Classic Mistakes and Ethics

This post talks about the classic mistakes and ethics regarding software development

In Web Development

Four categories of Mistakes

- People-related
- Process-related
- Product-related
- Technology-related

The People Dimension

- Results of studies indicates a 10-to-1 difference in productivity among different developers
- "Peopleware" issues have more impact on software productivity than any other factor
- Studies indicate that effects of individual ability, individual motivation, team ability motivation dwarf other productivity factors.
- Training helps with individual ability.
- Incentives helps with individual motivation
- Team bonding helps with team ability
    - Team synergy is important to ensure that everyone is able to give their best
- Team goals helps with team motivation
    - If the team have a clear goal to work towards to, the team is able to work together to meet the common goal

Variations in Productivity

- Greater than 10-to-1 differences in productivity among individuals with different depth and breaths of experience
- 10-to-1 differences in productivity among individuals with the same levels of experience
- 5-to-1 differences in productivity among groups with different level of experience
- 2.5-to-1 differences in productivity among groups with similar levels of experience
- Question : X-to-1 differences in productivity between AI and human ?

This shows that in a team setting, ensuring that individuals is able to work well in the team is important to ensure the success of the team. Individuals added to a team should be beneficial or able to maintain the team synergy to promote team cooperation and success.

People-Related Mistakes

- **Undermined Motivation**
    - (we have to like our job to ensure that we do our best)
    - Have a positive mindset, treat each job as a learning experience
- **Uncontrolled problem employees**
    - Discuss during tutorial
- **Heroics**
    - Over-confidence shows our inexperience an individual is
- **Noisy crowded offices**
    - Noisy might causes employees to be irritated which affects their mood negatively
- **Lack of user input**
    - Lack of participation and feedbacks from user might cause the developers to be not motivated to continuously improve the project
- **Lack of stakeholder buy in**
    - Lack of participation and feedbacks from stakeholders might cause the developers to be not motivated to continuously improve / evolve the project

Lack of user inputs and lack of stakeholder buy in is detrimental to the agile software development process as there is no constant improving and evolving of the system.

- **Politics placed over substance (can be fatal)**
    - Power struggle in organization. Access to data and rights can be seen as ranks and power which mislead developers in the team to work for climbing the ladder instead of working on the project for the success
- **Weak personnel**
    - Weak personnel can be solved via training if and only if the personnel have the right attitude to seek for improvement
    - Most companies are willing to send employees for training as this is a form of investment.
- **Adding people to a late project**
    - Might seem like a good solution, however, the new member might take a while to adjust with the work culture of the team which might in turn slow down the development of the team.

    - New members have to get use to the tools, software and adjust to the software development process the team is using.
- **Unrealistic expectations**
    - Unrealistic of project scope and management
- **Friction between developers and customers**
    - This makes it challenging to have a proper communication between the customer and the developers to have a proper discussion / feedback during the development phase of the project.
- **Wishful thinking**
    - Unreasonable deadlines due to inexperience in project planning
    - Overly optimistic might be the caused during the initial phase
    - This in turn leads to unethical work life to meet deadlines.
- **Lack of effective project sponsorship**
    - Insufficient budget, software requires heavy budget.
    - This might lead to cutting corner during the development phase which can be bring about bigger problem in the future or the project might not work as intended.

In conclusion, for developers to be a good software engineer

- Adaptive
    - Team have to be adaptive to the project, environment and the teammates
- Resilience
    - Set-backs are bound during the development phase and developers must have be ready to tackle any problems ahead. "Down but not Out!"
- Motivation
    - Developers must be motivated to produce a good product
- Resolve
    - Developers must have the grit in them to product a good product.

