

# Too Big To Trash(TBTT)

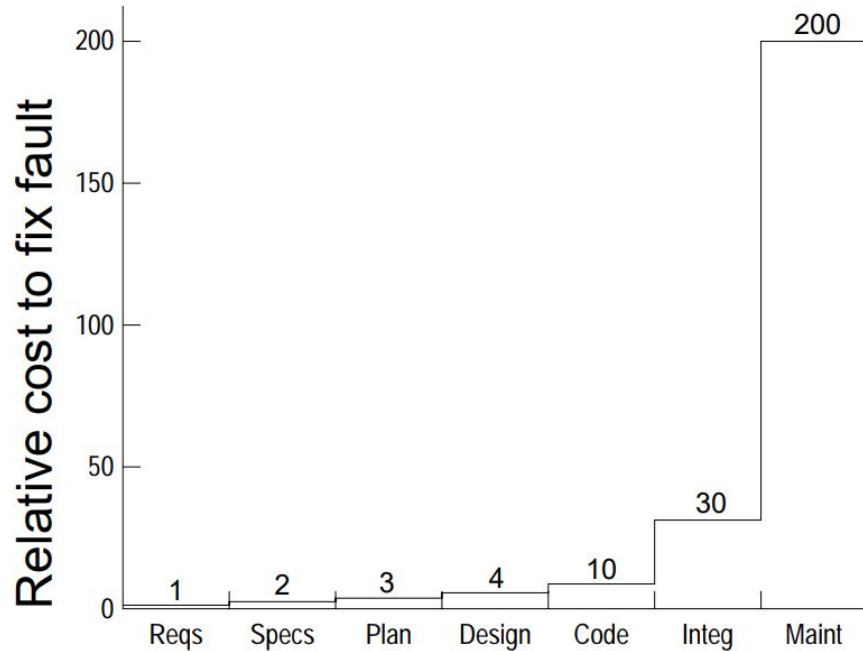
Requirements Engineering of Perfective Maintenance

# Outline

1. Recap from Dan's previous lectures
2. Extrapolating the recap
3. Research strategy
4. Open source's strategy
5. Agile
6. Is all perfective maintenance agile?
7. Tips I found for perfective maintenance
8. Closing thought



# Cost of Requirements Changes per Stage



Phase in which fault is detected and fixed

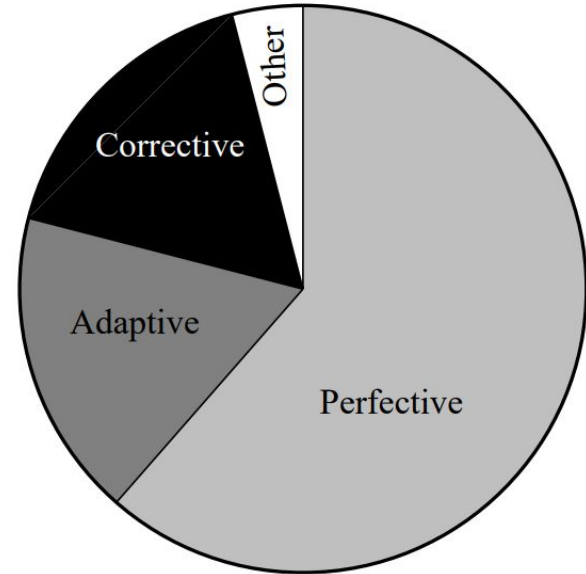
# E-Type Software

- Software that once it is deployed, changes the requirements of the system.
- Usually the systems allows something to happen that couldn't happen previously.
- Once the new behaviour exhibits itself, it needs to be addressed in new ways there were not considered beforehand.
- Ex. A bank that deploys ATMs as an optional feature, allows increased customer base, no longer optional, additional supports needed to provide redundancy.



# Types of Maintenance

- Corrective maintenance is traditional bugs.
- Perfective and adaptive maintenance is either e-type software or desired requirements that were not thought of ahead of time.



