

# Software Development Methods School Year 2019/2020

2<sup>nd</sup> Phase Report

# Magellan's Race - 500 years

Alexandre Correia (53298) Hugo Rodrigues (53309) Ricardo Ferreira (52915) Vasco Martins (53299) Lab P4 | João Araújo 04/12/2019

# Table of Contents

Assumptions	3
Analysis Phase	4
Use Case Diagram	4
Use Case Specifications	5
Register Team	5
Use Case Specification	5
Alternative Flows Specifications	6
Form Crew	7
Use Case Specification	7
Alternative Flows Specifications	8
High-Sea Technical Reparations	8
Use Case Specification	9
Alternative Flows Specifications	10
Refill Provisions	11
Use Case Specification	11
Alternative Flows Specifications	12
Send Message to Family	13
Use Case Specification	13
Alternative Flows Specifications	14
Activity Diagrams	15
Register Team	15
Form Crew	15
High-Sea Technical Reparations	16
Refill Provisions	17
Contact Family	18
Sequence Diagrams	19
Register Team	19
Form Crew	20
High-Sea Technical Reparations	21
Refill Provisions	22

Contact Family	23
Class Diagram	24
OCL	25
Invariants	25
Pre-Conditions	25
Post-Conditions	26
Statechart Diagrams	27
Leading And Communication	27
Trading Control	27
Team Manager	28
Design Phase	29
Design Diagram	29
Package Diagram	30
Component Diagram	31
Deployment Diagram	32
Conclusions	33
Bibliography	34

### Assumptions

We assume that the Race Organizer "manages" a team, this means, the Race Organizer is responsible for register and disqualify a team. When the Race Organizer wants to register a new team to the race, it asks the Skipper for every information about the boat and the Support Team. About the disqualifying process, we assume that the only thing that the OBR does is to report a violation committed by the team to the Race Organizer, being the Race Organizer the one that disqualifies the team.

It was assumed that all the communications are made through a communication system. When the conversation ends, the system notifies the OBR with the exchange messages. The OBR will check if it was committed a violation, and if so he reports to the Race Organizer.

While sailing the team can order provisions and/or request shelter to the Local Authorities of the port they are arriving.

## Analysis Phase

Use Case Diagram



### Use Case Specifications

**Register Team** 

#### **Use Case Specification**

Name: Register Team

**Id:** 1

#### Description: Register a team in the system

#### Actors:

- Main: Race Organizer
- Secondary: None

#### Pre-conditions: None

#### Main-Flow:

- 1. The UC starts when the Race Organizer decides to register a team.
- 2. The Race Organizer inserts the team's name.
- 3. The system validates the team's name.
- 4. The Race Organizer adds an On-Board Reporter to the team.
- 5. The system validates the On-Board Reporter.
- 6. The Race Organizer adds the Skipper to the team.
- 7. The system validates the Skipper.
- 8. The Race Organizer submits the boat information.
- 9. The system validates the boat.
- 10. The Race Organizer submits a list of the Support Team members.
- 11. The system validates the Support Team.
- 12. The system sends a notification saying "Team registered".
- 13. The UC ends.

#### **Alternative Flows:**

- Name associated to another team.
- OBR associated to another team.
- Support team associated to another team.
- Ship does not meet the requirements.

Post-Conditions: Team registered in the system.

#### **Alternative Flows Specifications**

Name: OBR associated to another team

**Id:** 1.1

**Description:** Describes the steps when an OBR is associated to another team **Actors:** 

- Main: Race Organizer
- Secondary: OBR

**Pre-conditions:** OBR is associated to another team **Main-Flow:** 

- 1. The UC starts when the system does not validate the OBR.
- 2. The system notifies the Race Organizer that the OBR is associated to another team.
- 3. The system provides the available OBRs.
- 4. Returns to step 4 of the "Register team" Main-flow.

Alternative Flows: None Post-Conditions: None

Name: Ship does not meet the requirements

**Id:** 1.2

**Description:** Describes the steps when a ship does not meet the race requirements **Actors:** 

- Main: Race Organizer
- Secondary: Skipper

**Pre-conditions:** Ship does not have the correct requirements **Main-Flow:** 

- 1. The UC starts when the system does not validate the ship.
- 2. The system notifies that the ship does not meet the race requirements.
- 3. Returns to step 8 of the "Register team" Main-flow.

