgentes en competencia

Required workflow:

1. Create a branch from develop
2. Make changes in a modular way
3. Commit often with clear messages
4. Push changes to the repository
5. Open a pull request into develop
6. Complete the required code review
7. Approve and merge

Commit standard:

Messages are clear, specific and useful to the team.

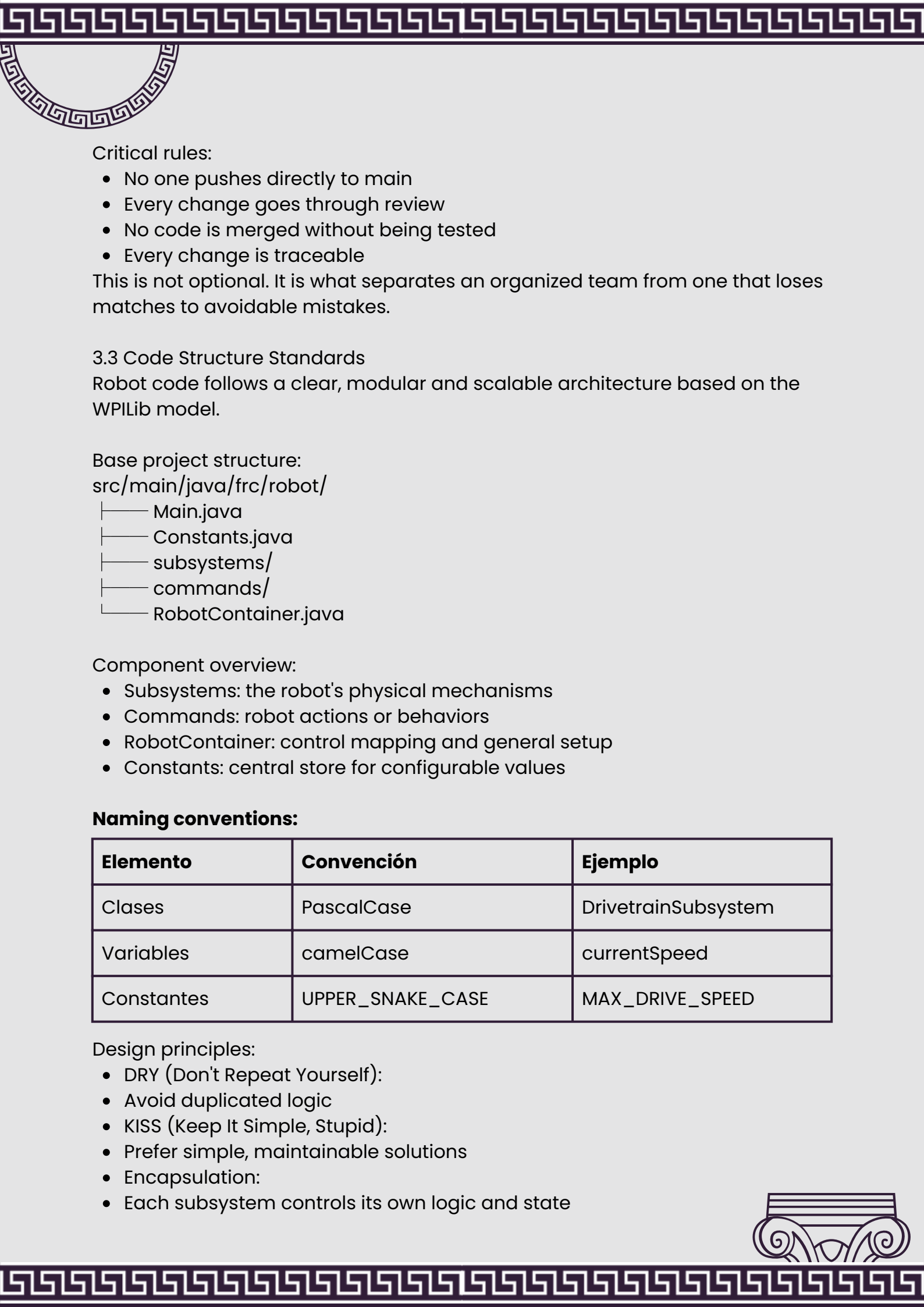
Good examples:

- Add PID control to the elevator
- Fix motor inversion on the drivetrain
- Implement automatic intake command

Bad examples:

- fixes
- changes
- update





Critical rules:

- No one pushes directly to main
- Every change goes through review
- No code is merged without being tested
- Every change is traceable

This is not optional. It is what separates an organized team from one that loses matches to avoidable mistakes.

3.3 Code Structure Standards

Robot code follows a clear, modular and scalable architecture based on the WPILib model.

