Keep everything documented and public

by Jacek Tomaszewski
Fullstack Web Developer at Appunite.com
Freelancer and Consultant at jtom.me

Problem

- How do you agree with the product owner on **what should be done**?
  - Private/Public conversations
  - Designs, mockups, slides, emails
  - “Beta testing -> Feedback -> Fixing stuff” loop
  - You don’t (and deliver whatever feels right)

Problem

- How do you agree with UX designers on **how should it work like**?
  - Private conversations
  - Guessing from designs
  - Developer's own intuition
  - "QA -> Developers -> Designers -> Developers -> QA" loop

- How do you agree on **what are the expected end user scenarios?**
  - Early conversations about the project with the product owner
  - Guessing from designs
  - By testing the app
  - By the real users feedback (and their complaints/bug reports)

Problem

- How do you agree with other developers on **how should it be done?**
  - Private conversations
  - Code review (after it is already done (!))
  - JIRA/Asana task DoD/comments
  - Styleguide
  - You don’t (just let it merge)

- When project gets larger, **how do you onboard new teammates?**
  - Private conversations
  - You don’t (let them onboard by themselves)

Problem

- How do you explain (to the team / product owner) **how does X work?** (& why?)
  - Private conversations
  - By referring to the code / the app (“see the code”, “see the app”)
  - By referring conversations/tickets/emails/slides/designs/docs (often outdated)

Problem

- How do you decide on anything **asynchronously** and/or **in a team**?
  - ~~Emails~~
  - ~~Calls~~
  - ~~Private conversations~~
  - Public chats (i.e. Slack)
  - Code reviews
  - “Implement -> Deploy -> Test -> Feedback” loop
  - You don’t - you do it synchronously and/or skip a part of the team that is not currently present

Problem

Emails: async, public, but *bulky*

Private calls/chats: async, not bulky, but *private*

Public calls: *sync, time consuming, hard to do in a bigger group*

Public chats: async, flexible, but *disappear in vain over time*

Designs: they are just designs, they *don't contain business logic knowledge*

Code review, beta testing: it's *too late*

Finding information is a manual process

Case: Person asks a question about something.
What does the team needs to do to give them the answer?

