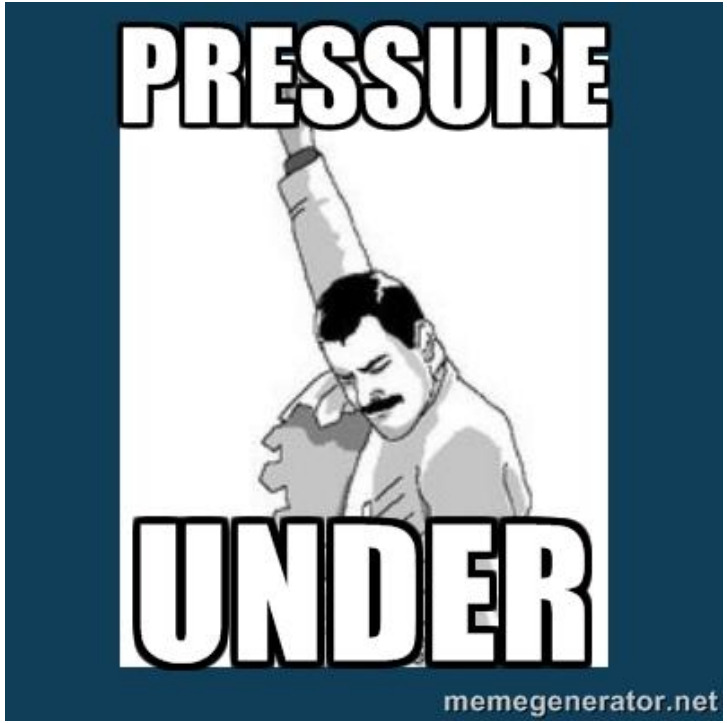
uncertainty driven development

burning mode

@oleksmarkh

some tags (unordered)

- **#mindset** #processes #practices
- #patterns #tooling #react **ecosystem**
- **#lessons learned**



#mindset

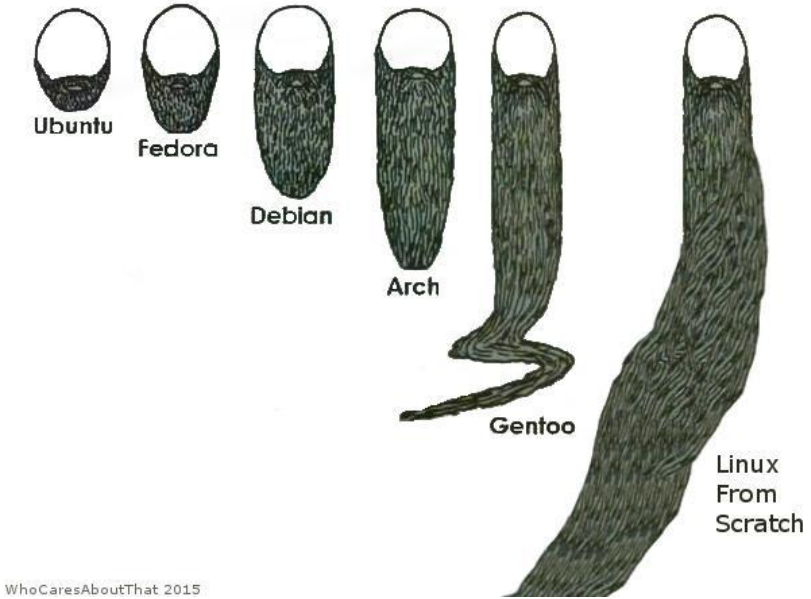
- OO vs. **functional**
  - crucial misconception: "data + actions" vs. "data -> actions" aka "monsters vs. **ninjas**"
  - patterns vs. common sense -> (don't) turn into **philosophical** debates
- DDD
- **art** of staying "in the zone" <- no matter what

#process

- invest in it (carefully)
- define uncertainties
  - e.g. UI, business logic, scalability etc.
  - arrange feature workflow accordingly
  - **revise** constantly
- isolate/**orchestrate** dependencies: **human** and technical