#### Alternative Flows: None

Post-Conditions: None

#### Form Crew

#### **Use Case Specification**

#### Name: Form Crew

**Id:** 2

Description: Updates a team's crew for next leg

#### Actors:

- Main: Skipper
- Secondary: Race Organizer

#### **Pre-conditions:**

- Team already registered
- Boat docked

#### Main-Flow:

- 1. The UC starts when the Skipper decides to change the crew before a leg or an In-Port Race.
- 2. The Skipper submits the male/female ratio.
- 3. The system validates the ratio.
- 4. The Skipper submits the information about each crew member.
- 5. The system validates the new crew members.
- 6. The system notifies the Race Organizer about the team's crew composition.
- 7. The UC ends.

#### **Alternative Flows:**

- Male/Female ratio is not valid.
- Crew composition doesn't match male/female ratio.

Post-Conditions: Crew has been updated in the system.

#### **Alternative Flows Specifications**

Name: Male/Female ratio not valid Id: 2.1 Description: Describes the steps when the ratio is not approved Actors:

- Main: Skipper
- Secondary: None

**Pre-conditions:** Genre ratio is not valid **Main-Flow:** 

- 1. The UC starts when the ratio is not approved
- 2. The system notifies the Skipper that the ratio is not valid.
- 3. The system provides the acceptable ratios.
- 4. Returns to step 2 of the "Form Crew" Main-Flow.

Alternative Flows: None Post-Conditions: None

Name: Crew composition doesn't match male/female ratio

Id: 2.2

**Description:** Describes the steps when the crew composition does not match the male/female ratio **Actors:** 

- Main: Skipper
- Secondary: None

**Pre-conditions:** Crew composition doesn't match male/female ratio **Main-Flow:** 

- 1. The UC starts when the crew composition does not match the male/female ratio
- 2. The system notifies the Skipper that crew composition does not match the male/female ratio.
- 3. The system requests a new composition.
- 4. Returns to step 4 of the "Form Crew" Main-Flow.

Alternative Flows: None Post-Conditions: None

#### High-Sea Technical Reparations

#### **Use Case Specification**

Name: High-sea technical reparations

#### **Id:** 3

#### Description: Describes the steps of a high-sea technical reparation

#### Actors:

- Main: Skipper
- Secondary: Support Team, OBR

#### Pre-conditions: The boat is not docked

#### Main-Flow:

- 1. The UC starts when the Skipper requests technical support or when the support team detects a problem.
- 2. The system establishes a connection between the Skipper and the Support Team.
- 3. The Skipper identifies the problem.
- 4. The system notifies the Support Team with a description of the problem.
- 5. The Support Team gives technical advice.
- 6. The system sends the Support Team's advice.
- 7. The Skipper fixes the problem.
- 8. The Skipper confirms that the problem has been fixed.
- 9. The system notifies the Support Team that the problem has been fixed.
- 10. The system closes the connection.
- 11. The system notifies the OBR with the sent messages.
- 12. The UC ends.

#### **Alternative Flows:**

- Team is disqualified.
- Problem has not been fixed.

Post-Conditions: Problem fixed

#### **Alternative Flows Specifications**

Name: Notify violation Id: 3.1 Description: Describes the steps when the OBR detects a violation Actors:

- Main: OBR
- Secondary: Team, Race Organizer

Pre-conditions: Violation committed Main-Flow:

- 1. The UC starts when the OBR detects a violation in a message exchanged.
- 2. The OBR submits the information about the violation.
- 3. The system notifies the Race Organizer about the violation.
- 4. The UC ends.

Alternative Flows: None Post-Conditions: Team disqualified

#### Name: Problem has not been fixed

Id: 3.2

**Description:** Describes the steps when boat problem has not been fixed **Actors:** 

- Main: Skipper
- Secondary: Support Team

# Pre-conditions: Malfunction detected Main-Flow:

- 1. The UC starts when the boat problem has not been fixed.
- 2. The Skipper informs that the problem has not been fixed.
- 3. The system notifies the Support Team that the problem has not been fixed.
- 4. Returns to step 4 of the "High-sea technical reparations" Main-flow.

#### Alternative Flows: None

Post-Conditions: None

#### **Refill Provisions**

#### **Use Case Specification**

Name: Refill provisions

#### **Id:** 4

Description: Describes the steps for the team to buy provisions from the local authorities.

#### Actors:

- Main: Skipper
- Secondary: Crew Member, Local Authorities

#### Pre-conditions: The boat is docked

#### Main-Flow:

- 1. The UC starts when the Skipper wants to order products.
- 2. The Skipper assigns a Crew Member to order the products through the system.
- 3. The system notifies the assigned Crew Member.
- 4. The assigned Crew Member submits the list of products to the system.
- 5. The system sends the list to the Local Authorities.
- 6. The Local Authorities check the stock of the requested products.
- 7. The Local Authorities submits that they have those products in stock.
- 8. The system notifies the assigned Crew Member that he/she can order the products.
- 9. The assigned Crew Member confirms the order.
- 10. The system saves the order.
- 11. The UC ends.