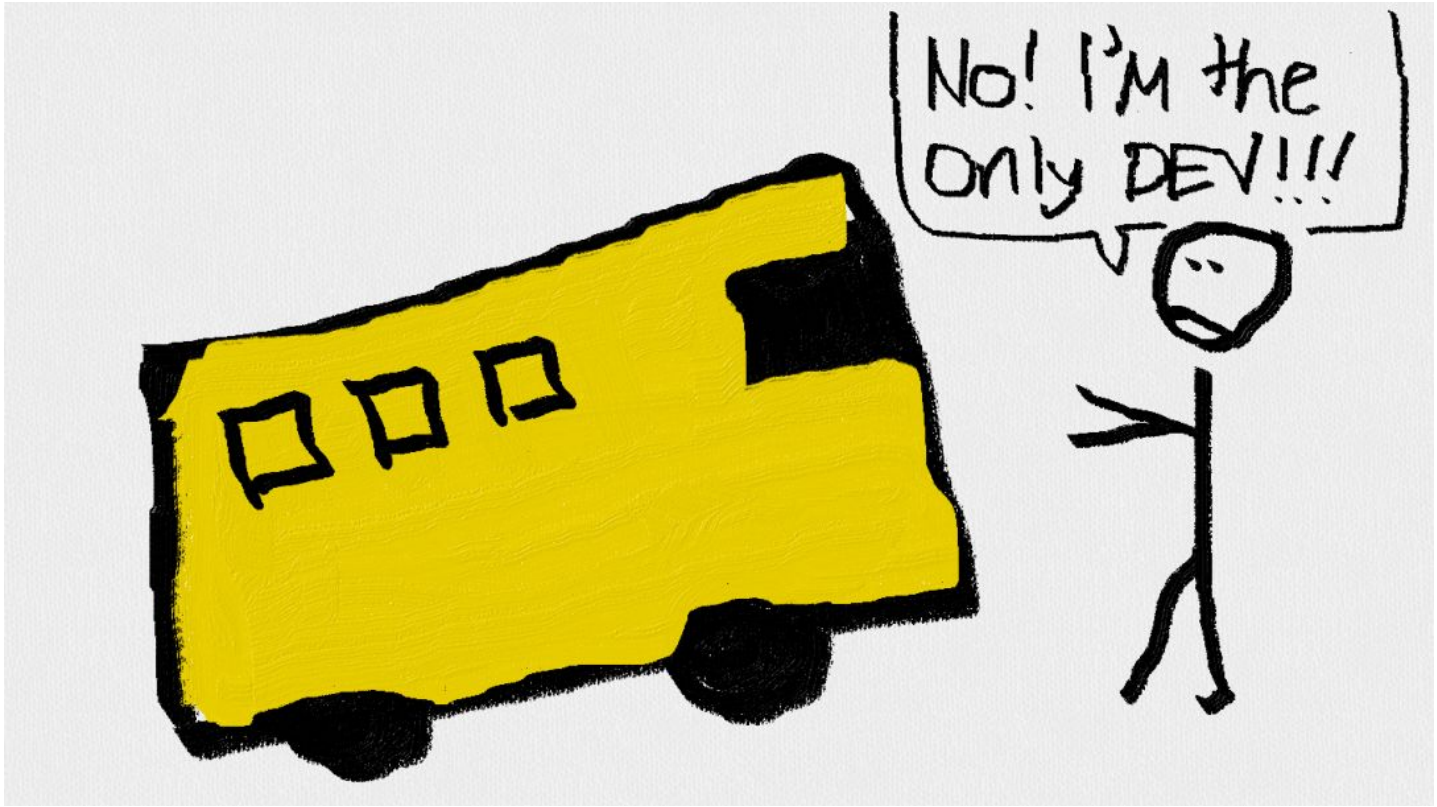
Emails: browse through old emails, find one, forward and comment it
Private calls/chats: find the people from the chat, then force them to repeat it
Public calls: same as above
Public chats: search through all the chats (or ask somebody for help)
Designs: ask the team to explain them
Code review: browse through past commits/PRs (or ask devs for explanation)
QA testing + feedback + hotfixing phase: browse through project tasks history (or ask the team to explain the changes)

Most of those solutions don’t scale

We do CI/CD to scale and automate the development.

Why don’t we do that with knowledge transfer?
Anything that needs manual work, will never scale!

+ remember about the bus factor



Public documentation as the solution

- Async
- Public
- Never disappears
- Not bulky (easy to read through ,to be commented and updated)
- Notifies about any changes
- Not noisy (skip observing docs that don’t interest you)
- Flexible (have docs in any format, for anything, for only the stuff you need)

Docs or no docs?

Cons of docs

- They can lie to you
  - Duplicated source of truth
  - Outdatedness

- Writing them might be time consuming

- Reading them might be time consuming

How to move to public docs* culture?

* without them being time consuming nor outdated

Rules

1. Everything must be public
2. Everything should be documented
3. Nothing can be outdated
4. Nothing can be duplicated
5. Define, Decide, Document, Implement
6. Mix your tools, but always document

Everything must be public

- **Avoid private chats**

- **Notify the interested**
  - about new discussions
  - about any decisions
  - about breaking changes
  - about your IT-life changing moments

- **Leave space and time for feedback**

Everything should be documented

- Anything that needs to be documented, **document it ASAP**

- **You can still define things through chat** - but remember:
  - **Eventually they need to be documented**
  - As soon as you reach a consensus, move it to docs

Nothing can be outdated

If there is a doc for something that will soon be changed:

1. Add a comment/note “this will soon be changed (to X by Y because Z)”
2. Later, resolve the comment and update the docs
    a. To reflect the current state
    b. Or, to say *“this is TBD”*
    c. Or, to say *“those docs are outdated, because Z. Read more: …”*)

Nothing can be outdated

- Docs can be up-to-date
- Docs can be TBD
- Docs can be deprecated
- **Docs should never be outdated**
    - (Unless you want them to lie to you!)

