Question

What does "high level" mean?

- bad hacker jargon in movies: "high level encryption"
- news: "a high-level briefing from a high-ranking source"

Q: What does it mean to be high-ranking?

- in charge of a lot of people / stuff
- have to pay attention to the big picture and long term

Sometimes leadership means digging into details, because the details matter, but if a leader spends all of their time digging into details, they'll never be able to focus on big-picture stuff like strategic direction and vision.

High- vs low-level

Typically *relative* terms

High(er) level:

Bigger picture, fewer details (more **abstract**)

Low(er) level:

More focused, more details (more **concrete**)

Words of caution

These terms are *relative* and not *absolute*

Abstractions *leak*

Imperfect but useful

There is a myth that you can have a clean separation between high- and low-level thinking.

- In organizations, everyone needs some idea of the big picture (**what we're doing and why we're doing it**), and the people at the top can't be hermetically sealed off from **how the organization actually works on a day-to-day basis**.
- In technology, abstractions are **leaky**. I can have a high-level abstraction for "a screen to display things", but that screen will behave very differently if it's an OLED on a Grove kit or a phone screen, or a tablet screen, or a laptop or a 4K 27" display!

This way of categorizing the world is imperfect and can be messy, but it is still very ______________.

Top-down design

a.k.a., functional decomposition

a.k.a., specification refinement

Figure out the big-picture requirements

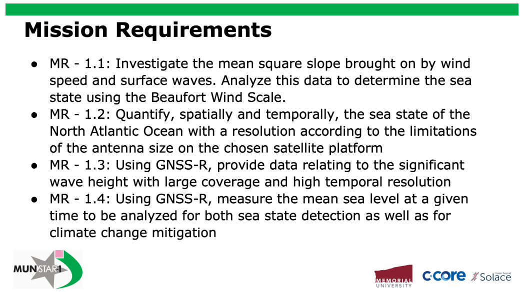
What does this thing we're designing need to **accomplish**?
What is it that our **users** need?

Break into problems we can actually solve (iterative)

Example

MR 1.3

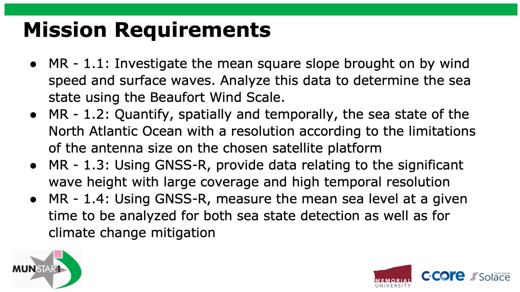
Using GNSS-R, provide data relating to the significant wave height with large coverage and high temporal resolution



Example

MR 1.3

Using GNSS-R, provide data relating to the significant wave height with large coverage and high temporal resolution



Example

MR 1.3

Using GNSS-R, **provide data** relating to the significant wave height with large coverage and high temporal resolution

Mission Requirements

- MR - 1.1: Investigate the mean square slope brought on by wind speed and surface waves. Analyze this data to determine the sea state using the Beaufort Wind Scale.
- MR - 1.2: Quantify, spatially and temporally, the sea state of the North Atlantic Ocean with a resolution according to the limitations of the antenna size on the chosen satellite platform
- MR - 1.3: Using GNSS-R, provide data relating to the significant wave height with large coverage and high temporal resolution
- MR - 1.4: Using GNSS-R, measure the mean sea level at a given time to be analyzed for both sea state detection as well as for climate change mitigation

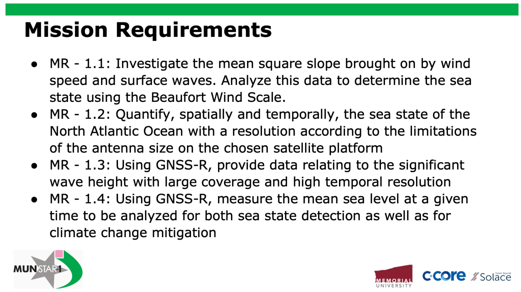
MUNSTAR-I

MEMORIAL UNIVERSITY C-CORE Solace

Example

MR 1.3

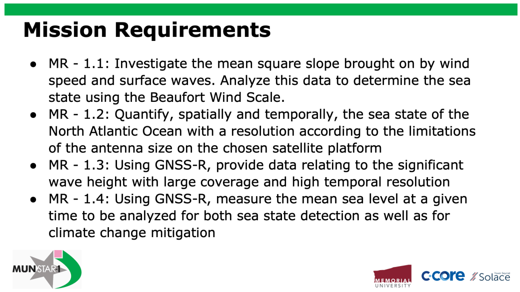
Using GNSS-R, provide data relating to the **significant wave height** with large coverage and high temporal resolution