Base project structure:

```
src/main/java/frc/robot/  
├── Main.java  
├── Constants.java  
├── subsystems/  
├── commands/  
└── RobotContainer.java
```

Component overview:

- Subsystems: the robot's physical mechanisms
- Commands: robot actions or behaviors
- RobotContainer: control mapping and general setup
- Constants: central store for configurable values

Naming conventions:

Elemento	Convención	Ejemplo
Clases	PascalCase	DrivetrainSubsystem
Variables	camelCase	currentSpeed
Constantes	UPPER_SNAKE_CASE	MAX_DRIVE_SPEED

Design principles:

- DRY (Don't Repeat Yourself):
- Avoid duplicated logic
- KISS (Keep It Simple, Stupid):
- Prefer simple, maintainable solutions
- Encapsulation:
- Each subsystem controls its own logic and state



Additional good practices:

- Do not mix logic from several subsystems in one class
- Avoid hardcoded values outside Constants.java
- Keep functions short and clear
- Put readability ahead of premature optimization

3.4 Technical Documentation and Engineering Notebook (CRITICAL)

Documentation is not optional. It is part of the engineering process and of how the team is judged.

All development is recorded in:

- Engineering Notebook
- Team technical log

Documentation includes:

- Design decisions (why a solution was chosen)
- Major code changes
- Tuning values (PID, offsets, speeds, etc.)
- Problems found
- Solutions applied
- Test results

Documentation in the code:

The code should be understandable without outside explanation.

Rules:

- Comment the why, not the what
- Document key functions with Javadoc

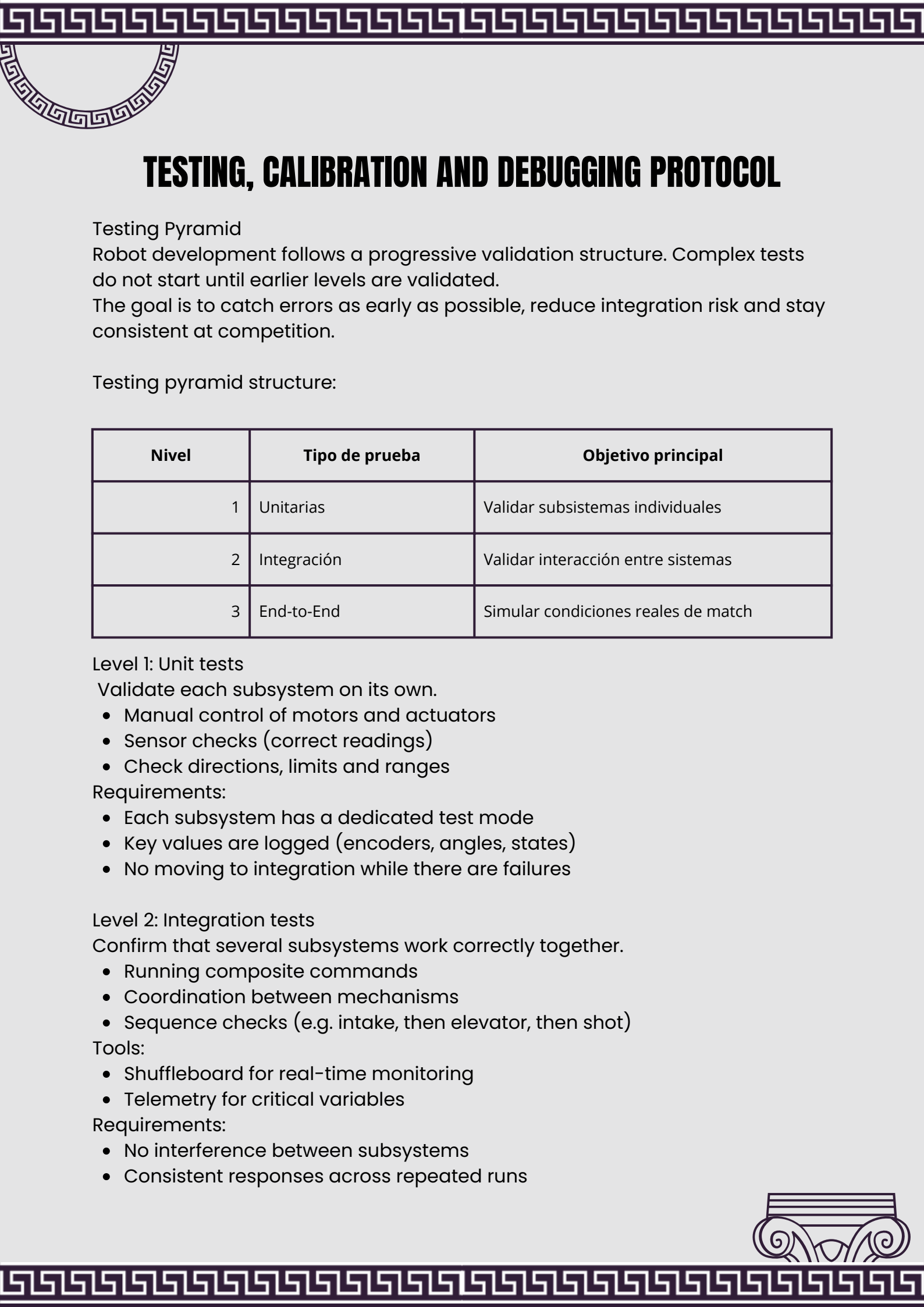
Example:

```
/**  
 * Calculates the best shooting angle based on distance to the target.  
 * @param distance Distance in meters measured by vision  
 * @return Angle in degrees  
 */
```

