Purposes: Some notes for coming up with a list of things to do for organizing a "proper" SE process for a typical modern software project. The purpose of these notes is both aspirational and organizational. We focus first more with the "what" we are trying to do as opposed to suffering the "how" too much. In other words, some of these points could be accomplished in a number of ways by a software team using a number of tools or process models.

Process: Team

- Team Formation
  - Introductions
  - Values Definition
  - Identification

- Sprint Process
  - Daily Stand-ups (style and capture)
  - Meetings (style and capture)
  - Ad hoc communications (style and capture)

- Team "Maintenance"
  - Onboarding
  - Keeping Cohesion Up
  - Evaluating Progress
  - Handling Conflict
  - Offboarding

- Team Disbanding

Process: Design, Plan, Execute

We should first aim to understand the problem before we try to solve it.

School PAs define the problem as a code puzzle to solve, most SE projects will be more ambiguous so we must employ a design process to address things

Process: Problem Definition and Scoping

We need to spend time before we get going too far along to figure out what exactly we are doing and how (roughly) we are going to go about it.

Brainstorming Process
- Double Diamond Methodology

- Tools: Miro, Zoom, Draw.io

Artifacts

- Architectural Decision Records (ADRs) in Markdown (Example)
- Vision / Overview Diagram or C4 Style System Context Diagram

Design Should Employ User Centered Thinking

We build software for people so we must try to think about people at all times instead of just features (Think of users using our features to solve their problems)

Artifacts

- Personas
- User Stories (initial ones may eventually be translated into tasks in our backlog)

Planning

As you get more specific you need to put your plans into specifications. There should be different degrees of fidelity starting first very rough and getting more precise and defined as the clarity of the project emerges.

Rough Before Finished

- Fat Marker Sketches
- Wireframes
- Flow diagrams (cross over with Event Modelling)

Detailed Design before Coding

- High Fidelity Prototype (for user test and to employ front-end first given high degree of project risk from user perception)

Execution: Tech

Tooling

- Do we have a slate of clear tools or is it BYOT?
    - Do we have an environment setup guide?
        - How do I set up VSCode? Are there scripts to help me with this?
- Specifics
    - 110 – Most tools and tech specified a few to pick
    - 112 – Tech specified (to a point), pick everything else
    - Industry – Varies though it indicates the type of management they have and a bit about their engineering values

Coding

- Where and how many repositories will we have?

- How do we organize our repo(s) and how rigorous is it adhered to?
    - Do we have repository guidelines? (example)
    - How do I set up Git? (example)
    - Do we have how to contribute guides? (example, example)

- Do our main branches have protection rules in place to prevent accidents?

- Do we have a development policy?
    - Feature branches (vs) Forks?
    - Are they being squashed after they're merged?
- Do we have proper commit messages?
    - Commit style guide?
    - Commit linter?
- How frequently is code being committed?
    - How big is each commit (we ideally want < 100 LOC)

Reviews

- Are code reviews happening for all code going into the repo? If so when?
    - Do we have code review guidelines and policies?

Documenting

- Who writes the docs and when?
- Where do we keep our documentation?
    - In repo vs Outside of Repo
        - SSOT vs Best Tool for Job?
    - This course is in Repo Only
- Do we have architecture diagrams?
    - Form of diagrams
        - UML
        - C4
        - Event Modeling
        - Ad hoc "boxes and lines"
- Internal documentation
    - Form
        - Private Wiki
        - Private Web Site(s)
    - Covers
        - Technical decisions (ADRs)
        - Processes (How to push a build)
        - Logistics and Management (Stand-up Notes, Meeting Notes, etc.)
    - Answers

            - I am a new developer, how do I contribute? (Onboarding) (example)
            - I want to know why you made the X/Y/Z design decision? (example)
            - I want to know how to tour the codebase? (example)
            - How does the team operate?
                - Values and code of conduct
            - Process documentation (meetings, communication, etc.)

    - External documentation
        - Form
            - Public Facing Web Site
            - Public Repo Wiki or Readme
        - Purpose
            - Sales: I’m just browsing; is your product worth using? Convince me!
            - Support: Okay, I’m sold! How do I get this thing going? How do I do X?

Building

- Do we have a CI working pipeline? i.e. on every pull request, is the pipeline running?
- Does our pipeline have these bare minimum steps?
    - Linter / Style Guide enforcer
    - Test Coverage checker
    - Build or Bundle of code
    - Optimization of code or assets (image compression, minification, etc.)
    - Code Quality checker (eg. CodeClimate)
    - Unit Tests Harness
    - End-To-End / Integration test harness
    - Packaging and pushing to delivery locations
- Is there an easy way to fire off the pipeline/ is there a local version of the pipeline?

Testing
- Do we have a test plan or at least test goals/philosophy?
- What kind of tests do we have?
    - Unit Tests
    - Integration Tests
    - End to End Tests
    - User Acceptance Tests
- Who is writing the tests?
    - Are coders writing tests?

- Is the testing team writing tests independent of the coders as well?

- When do we write the tests?
    - Before (TDD) or After?
- How much testing do we have?
    - Are code Coverage metrics available?
- How often are tests run and to what level of detail?
    - Quick tests on dev, Major tests on build, Full tests nightly or release?
    - How do the tests relate to the operations?

Operating

- Do we have an operating plan or at least goals/philosophy?
    - SLA, checklisting, escalation, etc.
- How well do we monitor the ongoing health of our system?
- How well do we monitor the ongoing health of the team?
- How does these use of the system inform ongoing changes to the software?
- DevOps – Analytics, Error Trace, etc.
    - Maintenance Mode
    - Staging Servers
- Do we have a road map in place that is both internal and external (feature request, relationship to support, etc.)
- Are the business metrics and technical metrics properly aligned?
    - Warning: Business metrics generally win in a tie and can impact road map significantly. Lack of business considerations in a design phase will manifest themselves in significant adjustment during operation since ultimately that is the reason the software was built TL;DR – the stay in your lane aspect of some software devs will significantly show its downside once software is operational. We must consider systems throughout to guard against these risks.