Team Lead Cheatsheet

What Should I handle?

- Several topics
  - common things
  - work distribution
  - processes
  - HR
- This **is not** a set of rules
  - ... it is more or less just a couple of thoughts and ideas
  - let yourself be inspired

Common Things

- Be firm and consistent
  - also to yourself!
- The main goal is to learn something
  - do not prefer performance at the expense of learning
  - including knowledge sharing

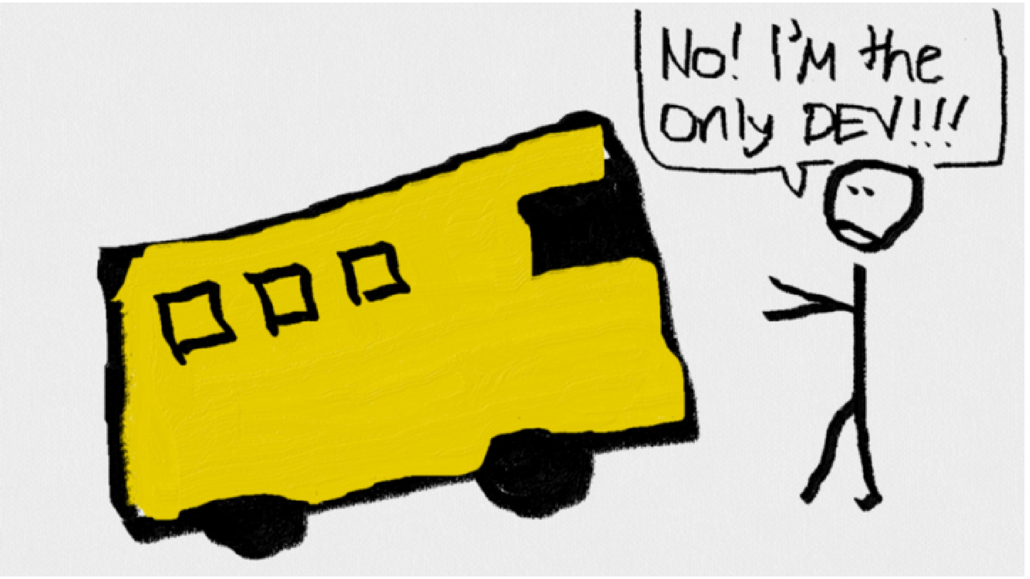
Performance Optimization

- *Divide et impera* style
  - each member is an expert in his/hers domain
- Useful in *short-term projects*
  - less substitutability
  - similar to sprint races
- Try to avoid this style of team leading
  - Even though a semestral project is a *short-term project*

Knowledge Optimization

- Almost necessary in *long-term projects*
  - similar to a Marathon race
- Prefer pair or mob to solo programming
- Open discussions about design, coding style, ...
  - It takes time, but it is worth it
- Does someone know any particular technology you are about to use?
  - Let him/her guide some other member to incorporate that
  - The knowledge should be **shared**

Bus Factor



> “ The bus factor refers to the number of people in your team who can put your project in trouble if they are hit by a bus. ”

(The Bus Factor: link)

- If the bus factor of your project is **1** **YOU HAVE A SERIOUS PROBLEM**

Roles of Leadership

- From your perspective, there are two roles:
  - work coordination
  - technical supervision
- You can take both roles
  - but it is not a shame to be just the first one
- Coordination takes time and effort
  - Good coordination seems to be standard
  - Bad coordination causes significant issues

More Eyes, Less Bugs

- Refers to knowledge sharing
- Encourage members to cooperate on every phase of the project
- You can easily overlook a mistake you have produced
  - in the code
  - in the design
  - in the DB schema

Watch Your Morale

- Watch your pace
    - avoid fast start
    - avoid sprint on deadline
- Continually contribute to your project

Work Distribution

Pair Programming

- Preferred over solo programming
- Stable pairs
    - effective after synchronization
        - especially design and implementation decisions
    - might lead to lesser knowledge sharing
- Rotating pairs
    - harder to get to know each other
    - higher knowledge sharing

Pair Programming (cont...)

- Pair of Dev and QA
  - knowledge sharing remained
  - less effective in design and implementation decisions

Task Allocation

- Vertical (**#fullstack**)
  - slightly slower development
    - everybody does everything
  - better knowledge sharing
- Horizontal
  - creates experts
  - watch your **bus factor**

Testing

- Write unit tests
  - manual testing is also useful
- Who does the testing?
  - the same pair?
  - or the other pair?

Testing (cont...)

- Same pair does testing
  - faster
  - the author's blindness
- The other pair
  - slower
  - higher knowledge sharing

Code Review

- Opposite to design
- Done by one person
  - faster
  - watch your **bus factor**
- Done by whole team
  - slower
  - knowledge sharing

Code Organisation

- Horizontal view
  - based on functionalities
  - packages, modules
- Vertical view
  - based on level of abstraction
  - classes and their relations
- (Shake) but do not mix the layers
  - harded to achieve in vertical view

Processes

Communication

- Choose a suitable platform for communication
    - discord
    - mail
    - zoom, meet
    - phone
- You are responsible for calendar scheduling of your team members
    - use **reply-all** instead of just **reply**

Regularity

- Establish a time slot for brief meetings
  - once/twice per week
  - 10-20 minutes maximum
- Topics
  - Who did what
  - Who is about to do what
  - Any issues?

Issue Tracking

- Establish an issue tracking platform
  - gitlab issues
  - trello (Atlassian)
  - youtrack (JetBrains)
- Features vs bugs
- Make it done
  - When?
  - By whom?

DoD - Definition of Done

- Define what needs to be done in order to complete an issue
  - might be specific for some issues
  - try to keep your own team standard, however
- Thoughts...
  - Who gives the approval?
  - What about tests? Code review?
  - Does it have a defined design?

Design

- Think first before you code
- Discuss within your team
  - come up together
  - ...or independently
    - discuss with teammates anyway

Human Resources

Abnormal Members

Skillmaster

- let him/her lead the development
  - important as a designer
  - important as a code reviewer
- force him/her to less coding
  - he/her shall guide others

Abnormal Members (cont...)

Beginner

- force him/her to participate on all parts of the project
- do not allow him/her work alone
  - rotate the buddy in his/hers pair
- he/she should do the things, the buddy should organize
- prefer multiple easy tasks over few complicated ones

Conflict

- Wait till they calm down
  - but do not wait indefinitelly
- Discuss an issue without spectators
  - listen each side separately
  - then be a mediator
- **Prefer personal communication**
  - written text hides **MANY** layers of communication
  - in case of online, turn on cams

Conflict (cont...)

> “ In case everything failed, share your problems with your assigned Tech lead. **ASAP!** ”

(PV168: Course Information)

Time for Retrospective

How did you manage to lead your teams so far?