Example

MR 1.3

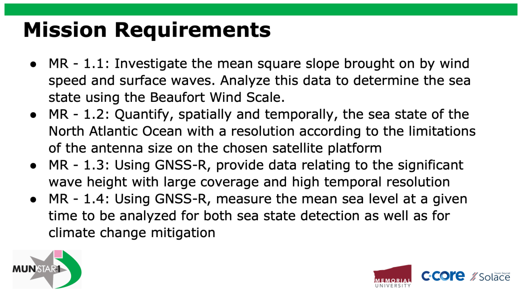
Using GNSS-R, provide data relating to the significant wave height with **large coverage** and high temporal resolution



Example

MR 1.3

Using GNSS-R, provide data relating to the significant wave height with large coverage and **high temporal resolution**



Example flow-downs

Using GNSS-R: need to *receive* and *interpret* GNSS signals (both original and reflected)

- GNSS receiver
- radio receiver(s)
- something to interpret reflected GPS signals
- some way to store the resulting data

Example flow-downs

provide data: need to *transmit*, *store* and *provide access* to data

- on-satellite storage
- communication with ground
- storage on the ground
- public interface to data

Example flow-downs

significant wave height: drives interpretation of GNSS signals

- algorithm(s) to estimate wave heights
- impacts on specifications of receivers, etc.

Example flow-downs

large coverage and **high temporal resolution:** impacts on designs of *antenna*, *receiver*, etc.

- antenna and receiver must support large coverage
- whole system must be ready to receive more data *quickly* (implications for how data is passed from one system to another, how long each system can take, etc.)

Result

High-level system "block diagram":

- set of components
- what those components do (**very** abstract description)
- relationships among them (communication)

Then keep going!

Break systems into subsystems, subsystems into smaller subsystems, until they're small enough to implement

Functional decomposition

Break a problem down into smaller parts

Keep going until we reach *functions*

- functions that haven't been written yet!
- functions whose behaviour we can **specify**

Contracts

Design by contract:

- agree on *what* code does before *how*
- write clear *preconditions*, *postconditions* and *invariants*
- check with *assertions*

Preconditions

Things your code can *assume* to be true

```
def geometric_mean(values):
    """Compute the geometric mean of an iterable collection of values.

    Parameters
    ----------
    values
        a collection of non-negative numbers
    """
```

- your code will do something **iff** preconditions are met
- if not... all bets are off!

If a pre-condition is violated, your code is allowed to do anything: return the wrong answer, throw an exception (we'll learn about those in Terms 3 and up), halt and catch fire...

Postconditions

Things your code must make true

```
def foo(xs):
    """Do something.

    Postcondition: xs will have an even number of values

    Returns the number of values that foo the wibble (>= 0)
    """
```

- **iff** all preconditions are met, your code is responsible for ensuring that the postconditions are met

This code example has two postconditions: one is explicitly noted using the word "postcondition", but the one about the return value is also a postcondition!

Invariants

Things that must *always* be true

- act as preconditions *and* postconditions
- mostly relevant to things that keep state (objects, modules with global variables)
- good example: a `Student` object's name is not `None`
- bad example: a `Student` object's name never chagnes

This example of a bad invariant is something that doesn't relate to the way students actually work: it's a ____________ assumption designed to make the ______________________________ rather than to make the ______________________. Other lazy assumptions include:

- everyone has a first name and a last name
- everyone goes by their first name

So what?

What do we do with these things?

1. Identify them and **state them clearly**
2. Question them and **let clients question them**
3. Check them:

```
def sqrt(x):
    """[...]
    x : a non-negative number
    """

    assert x >= 0
```


An assertion is an excellent way of both ____________________ your assumptions and also ________________ them. Instead of silently corrupting data (as can often happen when your implicit assumptions are violated), an assertion will bring the issue to the fore and ____________ ________________.

Design by contract

Clearly related to test-driven development (TDD)!

- top-down *refinement* of specifications
- specification-driven *preconditions*, *post-conditions* and *invariants*
- specification-driven *testing*

... all of which can happen *before* you write any code!

Example

Poker assignment from a previous year:

- given two hands represented by 5-tuples of strings:
  - each string is `value + ' ' + suit`
  - value is `'A'`, `'K'`, `'Q'`, `'J'`, `'10'`, `'9'`, ...
  - suit is `'C'`, `'D'`, `'H'`, `'S'`
- return 1 or -1 if one hand wins; return 0 if they tie

You'll see in the video explanation, solving this problem provides a great opportunity for breaking a big problem down into smaller parts and solving those parts independently.