#### **Alternative Flows:**

- Products are not available
- Fresh food is not allowed

Post-Conditions: Provisions have been sold

#### **Alternative Flows Specifications**

Name: Products are not available

**Id:** 4.1

**Description:** Describes the steps when the local authorities do not have the requested products **Actors:** 

- Main: Local Authorities
- Secondary: Crew Member

**Pre-conditions:** Products have been requested **Main-Flow:** 

- 1. The UC start when Local Authorities do not have the products in stock.
- 2. The Local Authorities inform the system that the products are not available.
- 3. Returns to step 11 of the "Refill provisions" Main-flow.

Alternative Flows: None Post-Conditions: None

Name: Fresh food is not allowedId: 4.2Description: Describes the steps when fresh food is not allowedActors:

- Main: Local Authorities
- Secondary: Crew Member

**Pre-conditions:** Fresh food have been requested **Main-Flow:** 

- 1. The UC starts when the Crew Member includes fresh food in the list of products.
- 2. The system notifies the Crew Member that fresh food is not allowed on board.
- 3. Returns to step 4 of the "Refill provisions" Main-flow.

Alternative Flows: None Post-Conditions: None

#### Send Message to Family

#### **Use Case Specification**

#### Name: Contact family

#### **Id:** 5

**Description:** Describes the steps when a crew member wants to send messages to his/her family.

#### Actors:

- Main: Crew Member
- Secondary: OBR, Communication System

#### Pre-conditions: Crew Member is sailing

#### Main-Flow:

- 1. The UC starts when the crew member wants to send messages to his/her family.
- 2. The Crew Member requests to the system to contact family.
- 3. The system informs the OBR that the crew member wants to contact family.
- 4. The OBR approves the request.
- 5. The system establishes a connection between the Crew Member and the Communication System.
- 6. While the Crew Member wants to send another message.
  - 6.1. The Crew Member submits the message.
  - 6.2. The system sends the message to the Communication System.
  - 6.3. The Communication System notifies about a response.
  - 6.4. The system shows the response to Crew Member.
- 7. The Crew Member tells the system to end the connection.
- 8. The system notifies the OBR about the messages exchanged.
- 9. The UC ends.

#### **Alternative Flows:**

- Notify violation.
- OBR doesn't allow the Crew Member to contact family.

Post-Conditions: Crew Member contacted family

#### **Alternative Flows Specifications**

**Note:** The alternative flow "Notify violation" in the use case "Contact family" is described by the "Notify violation" use case, which id is 3.1.

Name: OBR doesn't allow the Crew Member to contact family

ld: 5.1

**Description:** Describes the steps when the OBR does not approve the crew member to contact family

Actors:

- Main: OBR
- Secondary: Crew Member

**Pre-conditions:** OBR didn't allow the Crew Member to contact family **Main-Flow:** 

- 1. The UC starts when the OBR doesn't allow the Crew Member to contact family.
- 2. The system informs the Crew Member that is not possible to contact family.
- 3. Returns to step 10 of the "Contact family" Main-flow.

#### Alternative Flows: None

Post-Conditions: None

### Activity Diagrams Register Team



#### Form Crew



#### High-Sea Technical Reparations



#### **Refill Provisions**



#### **Contact Family**



# Sequence Diagrams

### Register Team

nteraction SequenceDiagram
actors econtrols econtrols edutas ed
loop [name not valid]
1 registerTeam(teamName) 2 registerTeamName) 2 rind(teamName)
[name not valid]
<ul> <li>Strateger (strateger (strateger</li></ul>
6 : m(error)
7 : rn(nl)
1. Team
9: m(success)
[loop [2087 net valid]]
11: issedua/OBR/OBRID, 11)
4 i associativam(Ummut, 1) 12 i find (ORRID)
[68# retval]
15: trajeren
[088 valid] 17: m(obr)
18 : set08R(0r)
e
Teop (Skoper net valid.)
21 [associauSkipper(5kipper(5,1)
22 sacciauSkipper(D)(1) 22 find Skicose(D)
2
25 : rm(error)
26 : m(erre)
S
27 - et Share (kinon)
au : m(succes)
loop [Ship does not meet the requirements]
\$1 : sascelawBear(BearD, t)
44- Associate usati usati usati 33 meefaquiements[Boat]0
24 mm/240
35 (molecul) 35 (molecul)
27 micros
28. sediceģicaj
v: mpucceasy months and the second se
(leop TeanSupportNembers tet not valie)
42: associative provintian (support flam) (support
[learn-seporthemeans lat not valid]
43 m/geno) 43 m/geno)
4
49 : setSupport=amMemberghe()
50 - this (success)