Documentation goals:

- Keep continuity between seasons
- Make debugging easier
- Speed up training for new members
- Back up technical decisions for judges





TESTING, CALIBRATION AND DEBUGGING PROTOCOL

Testing Pyramid

Robot development follows a progressive validation structure. Complex tests do not start until earlier levels are validated.
The goal is to catch errors as early as possible, reduce integration risk and stay consistent at competition.

Testing pyramid structure:

Nivel	Tipo de prueba	Objetivo principal
1	Unitarias	Validar subsistemas individuales
2	Integración	Validar interacción entre sistemas
3	End-to-End	Simular condiciones reales de match

Level 1: Unit tests

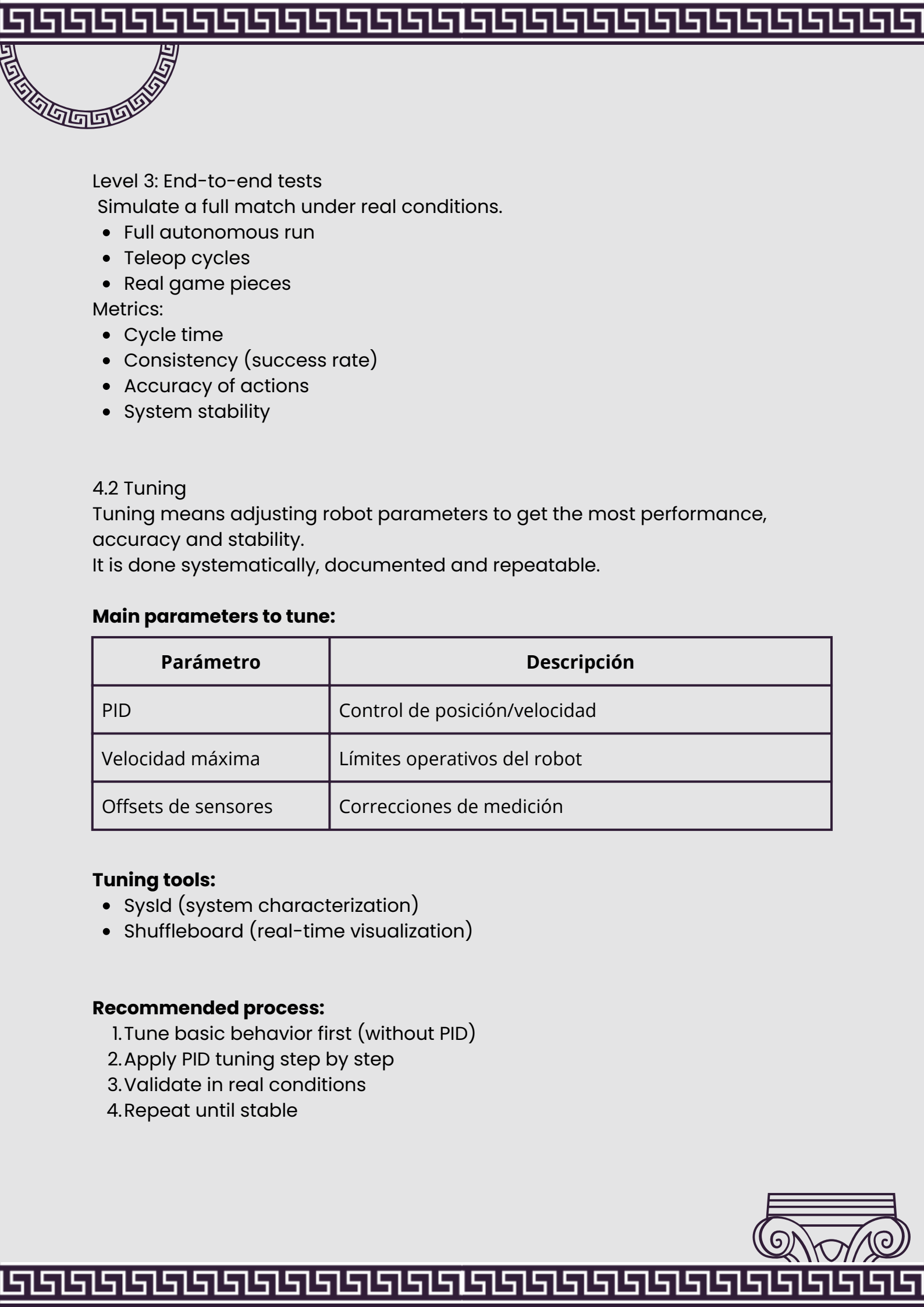
- Validate each subsystem on its own.
- Manual control of motors and actuators
 - Sensor checks (correct readings)
 - Check directions, limits and ranges
- Requirements:
- Each subsystem has a dedicated test mode
 - Key values are logged (encoders, angles, states)
 - No moving to integration while there are failures