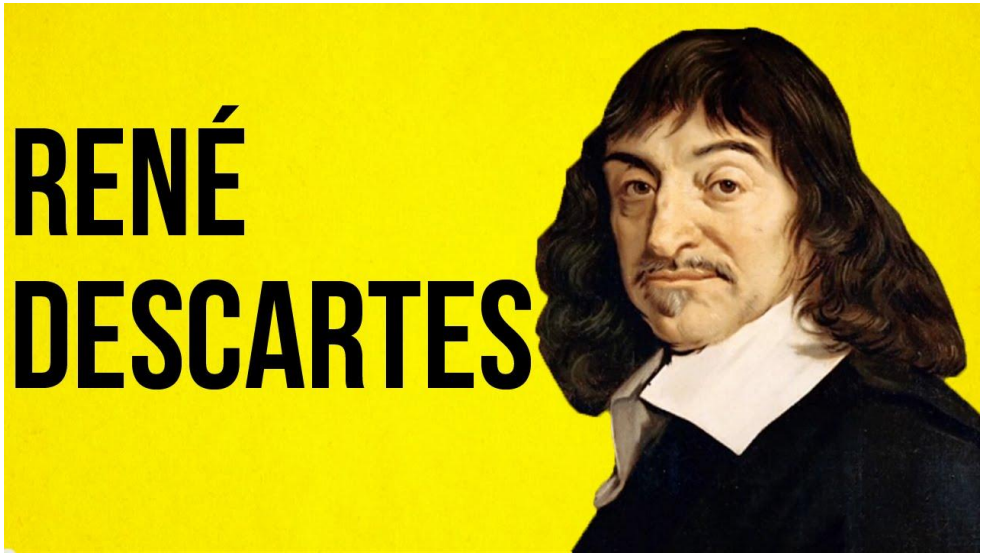
team/env setup

- role of producer <- (ninja) yellow duck (with **superpowers**)
- sharing/rotating responsibilities
- leveraging **#guilds**



conventions

- terminology
  - promote semantics and **purity** in definitions
  - garden it
- spellcheck everywhere
  - editorconfig
  - (strict) linter
  - patterns



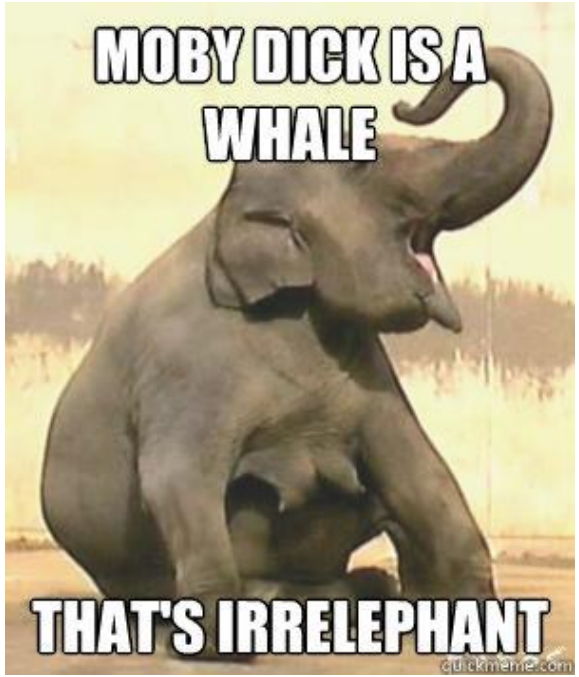
daily routines

package your work and ship it **properly**

- instant planning: "agenda" gist -> log repo + features Q&A gists -> issues
- "zero everything" principle: inbox, PRs, todos/comments, tabs

backing issues/PRs

- outline **scope** and next steps after first commit
- optional templating <- example: transport layer refactoring
- keep granular: avoid Dragons and Moby Dicks
- use labels: "in progress", "needs review" etc. and **milestones** (reflecting epics/OKRs)



