

Facts and Fallacies of Software Engineering - Conclusions

Group discussions

During the evening, participants met in different groups to discuss a number of issues often viewed as controversial in the software development arena. There were three 30 minutes sessions. During each session three different questions were discussed, each in a different place in the room. The delegates in the room were free to move to whichever discussion they liked.

There was not enough time to have a thorough discussion on each topic. Groups listed comments in support of the statement, against the statement, or as relevant or interesting facts. Groups were asked not to try to resolve the issues but to understand the underlying or related factors.

Structure

Issues were grouped into three major groupings of issues:

- Session 1 – Management
- Session 2 – Requirements & Estimation
- Session 3 – Coding & Testing

Management

The three topics covered in this session were:

1. “The most important factor in software work is the quality of the programmers.”
2. “You can manage quality into a software product.”
3. “Hype (about tools and techniques) is the plague on the house of software.”

The most important factor in software work is the quality of the programmers

1. As the bulk of a software deliverable is created by programmers, this is true.
2. Management does have a strong influence.
3. For management to add value, they should have software experience.
4. For management to effectively manage a project and team and command respect, they should understand software and programmers.
5. Quality programmers will attract other quality programmers.
6. Quality management will attract quality programmers.
7. Quality programmers are not the most important aspect.
8. A lot depends on how quality is measured and which standards are set.
9. Quality programmers don't always have team skills.
10. Teamwork is more important than single programmers.

You can manage quality into a software product

1. Quality is a key objective of many management processes.
2. There are often conflicting interests between quality and other managerial objectives, such as cost or timely delivery.
3. Good management can cultivate a culture that can have a positive contribution.

4. Good management with bad programmers will fail.
5. Bad management with good programmes has a chance of success, but questions surrounding sustainability emerge.
6. Management provides resources and can therefore prohibit or encourage quality,
7. Management helps define quality in terms of business objectives and should ensure that a definition of “good enough” is shared by team members..
8. If management does not appreciate the technical constraints it can doom a project.
9. Developers must understand the business goals.
10. Management is the gatekeeper and is responsible for ensuring selecting the correct projects.
11. Management should establish an environment that allows quality to flourish.
12. Bad management can erode team relationships.

Hype (about tools and techniques) is the plague on the house of software.

During this discussion, points were raised about specific bad features of particular tools rather than the issue of hype as such.

1. UML
 - Specification
 - Prototype
 - Design
2. For 10 years there has been the greatest hype about Model Driven Architecture (MDA). This led to large amounts of money being spent, with essentially no reward.
3. Most tool hype is vendor driven, sometimes resulting in the real value of the tool being lost in the hype.
4. Some companies have created their own in-house tools.
5. Repository.
6. Certain tools, especially vapourware, are often sold to management. Implementation issues and benefits of use are discovered after the fact and left for the programmers to sort out.
7. The Business case for new tools is often not clear.
8. There are situations where the right tool can bring a huge increase in productivity. In Europe and the USA, organisations have the money to buy the tools but South African companies tend to throw people at problems.
9. There is need for background information when selecting tools.

Discussion – Requirements & Estimation

The three topics covered in this session were:

1. “Software estimation usually occurs at the wrong time.”
2. “Explicit requirements “explode” as implicit (design) requirements for a solution evolve.”
3. “There is a disconnect between software management and their programmers.”

Software estimation usually occurs at the wrong time.

1. More important to sell software than to develop it.
2. RFP process forces this.
3. Cover yourself by the right kind of contract.
4. 100% of requirements change!
5. Estimation is an iterative process.
6. Involve technical people at start of project.
7. There is a disconnect between software processes and customers.

Explicit requirements “explode” as implicit (design) requirements for a solution evolve

During this discussion it became clear that healthy communication was a pre-requisite for effective requirements analysis. This was a common thread through most of the discussions.

1. Developers want answers.
2. Prototypes can clear up fuzzy requirements as long as the feedback loop is closed.
3. There is a need for honesty about requirements, about what you can and cannot handle.
4. Check non-functional requirements.
5. Users and developers differ and don't put effort into resolving the differences.
6. Co-operation with users.
7. Middle management has an important role.
8. Knowledge of how to elicit requirements is important.
9. Don't understand problem – assumptions.

Discussion – Coding & Testing

The three topics covered in this session were:

“Programming can and should be egoless.”

“Errors tend to cluster.”

“Software developers talk a lot about tools, but seldom use them.”

Programming can and should be egoless.

1. No ego, no pride.
2. No ego leads to apathy, which stifles innovation.
3. Isn't passion good enough?
4. It's a trade-off between too much and too little ego - there is a sweet spot somewhere.
5. Ego can negatively impact relationships.
6. Ego should ideally be filtered and managed by countering it with confidence (on the side of the parties with less ego).
7. Team should have ego as a whole, but not individually.
8. Passion considers the environment, ego does not.

Errors tend to cluster

1. Experience bears this out.
2. A side effect of modular development – we try to abstract everything.
3. Uncertain about whether errors are due to the programmer or to fundamental complexity. It is not necessarily due to programmer and errors can often be blamed on inadequate time, lack of skills or the problem not being understood well enough.
4. Can trace to programmer – why? Psychological side – create mental blocks – bad code begets more bad code.
5. Leaky abstractions – abstracting at wrong level, e.g. TCP/IP.
6. Is this really true?

Software developers talk a lot about tools but seldom use them

1. Is management at fault? They are responsible for establishing the environment.
2. Tools are sometimes brought in without plan.
3. 5% of organisation should focus on methods and tools.
4. Have a plan.
5. Good champions (also objective)
6. If the value that a tool provides is not understood, it has little chance of being used.
7. Certain tools take a long time to repay initial investment and requires short-term benefits to be traded for long-term ones.
8. Open source is losing as tool due to over complexity and over design.
9. Tools are often sold to management, without input from programmers.
10. Change process.

There is a disconnect between software Management and their programmers.

1. Responsibility for strategic objectives are often (implicitly) pushed onto programmers, who do not have the authority or experience to make the decisions required of them.

2. Programmers and managers speak different languages.
3. “Ex-techie” managers are a good idea as they often get respect from programmers.
4. Separate technical and Project Manager responsibilities work well when the managers and programmers are gelled.
5. There must be mutual understanding between programmers and managers.
6. Some Project Managers are unwilling to delegate if they’re carrying the responsibility.
7. Certain cultures “shoot the messenger” when delivering bad news relating to technical developments.
8. Hiding complexity at the right level is challenging – this is exacerbated by a general lack of trust between programmers and managers.
9. Good techies often lack adequate people skills to manage. Technical prowess is not always congruent with good leadership.
10. There is often an adversarial relationship and mutual disinterest between programmers and managers. People, both managers and programmers should “get over themselves”.

How the session could be improved

At the end of the session the participants were asked how they thought the session could be improved. They raised the following points:

- Get scribes and facilitators.
- Get slide printouts.
- Short time to discuss complex topics – send them out to prepare.
- More time on fewer topics.
- Divide groups more evenly.
- Let’s capture and put this on website.

Concluding comments

Most participants felt it was an evening well spent, although they would like to have spent more time on some of the subjects. The comment was raised that we should do the process with some of the *really* contentious issues in the industry.

The topics as they have been discussed and documented could provide a useful basis for online discussion in an appropriate forum.