#### Form Crew

Skipper: Actor	SkipperInterface: Interface	TeamLeading: Control	«enumeration» GenderRatio	«data» Team	t1: Team
^ qc					
[ratio not valid]	2 : gubmitD	lotio(rotio)			
1 : subm	itRatio(ratio)				
		3 : find(r	atio)		
alt .		<			
[ratio not valid]	<5:rtn(f	alse)	aise)		
6 : dis	playRatios				
0.00					
[ratid valid]		<7 : rtn(t	true)		
×	8 : rtn(t	true)			
9 : requ	esicrewinio				
[crew not valid]					
10 : change	Crew(crew, t1)				
	II : changeUr	rew(crew, t1)			
alt .					
[crew not valid]	<	error)			
≮13:	rtn(error)				
K	setCrowinfo				
[crew valid]			15 : find(teamID)		
		<	16 : rtn(t1)	······	
	K		17 : setCrew(cre	w)	
<	18 : rtn(su	cessful)			ų –
19 : rtn	i(sucesstul)				
		LJ			
	i i		· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	

### High-Sea Technical Reparations

«actor»	«actor»	«actor»	«interface»	«interface»	«control»	«data»	skipper: Skipper
OBR	SupportTeamMember	Skipper	Communication	SkipperInterface	Leading&Communication	OnBoardReporter	
		1 : reque	estCall(idOBR)				
			4	2 : requestCall(idOBR)	3 : find(id	IOBR)	
					↓ ≪		
		1		5	rtn(obr)		
		6 : rth(obr)					
		7 : permission					
				8:p	ermission		
					-Ų		
		-					
			K	9 : rtnPermission(false)			
ot possible]		10 : cal	IINotPossible				
ossibe]			<	11 : rtnPermission(true)			
		≪	allPossible	11.10.01.000			
			13 : problemAdvice(descrip	ation)			
				14 : problem	Advice(description)		
						«create»	
						15 :	
loop [problem not fixed]							
surer.		16 : sendMessage(me	essage, supportMemberID)				
			Ļ		«create» 17 :		
	18 : sen	dMessage(message, sk	opperID)				
					«create»		
					19 :		
L							
			20 : fixProblem(prob)				
				21 : 6v5	(noblem/orob)		
						22 : foc(void)	
					23 : notifyOl	BR(messagesExchange	d, OBR)
			_		Ļ		
¢	25 : m/oon	fication)		24 : rtn(notification)			
	23.101100	(calon)				-	
2							
violation detected]	26 submitViolatio	n(description)					
				27 : submitViolation(description	on)		
		1					1