Level 2: Integration tests

- Confirm that several subsystems work correctly together.
- Running composite commands
 - Coordination between mechanisms
 - Sequence checks (e.g. intake, then elevator, then shot)

- Tools:
- Shuffleboard for real-time monitoring
 - Telemetry for critical variables
- Requirements:
- No interference between subsystems
 - Consistent responses across repeated runs





Level 3: End-to-end tests

Simulate a full match under real conditions.

- Full autonomous run
- Teleop cycles
- Real game pieces

Metrics:

- Cycle time
- Consistency (success rate)
- Accuracy of actions
- System stability

4.2 Tuning

Tuning means adjusting robot parameters to get the most performance, accuracy and stability.

It is done systematically, documented and repeatable.

Main parameters to tune:

Parámetro	Descripción
PID	Control de posición/velocidad
Velocidad máxima	Límites operativos del robot
Offsets de sensores	Correcciones de medición

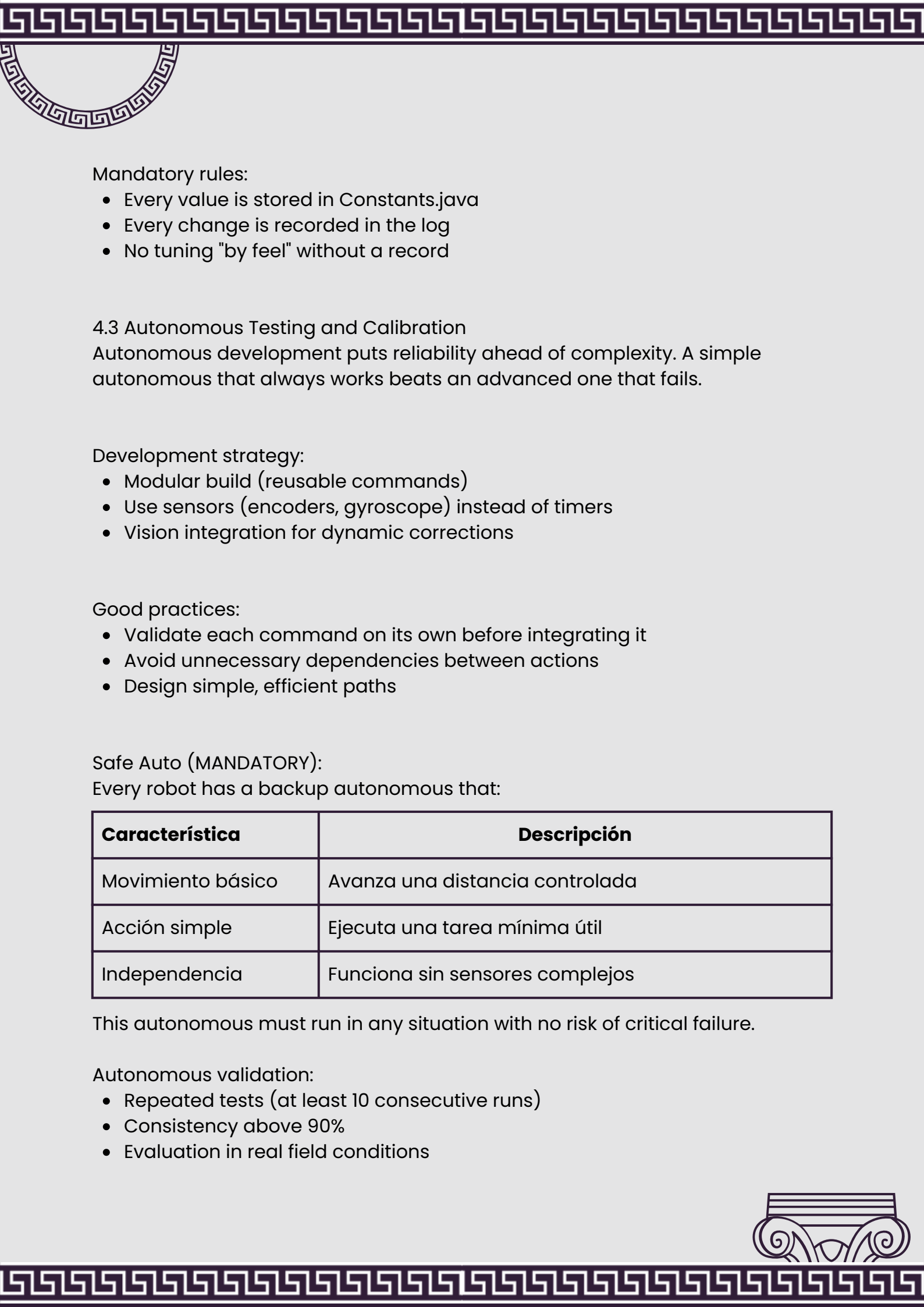
Tuning tools:

- SysId (system characterization)
- Shuffleboard (real-time visualization)

Recommended process:

- 1.Tune basic behavior first (without PID)
- 2.Apply PID tuning step by step
- 3.Validate in real conditions
- 4.Repeat until stable





Mandatory rules:

- Every value is stored in Constants.java
- Every change is recorded in the log
- No tuning "by feel" without a record

4.3 Autonomous Testing and Calibration

Autonomous development puts reliability ahead of complexity. A simple autonomous that always works beats an advanced one that fails.

Development strategy:

- Modular build (reusable commands)
- Use sensors (encoders, gyroscope) instead of timers
- Vision integration for dynamic corrections

Good practices:

- Validate each command on its own before integrating it
- Avoid unnecessary dependencies between actions
- Design simple, efficient paths

Safe Auto (MANDATORY):

Every robot has a backup autonomous that:

Característica	Descripción
Movimiento básico	Avanza una distancia controlada
Acción simple	Ejecuta una tarea mínima útil
Independencia	Funciona sin sensores complejos

This autonomous must run in any situation with no risk of critical failure.

Autonomous validation:

- Repeated tests (at least 10 consecutive runs)
- Consistency above 90%
- Evaluation in real field conditions





4.4 Fine Tuning

Fine tuning means making final adjustments under real operating conditions to get the best performance from the robot.

Goals:

- Shorter cycle times
- Better accuracy
- More consistency

What to optimize:

- Mechanism speeds
- Transitions between actions
- Response times
- Fine PID adjustments

Method:

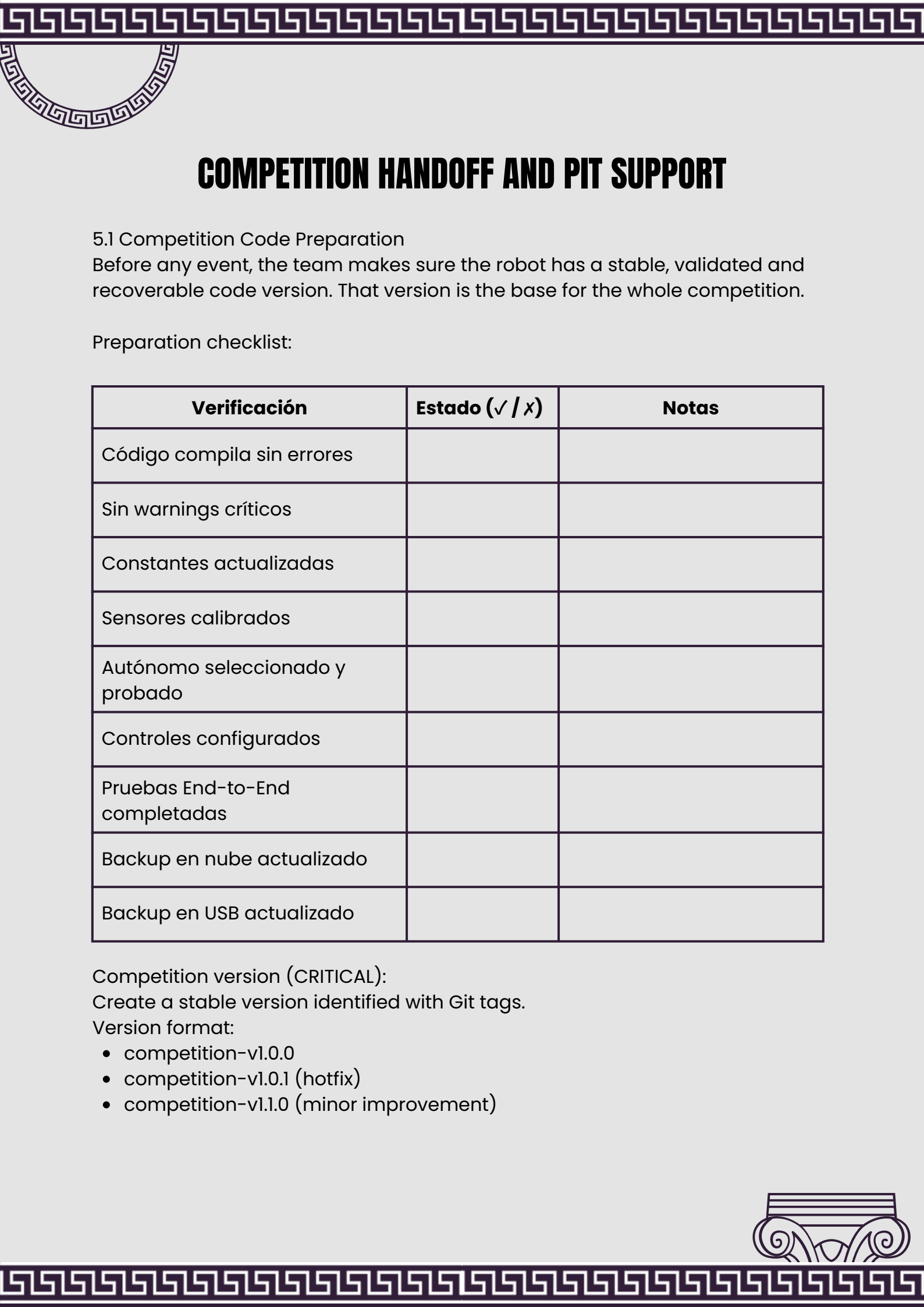
- Make small, controlled changes
- Test right after each adjustment
- Compare results with earlier metrics