The Process Dimension

- "Process" includes both management and technical methodologies
    - Companies that have focused on process have cut their time-to-market by about one half and reduce cost and defects by factors of 3 to 10

This shows that ensuring proper process is beneficial in terms of time-taken to finish the product, cost of developing the product and also the maintenance of the product after publishing

<u>Aspects</u>

- Rework avoidance
  - Orient your process so you avoid doing things twice
  - Quality Control
- Quality assurance
  - To ensure that product has acceptable level of quality
  - To detect errors at the stage they are least time consuming and costly to fix
    - Errors can be
      - Use-case planning
      - Ensuring that we understand the functional requirements
      - etc.
  - This can be avoided via regular testing and testing at every phase
- Development Fundamentals
  - Don't cut corner
- Risk management
  - Ensure proper planning to ensure that we take all cases into account
  - Have proper contingency planning
- Resource targeting
  - Plan early and sufficiently to ensure that all resources (database, cloud, software) are ready and available during the development phase
- Life cycle planning
  - Be realistic during the planning phase
- Customer orientation
  - Maintain good relationship with the customer

Process-Related Mistakes

- Overly optimistic schedules
  - A good practice is having justification for each schedule
- Insufficient risk management
- Contractor failure
  - Vet contractor works and products before signing contracts with them to ensure that their product is able to fulfil the project requirement
- Insufficient planning

- Wasted time during the fuzzy front end
    - Fuzzy front end should be the last focus, ensuring the development of the project takes priority.
- Inadequate design
- Abandonment of planning under pressure
    - Planning is important to ensure that the team is on the right track in terms of deadlines and project requirement
- Shortchanged upstream activities
    - If we miss some of the functional requirement of the customer(project planning phase), this will affect the product development (upstream stages)
- Shortchanged quality assurance
    - Do not take chances in quality assurance
    - Be thorough in testing!
- Insufficient management controls
    - Review the team performance after each project and reshuffle the team if needed to ensure good team synergy and collaboration between the team.
    - Premature or overly frequent convergence
- Ensure that each stages is done properly including testing. Individuals might cut-corners to meet deadline quicker or give the impression that they are have completed their task early
- Omitting necessary tasks from estimates
- Planning to catch up later
    - Snowballing not good
- Code-like hell programming
    - Invest in software tools to ensure we don't have to code too much to boost productivity

The Product Dimension

- If you can reduce a product's feature set, you can reduce the product's schedule
- 80/20 rule - 80% of the product takes only 20% of the time
- Product Size - the biggest single contributor to development schedule (effort required to build software increases disproportionately faster than the size of the software)
- Product characteristic - hard to reach goals regarding performance, memory use, etc. will take longer

Product-Related Mistakes

- Requirements gold-plating
    - Don't over do perfecting each small bit, completing the functional requirement takes priority.
- Feature creep
    - Control customer request. Plan holistically with each user feedback to determine which cycle should the request be met.
    - Evaluate user request.
    - Do not simply prioritize meeting customer request over software development.
- Developer gold-plating
    - Don't over-perfect product or design
- Push-me, pull-me negotiation
    - Developers being push around.
    - Have proper negotiation with the customer to ensure that we are able to meet their requirements with current capabilities
- Research-oriented development
    - Tend to have more leeway in terms of deadline given the discovery nature of the project.
    - Application-oriented software tends to have well-define functionality that is expected to be delivered.

The Technology Dimension

- Changing from less effective tools to more effective tools can be a fast way to improve development speed
- The change form low-level languages to high-level languages wash one of the most important changes in software development
- Choosing tools effectively and managing the risk involved are important.

Technology-Related Mistakes

- Silver-bullet syndrome
    - Technologies isn't a miracle, it doesn't solve all problem. Developers have to also play a part to ensure that the technology is used correctly and appropriately in meeting the need of the project.
    - Dumping of technologies into the system does not create a good product.
- Overestimated savings from new tools or methods
    - Saving is likely to be incremental, it is not from using new tools or method
- Switching tools in the middle of a project

    - Every tools has their own learning curve, time is spent each time when learning a new tool. Hence, productivity is not gain if tools are being switch every time in the middle of the project.
- Lack of automated source-code control
    - Learning how to use Git is good to ensure the control of the source code.