#### **Refill Provisions**



### Contact Family

[no permission [permission given] [loop * [wants to send a message]	1 : requestCall(dOBR)     6 : re(ob)     7 : permission     10 : rePermission(false)     12 : rtnPermission(false)     13 : sentMessage(content,destination)	2 : requestCall(dOBR) 3 : finlOBR(idOBR) 4 : th(dB) 8 : permission 9 : thePermission(tale) 4 : th(dB) 11 : thePermission(tale) 14 : sontMessage(content, destination)	
[permission [permission gran]	C C C C C C C C C C C C C C C C C C C	2 : requestCall(dOBR)     3 : fndOBR(x00BR)     4 ::tn(ob)     6 : permission     6 : permission     6 : permission(fuse)     11 : tripPermission(fuse)     14 : sondMessage(content, destinaçor)	
[permission given] [copp * [vants to send a message]	C C (1900) 7 ( permission 10 (nt/Permission(futue) 12 ( nt/Permission(futue) 13 ( sentiMessage(content_destination)	3 : fm0BR(dOBR)           6 : permission           8 : permission           • . rsrPermission(fulse)           • . rsrPermission(fulse)	
[permission]	6 - rh(dot) 7 : permission 10 : m/hermission(fulse) 12 : nhPermission(fulse) 13 : sendMessage(content_destination)	6: permission           8: permission           9: rtnPermission(false)           11: rtnPermission(false)           14: sendMessage(content, destinaçion)	
[permission given]	e		
[op permission [permission guinn] [coop * [vants to pend a message]	6 - rh(ob) 7 : permission 10 : rtriPermission(lake) 12 : rtriPermission(lake) 12 : rtriPermission(true) 13 : sendMessage(content,destination)	8 : permission     9 : rsrPermission(false)	
[no permission [normission given] [comp *] [varits to end a message]	7 : permission 10 : reinPermission(table) 12 : reinPermission(true) 13 : sendMessage(content,destination)	B : permission     0 : rmPermission(false)       11 : rhiPermission(true)     14 : sondMessage(content, destination)	
[no permission [no permission [permission grain] [loop • ] [varits to permit a message]	12 : ntPermssion(lake)     12 : ntPermssion(lake)     12 : ntPermssion(true)     13 : sentMessage(content,destination)	8 : permission     9 : rsrPermission(fute)	
[permission gran] [permission gran] [coup * ] [varia to send a message]	10: miPermission(false) 12: miPermission(faule) 13: sendMessage(content_destination)		
In permission (permission given) (compare) (toop * (wants to end a message)	10 : ntPermission(lake) 12 : ntPermission(fue) 13 : sendMessage(content_destination)	G	
[no permission]	10 : ntiPermission(false) 12 : ntiPermission(rule) 13 : sendMessage(content,destination)	O : re-Permission(false)	
[permission given]	10 : rtriPermission(false) 12 : rtriPermission(true) 33 : sendMessage(content,destination)	11: mPermsson(tue)	
[permission given]	12 : triPermission(true) 13 : sendMessage(content,destination)	II: mPermission(true)  I4: sendMessage(content, destination)	
[loop * ] [(wants to send a message]	12: iniParmission(true)	14 : sendMessage(content, destinaţion)	
[toop * ] [wants to end a message]	13 : sendMessage(content,destination)	14 : sendMessage(content, destinaçon)	
[loop *_] [wants tojeend a message]	33 : sendMessage(content,destina@on)	14 : sendMessage(content, destination)	
[vants to send a message]	13 : sendMessage(content,destination)	14 : sendMessage(content, destination)	
[wants to send a message]	13 : sendMessage(content,destination)	14 : sendMessage(content,destination)	
	20. Serumessage(content,desunadori)	14 : sendMessage(content,destination)	
	8		
		15: m1: Me	ssage
é		16 : rtn(messageSent)	
	17 : rtn(messageSent)		
		18 : notifyOBR(messageList,dbr)	
		19 : rtn(notification)	
	< 20 : rtn(ngtification)		
opt .			
[violation detected]	21 : rubmit/(alation(description)		
	21. submitviolation(description)	22 - submit/foldsion(description)	

### Class Diagram



#### OCL

#### Invariants

#### context MediaContent

inv:

#### MediaContent.allInstances->isUnique(i | i.ID)

#### context MediaContent

inv:

```
MediaContent.allInstances->forAll(i | i.views >= 0)
```

#### context Team

inv:

Team.allInstances->forAll(i | not i.supportTeam->isEmpty())

#### **Pre-Conditions**

context RaceOrganizer::disqualifyTeam(team: Team): void

pre:

self.teamsDisqualified->excludes(team)

context CrewMember::sendMessage(message: Message): void

pre:

self.messagesSent->count(message) = 0

context Team::setSkipper(skipper: Skipper): void

pre:

self.skipper.oclIsUndefined()

#### **Post-Conditions**

context RaceOrganizer::disqualifyTeam(team: Team): void

post:

self.teamsDisqualified->includes(team)

context CrewMember::sendMessage(message: Message): void

post:

self.messagesSent->size() = self.messagesSent@pre->size + 1

context SupportTeamMember::sendMessage(message: Message): void

post:

self.messagesSent->includes(message)

### Statechart Diagrams Leading And Communication



### Trading Control



### Team Manager



# Design Phase Design Diagram



## Package Diagram



### **Component Diagram**



### Deployment Diagram



### Conclusions

With this project, it was possible for us to apply the theoretical knowledge taught in the classes in a very practical situation. We learnt a lot about the development of a complex project from scratch and about the difficulties of satisfying the client needs.

The project had a very large content and was very abstract, hard to determine the necessary situations to develop. To avoid this fact, it was written at the beginning of this report our assumptions about the statement ambiguities.

We also felt difficulties doing the StateCharts Diagrams, on the fact that it was not practiced on the practical classes.

# Bibliography

- Classes digital support.
- http://www.db.informatik.uni-bremen.de/projects/USE/use-documentation.pdf