Nothing can be duplicated

- One source of truth:
  - Test cases tool for test cases
  - Design tool for designs
  - Code repo for code
  - Code repo for code styleguides
  - Code repo/Docs for abstraction explanations
  - Docs for terminology
  - Docs for explanation of how things work
  - Docs as a start of the net from which a link to all of the above can be found

Nothing can be duplicated

- If something is duplicated, remove the duplication
  - I.e. link from Jira to confluence
    - Instead of keeping docs in Jira, briefly mention only the DoD (“should work like x”), and for the explanation of X, link to confluence
  - Or, remove the outdated docs and add a comment “This has been changed, more details here: x, we should update those docs soon”
  - Or, remove the outdated docs and add a comment “More information about this can be found here: x”

Nothing can be duplicated

- Have one root knowledge base
  - *I.e. for us, everything starts with Confluence*
- Link to everywhere from the root
  - *Our base doc links to the task board (Jira), designs (Zeplin), test case scenarios (XRay), test URLs and credentials, API docs, Analytics & Accessibility requirements, etc.*
- Use other tools when they are better
  - *We use Dropbox Paper for fast collaboration on ideas [because Confluence is bad at it ;/]*
- Your docs can be just bunch of links
  - *Often in Confluence we just link to Slack/GH/JIRA/Dropbox Paper instead of copying stuff over*
- Keep other stuff where their place belongs
  - *In confluence we store only app, features, business logic, designs documentation*
  - *Code related decisions we keep in the code*
- Link back when you can
  - *I.e. link back from the code to the documentation it refers to*

Docs don't have to be just one place

- Keep stuff where their place belongs
  - In confluence we store only app, features, business logic, designs documentation
  - Code related decisions are kept in the code
  - Project setup is kept in the project repo
  - Project code styleguide is in the project repo
  - API schema docs are kept in the API docs
  - Implementation details for small/trivial tasks are described in JIRA (as DoD)
    - Only for major features/pages that require deep insight / discussion, their business logic description is defined in Confluence

Define, Decide, Document, Implement

1. **Define**
    a. Problem, context, solutions, their pros & cons
    b. Use a public doc, if it needs to be discussed/reviewed
2. **Decide**
    a. What and why? Next steps?
    b. Use a public doc, if it needs to be discussed/reviewed
3. **Document**
    a. Problem -> Possible solutions -> Decision -> Next steps
    b. If problem is major enough, add/link it in your doc root
    c. If decision impacts your existing documents, update them
4. **Implement**
    a. Link back
    b. Add “Why?” comments
    c. Extend README if needed

Mix your tools, but always document

- You can still use your existing communication tools (Slack, Hangout, etc.)
  - Use whatever feels the best to you

- **Always document afterwards**:
  - What has been discussed?
  - What has been decided?
  - What are the next steps?

Our recent lessons

Docs take time to master

- Writing nice docs is just like writing nice code
  - Biggest enemy in writing nice commit messages / PRs? Laziness
  - Biggest enemy in writing nice docs? Laziness

- Be flexible - think about who's the doc for!
  - Terminology doc: we removed business related terms, and added a lot of ones that are important from dev perspective
    - It saved our asses! And it appeared that the product owner didn't have a lot of stuff defined on his own

Be flexible

- Think about who's the doc for

- Our example: Terminology doc
  - We removed business related terms, and added a lot of ones that are important from dev perspective
  - It cleared up all the confusions
  - It showed that even the product owner doesn't have everything well thought over
  - It showed that the product owner, designers and PMs know stuff, but not exactly... and they all name it differently

Docs help us in writing better code

- **Promotes “Interface -> Implementation” workflow**

- **Helps the dev focus** on the essential
  - Good DoD lessens the chance of the dev overcomplicating stuff

- **Makes the implementation easy** of any given feature
  - If feature is not easy to understand: document it
  - If implementation details are not easy to guess: document them
  - Once implementation details are documented, coding it is a piece of cake

Docs help us in writing better code

- **Gives more time for better code** (by saving time wasted on unneeded code)

- **Good for juniors/mids**
  - through async collaboration on implementation docs, they can quickly learn how things should be designed

- **Good for software architects**
  - It lets anybody question architect's any assumption with "Why?", thus forcing him/her:
    - to have an objective reason for every decision
    - to learn how to explain anything to anybody

Thanks!

Questions, opinions?