Software Engineering Tips:

Analysis | Estimation | Troubleshooting | Testing

Anshuman Sanghvi,
Project Lead, Java.
STTL

Analysis

Tips on Analysis: Outcome

- List of the functionalities, exact requirements and boundaries of the task
- List of blockers you will face or places of RnD you require to do and their solutions
- The best way to approach the solution out of multiple options
- The design (overall architecture, component level, design pattern, API contract, DB schema etc.)
- All the sub tasks required to complete this task
- List of validations required
- Understanding of the components, classes, tables, columns, stored procedures, triggers that you have to work with
- The 5 critical test scenarios that will evaluate all the important facets of the task
- The total time estimate for the overall task

Tips on Analysis: How

- Go through all relevant available material: SRS, KT Videos, Ticket descriptions, reference software, reference code base, domain information.
- If the task is part of a bigger module, you need to have a high level understanding of the overall module as well.
- For every task type, technology, library, or piece of code that you have not already worked on before, dive deeper into its details.
- Analyze that you factor in any parent component's behaviour is not broken, any sibling components that can affect this component or vice versa are also factored in.
- Analyze non functional requirements: security, performance, reliability etc. which can require things like parallelism, transactions rollbacks etc.

Estimation

Tips on Estimates

- Clarify all requirements, asking questions if necessary to ensure that the correct scope of the task is understood and confirmed with your task assigner, so that there are no surprises later.
- Break down the task into subtasks that are easy to analyze, develop and estimate.
- Include time in your estimate for RnD, design, initial setup, testing, bug fixing, cross review/testing, and deployment by creating individual sub tasks for them.
- Ensure that the architecture/design is approved with task assigner before you begin coding as this will ensure that you are estimating based on the correct approach.
- Have clarity about the order in which your tasks should be completed.
- Provide the total task estimate by adding individual subtask estimates.
- Whenever available, use actual observed historical data for estimating task instead of guessing.
- Use this formula: (Optimistic Estimate + Pessimistic Estimate + 4*Realistic Estimate)/6
- Inform your task assigner if at half way of your estimate you have not completed at least 40% of your total subtasks or 40% of the overall task.

Troubleshooting

Tips on Troubleshooting

- Confirm whether the error message is a symptom or the actual illness.
  - E.g the application is not starting due to a database connection issue. But the issue is connection might be just a symptom, the actual cause might be that the database server is down.
- Collect all possible information about the bug e.g.:
  - is there an exception? What type of exception? Where does it bubble up from?
  - What are the values of the variables at the “crime scene”? Is it possible to log the statements and the line of code at which the bug occurs?
  - What are the specific steps to reproduce this issue or bug?
  - Did any unit or integration tests fail? Can you trace why?
- Once you have an idea for a root cause, make a quick dirty way to test your hypothesis if possible, instead of going through long cycles of restarting all components of your application for every iteration.

Tips on Troubleshooting

- Iteratively isolate the problem into smallest surface area possible.
  - That is, iteratively eliminate the surface area logically or through code.
  - e.g front-end or backend? Backend. Code or Database? -> Database. Design or Data? Design. And so on)
  - Is this bug still present if I roll back the latest relevant code change before this was working correctly?
- Understand the meaning / cause and effect of the bug
  - If there was an exception, what is the exception type's meaning?
  - If there was a specific error message, can I go online and check what it means? Is there information regarding whether this is a known issue?
- Often the cause of the bug is the mismatch between what the code intends to do and what it actually does. Having clarity of the code intention and reading the code helps identify this.
- Use java debugger to use features like adding breakpoints, stepwise debugging, getting frame information etc.

Testing

Tips on Testing

- Think deeply about how to test the code.
  - If there is threaded or parallel execution, how will you test it?
- Think deeply about what you are testing?
  - Are you testing the logic in your code, or are you testing what a well tested library that you use already does?
- Don’t over or under compensate your testing.
  - test the critical functionality, but nothing more or less.
- Test according to the requirement
  - Sometimes integration test or regression test for a task is more important than a unit (logical) test.
- Test from multiple perspectives
  - If your feature is going to be used by multiple users/roles/clients, don’t just test with a single use case.
- Higher the risks/unknowns/security concerns/criticality, more intensive the testing.
- Don’t just test your code for your task
  - Test for vulnerabilities of the dependencies you are using,
  - test for whether the threading is working as expected on all the environments you task will be deployed to
  - If you are using microservices, have you tested your task’s API call across multiple services?

But there is one more thing…

The entire SDLC presentation was background/context to show you the following slides:

What this actually means for you?

DOs:

- Analysis before starting development:
  - clarify requirements,
  - agree on expected outcome,
  - do RnD on technology or implementation,
  - decide libraries to be used,
  - decide design: architecture, code design patterns, db design (tables, indices, relationships, primary and foreign keys), api contract etc.
  - critical test scenarios

DONTs:

- Give either unrealistic or exaggerated time estimates based on pressure, lack of required analysis, fear of unknown, or creating comfort zone.

- Start coding before analyzing task and eliminating unknowns, thinking that a task is easy or that there is insufficient time to do analysis due to pressure.

- Waste time on seeking perfection e.g: Non critical validations, Low impact bugs or testing, Writing and rewriting code even after goal achieved etc.

What this actually means for you?

DOs:

- Consider a task done only when it meets the “definition of done” criteria:
  - Tested after deployment on staging/prod environment
  - Does not break existing system
  - Works with other components of the system
- Factor in analysis, design, testing and deployment in your task estimates. (and not just development)

DONTs:

- Expect to be taught a technology or project code base, beyond basics; self starting goes a long way.
- Become stuck. If after 2-3 attempts or 2-3 hrs elapsed you are still stuck, seek help, don’t wait till last minute.
- Hesitate to ask help thinking you will look bad if your blocker/unknown is something “simple” / ”easy”.

What this actually means for you?

DOs:

- Inform immediately and repeatedly when you are stuck or out of tasks
- Take complete ownership of your task.

DONTs:

- Ask for help before doing your own troubleshooting.
- Hesitate to call out concerns even if the client or senior has asked you to do something a specific way.
- Hesitate to share status of your task accurately and often, even if you are stuck, late etc.

What this actually means for you?

Your work and career progress as a developer is judged on:

- Making working with you fun, easy, smooth for your coworkers. Handling code is easy, handling people is an ART!
- The predictability in your work
    - by following the processes and guidelines
    - giving realistic time estimates and achieving them
- The complexity of the tasks you can handle
- The ownership you show in terms of your tasks and responsibilities
- Communication skills and habits
    - Proactively communicating any status updates, milestones or major changes in your work
    - Communicating your issues clearly, and EARLY
    - Communicating with all stakeholders when you are late or unavailable

What this actually means for you?

Your work and career progress as a developer is judged on:

- How independently you can handle tasks
    - Independently researching solutions
    - Independently troubleshooting when stuck
    - Independently coming up with designs and architecture solutions

- The trust you generate by
    - Not taking shortcuts or skips in any phase of your SDLC
    - completing your tasks accurately and within estimates
    - Being courageous and honest when you don’t know, are stuck, are late, have made mistakes

Thank You

Skype: live:.cid.925bae56fe89963e
Email: anshuman . sanghvi @ silvertouch . com