Key rule:

Consistency matters more than top speed.

A slightly slower but reliable robot wins more matches than a fast but unstable one.





COMPETITION HANDOFF AND PIT SUPPORT

5.1 Competition Code Preparation

Before any event, the team makes sure the robot has a stable, validated and recoverable code version. That version is the base for the whole competition.

Preparation checklist:

Verificación	Estado (✓ / ✗)	Notas
Código compila sin errores		
Sin warnings críticos		
Constantes actualizadas		
Sensores calibrados		
Autónomo seleccionado y probado		
Controles configurados		
Pruebas End-to-End completadas		
Backup en nube actualizado		
Backup en USB actualizado		

Competition version (CRITICAL):
Create a stable version identified with Git tags.
Version format:

- competition-v1.0.0
- competition-v1.0.1 (hotfix)
- competition-v1.1.0 (minor improvement)





This lets us:

- Go back to a working version in seconds
- Never lose stable code
- Stay in control of changes at competition

Rules before competition:

- No last-minute changes without full testing
- All code on main is stable
- We can deploy the robot from scratch at any time

Environment preparation:

- Main laptop set up and tested
- Project builds without an internet connection
- Required tools installed and working

5.2 Pit Checklist and Competition Support

At competition, the goal is to solve problems fast, without mistakes and without improvising.

Minimum pit crew:

Elemento	Estado (✓ / x)	Notas
Laptop de desarrollo		
Cables (USB / Ethernet)		
USB con respaldo completo		
Instaladores WPILib		
Drivers del roboRIO		
Código de competencia		
Notas de troubleshooting		





Required contents of the backup USB:

- Source code for the competition version
- WPILib installers
- Required drivers
- Earlier working versions
- Basic system documentation

Change protocol at competition (CRITICAL)

At an event, every change is controlled. Unmanaged changes are the most common cause of match failures.

Required process:

1. Find the problem and its likely cause
2. Confirm the change is really needed
3. Get approval from the Software Lead
4. Create a hotfix/... branch from the current version
5. Make the smallest change needed
6. Test on the robot right away
7. Validate full operation (not just the fix)
8. Merge and create a new tag
9. Update the USB backup

Critical rules:

- No large changes at competition
- Never test code for the first time in a match
- Always test before deploying
- Every change must be reversible

Quick diagnosis in the pits

The team must be able to find faults in minutes.

Common things to check:

- Connection to the robot (Driver Station)
- roboRIO status
- CAN bus active
- Sensors reporting data
- Code deployed correctly





DOCUMENTATION

6.1 Purpose of Documentation

Documentation in the programming and electronics area aims to:

- Record technical decisions and why they were made
- Keep continuity between seasons
- Make debugging and maintenance easier
- Build solid evidence for judges
- Standardize knowledge across the team

All documentation must be clear, accessible, current and useful, not just decorative.

6.2 Engineering Notebook

The Engineering Notebook is the team's official record of its engineering process and must clearly cover everything related to software and electronics.

Required content for programming and electronics:

- Code architecture (project structure)
- Electrical system diagram
- Hardware-software integration
- Design decisions (e.g. sensor use, control logic)
- Autonomous strategy
- Subsystem descriptions and how they are controlled

Development record:

- Code iterations
- Problems found
- Solutions applied
- Major system changes

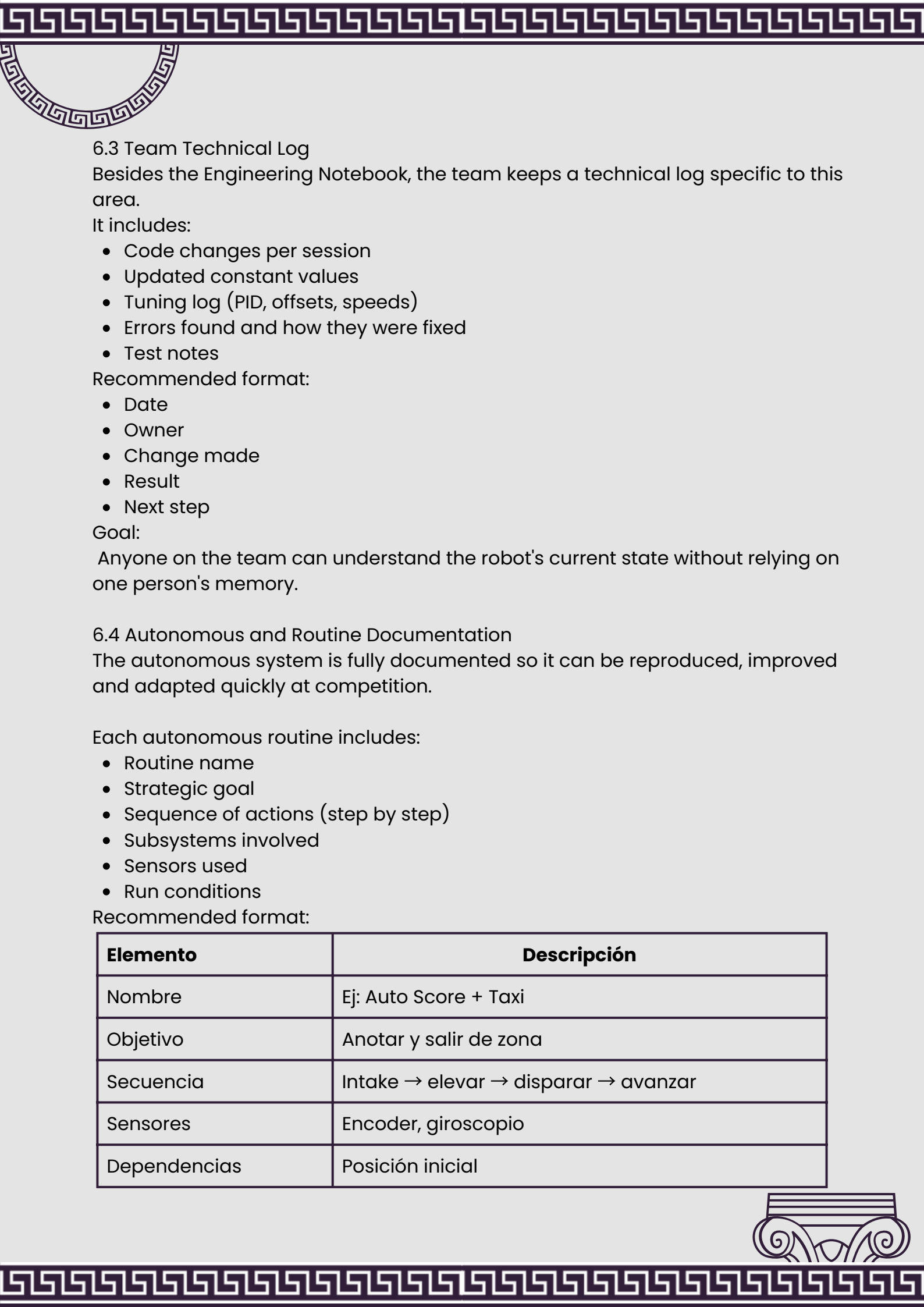
Test record:

- Results of unit, integration and end-to-end tests
- Tuning values
- Adjustments made and their results

Good practices:

- Include diagrams, images and screenshots
- Keep chronological order
- Explain the "why" behind decisions





6.3 Team Technical Log

Besides the Engineering Notebook, the team keeps a technical log specific to this area.

It includes:

- Code changes per session
- Updated constant values
- Tuning log (PID, offsets, speeds)
- Errors found and how they were fixed
- Test notes

Recommended format:

- Date
- Owner
- Change made
- Result
- Next step

Goal:

Anyone on the team can understand the robot's current state without relying on one person's memory.

6.4 Autonomous and Routine Documentation

The autonomous system is fully documented so it can be reproduced, improved and adapted quickly at competition.

Each autonomous routine includes:

- Routine name
- Strategic goal
- Sequence of actions (step by step)
- Subsystems involved
- Sensors used
- Run conditions

Recommended format:

Elemento	Descripción
Nombre	Ej: Auto Score + Taxi
Objetivo	Anotar y salir de zona
Secuencia	Intake → elevar → disparar → avanzar
Sensores	Encoder, giroscopio
Dependencias	Posición inicial





Also:

- Path diagram (if applicable)
- Related code or command references
- Test results and consistency

6.5 Code and System Documentation

Code should explain itself and be backed by technical documentation.