refactoring <- granulated vs. massive

- fighting the mess (long run): notes -> **guidelines** -> conventions + architecture
- gardening (short run): motivation + **value** -> acceptance criteria (scope) -> next steps (out of scope)

testing/docs

- requirements validation
- intentionally breaking things: showcase **robustness** and predictability
- coverage distribution between layers: % == misleading
- reflect a **purpose** (WHY) everywhere (even in commit messages)
- create tests before addressing isolated fixes <- discipline **and** motivation

but first things first

API first

- your contract, source of truth for your state
- promote common guidelines <- http://zalando.github.io/restful-api-guidelines
- *swagger.yaml*: same language for all
- bridge non-RESTful cases **elegantly**
  - example: reporting (inversion of entities vs. metrics)
  - schema vs. convention <- https://github.com/paularmstrong/normalizr

design first

- structure <- **maintain** it
- visuals <- Zeplin (you'll love it!)

mobile first

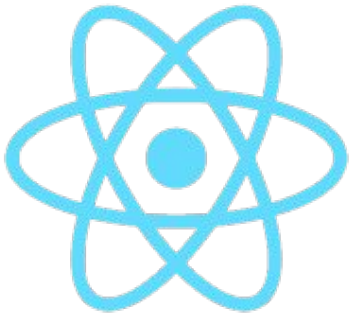
- even for the back-end mindset
- try to **think** offline: UX, performance

architecture

- avoid frameworks, use libs instead <- don't let to **tie** you **down**
- cohesion vs. coupling <- "Simple made Easy"
- orchestrating complex things <- https://www.mosaic9.org

The ecosystem

Finally :)



React

React itself

- stateless components <- so freaking easy to **comprehend**
- component workflow: design -> breakdown -> basic layout/styling -> detail -> polish
- container workflow: API -> state -> proxy down + dispatch

Redux and the others

- data flow as a "framework"
  - why to follow **rules**: immutability, purity
- more than a **predictable** state container
  - again, on terminology
  - state vs. store
  - actions vs. action creators
  - reducers

Redux and the others

solving UX issues by state **design**

- bulk collections saving
- keeping scroll position
- navigation, history replay
- … you name it

Redux and the others

some patterns

- routing as a part of application state <- ?! (ok, indirect injection)
- async actions: thunk vs. sagas
  - managing side effects <- the only possible place
  - **where** and what to dispatch <- keep **clear** separation of concerns
    - loading UX
    - error handling
  - how to interact with transport layer
    - abstract HTTP away vs. libs (`axios`, `superagent`)
- collection manipulation
- deriving subtle sub-collections vs. injecting flags into collection items

tooling

- know your boxes <- being boring again
  - Jira + Github: issues/PRs -> epics/milestones + labels (yep - colors are important)
  - tabs: browser, editor, tmux
- deployment and compliance/traceability
  - STUPS <- promote AWS workflow
  - Jenkins/Travis are your friends, but they don’t have to **stay** smart
- React + Redux devtools (log, diff, chart) <- no live demo this time :/

You Learn About It

- do **keep** things in order <- Ordnungsamt
- do plan ahead (short and middle-term), share your predictable plans with others <- helps **avoiding** chaos
- do self-document code: folder structure, consistent naming, `React.PropTypes`
- do follow (detailed) REST error codes <- helps **keeping** communication precise
- do lint and unit test each commit: git/Github hooks <- `npm i husky`, CI and chat integrations
- do **reflect** on each single bug: WHY this **happened** vs. HOW it revealed
- do produce granular artifacts <- avoid blocking (yourself and the others) with PRs
- do test each PR manually <- try not to make it boring as hell
- do keep CI log concise but error messages **explanatory**
- don't ever commit broken code <- hard to **trace** unpredictable changes
- don't spam people <- communication channels
- don't stop **mastering** your tools: git, text editor, browser, issue tracker, chat etc.

You Learn About It

- do **fight** the fatigue, don't get lost in apathy
- don't ignore **important** surveys (e.g. http://stateofjs.com)
- shape the **future** (ES7, API groups, W3C etc.)

stand out and shout! \../