Requirements:

- Comments explain the "why"
- Javadoc on key functions
- Clear organization by subsystem

Document:

- What each subsystem does
- How commands relate
- Sensor use
- Control logic

Constants.java (CRITICAL):

- All constants are centralized
- Every value has documented context
- Changes are recorded in the log

6.6 Tech Binder (Optional)

The Tech Binder is a physical or digital reference built for the pits and presentations.

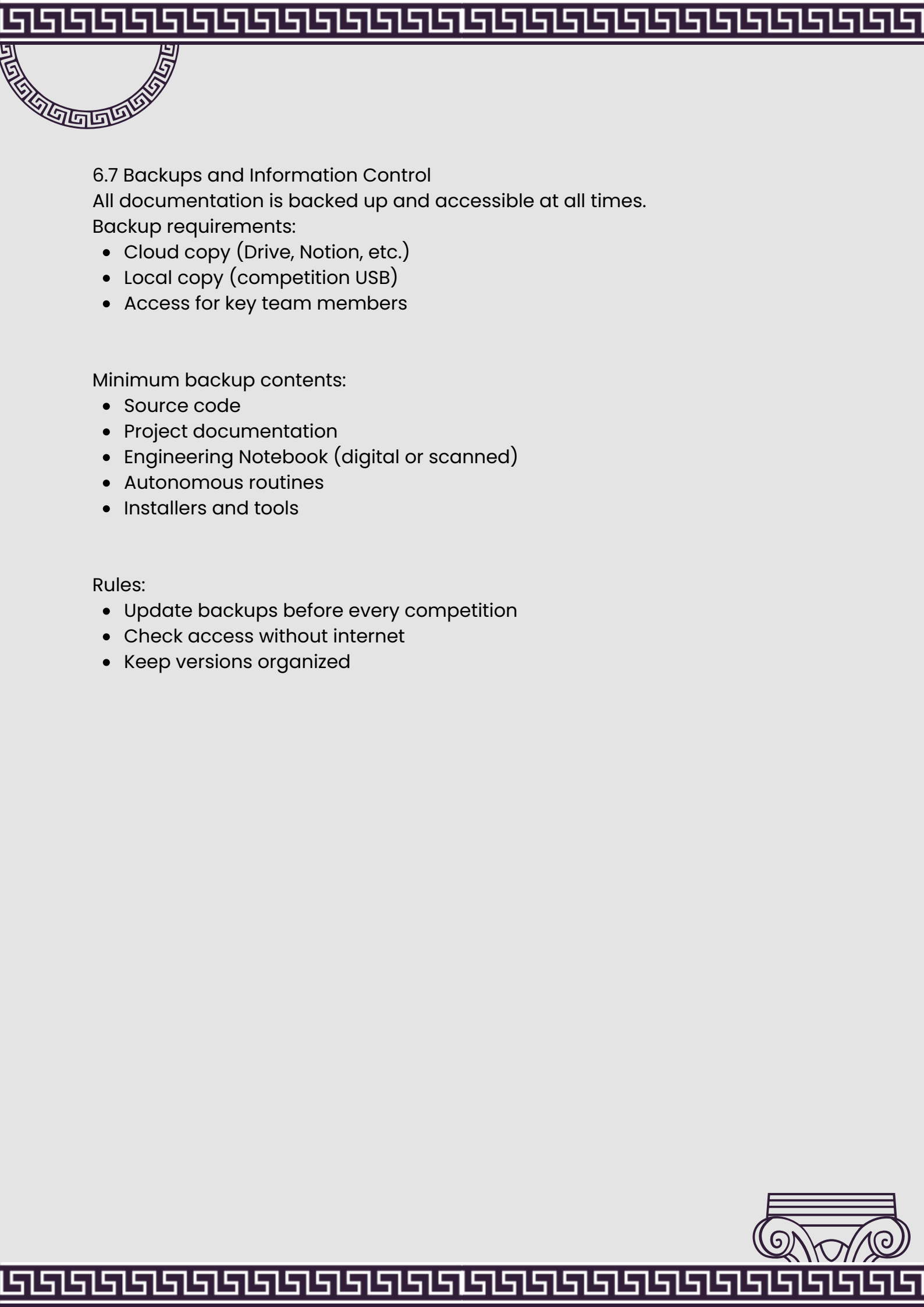
It includes:

- Electrical diagram
- Code architecture
- Subsystem list
- Autonomous routines
- Basic troubleshooting
- Relevant code versions

Goal:

- Quick access to critical information at competition
- Debugging support
- Support in judge interviews





6.7 Backups and Information Control

All documentation is backed up and accessible at all times.

Backup requirements:

- Cloud copy (Drive, Notion, etc.)
- Local copy (competition USB)
- Access for key team members

Minimum backup contents:

- Source code
- Project documentation
- Engineering Notebook (digital or scanned)
- Autonomous routines
- Installers and tools

Rules:

- Update backups before every competition
- Check access without internet
- Keep versions organized