# What does this all mean?

- Less and less large software seems to be built in a way that would facilitate full scale upfront RE.
- But upfront RE has numerous benefits, as shown by previous lectures.
- When a CBS gets to the maintenance stage, the most expensive stage, a lot of RE might still left to do for the perfective and adaptive maintenance.
- If the size of the software is large enough, throwing it out could be an impossibility (maybe not logically, but politically/realistically)
- What is the right way to accomplish this?



# Research Strategy

- Looking at Google Scholar, IEEE Xplore, ACM DL, “perfective maintenance requirements” or returned 0 seemingly relevant results
- During project proposal discussions, Dan was nice enough to give me a few seed papers and online books that he had found.
- Followed a loose mental snowball of each paper suggesting new topics or related research



# Open Source Software Development (OSSD)

- Open source software (OSS) seems to manage this on the regular (Mozilla, Linux, etc.)
- Lots of OSS seems to exist for a long time adding features, Linux was originally released in 1991.
- With OSS, almost anybody can submit a change request, and almost anybody can make and submit modifications.
- Where do they get their requirements?





# OSSD Requirements

- Many OSS developers are also OSS users.
- A common reason to contribute is they will use the feature they add.
- OSS developers working on a feature work very closely with an OSS user who is highly available to ensure that the feature matches the requirements.
- A change being reviewed by other OSS developers, is also reviewed by OSS users to ensure the feature matches the requirements.
- Sounds a lot like Agile.



# Agile Software Development (Agile)

- At a high level, Agile involves:
  - Keeping software runnable
  - Working closely with customers to define requirements when they are needed
  - Accepting and working with changes as they come
  - Valuing software over documentation



# Two Common Agile RE Techniques

Test Cases as Requirements(TCR)

And

Just In Time Requirements Determination



# Test Cases as Requirements

- If a test case is testing the required behaviour, as long as the test case is understandable, it can serve a similar purpose.
- Can involve recording informal or formal requirements and translating to test cases or bypassing separately recorded requirements entirely.
- Adding a new feature to a software in maintenance would involve just creating the new test cases.
- All other test cases should be able to maintain satisfaction of all other requirements.



# Just In Time(JIT) Requirements Determination

- JIT involves maintaining requirements in high level simple formats until implementation is about to begin.
- Requirements are elaborated in conjunction with implementation beginning.
- If an entire system is built this way, all issues will be built on top of other requirements with finished implementations.
- This means that the development process is maintained the same through maintenance and should cause no undue issues than the initial development.



# How Does This Relate to Perfective Maintenance

- A few of the principles of the Agile Manifesto relate to always having working software.
- Dan suggested while discussing my project proposal that if you always have working software, every iteration past the first is perfective maintenance




# How Does This Relate to Perfective Maintenance

- It seems based on my research that all perfective maintenance is loosely agile until you throw everything away and start over or just continue being agile forever.
- It all involves working with users to some degree to add the features they want into an existing codebase. When the feature is done you release out to the masses.
- Beatty and Weigers suggest adopting agile practices when taking on an enhancement project regardless of how the original project was built.




# Tips for Perfective Maintenance

- If requirements documentation is missing for what you are perfecting, create as much documentation as makes sense for your change and how it interfaces with the existing software.
  - Pay attention to the costs and benefits of documentation to determine whether poorly documented existing work should be documented.
  - Try to keep track of which requirements are no longer necessary and drop them to reduce bloat.
  - Practice politics like Dan described in previous lecture with your userbase to make any changes as easy as possible.
- 



# A Further Question I Had

- Dan suggests that upfront RE is best in his experience and to throw out written software when the cost of updating it is too high.
  - Large software makes that difficult because each individual change is so small that it doesn't cost enough to throw out.
  - Could large software be written in a way that was modular enough and with small enough modules that you could throw away just a module and start over with upfront RE?
  - Seems like a large discipline crossover between Software Architecture and Requirements Engineering.
- 

# References

Dan Berry, "The Requirements Iceberg" <https://cs.uwaterloo.ca/~dberry/ATRE/Slides/IcebergSlides.pdf>

Karl Wiegers and Joy Beatty, "Developing Requirements for Enhancement and Replacement Projects", 8/15/2013

Dietze, Stefan. (2005). Agile requirements definition for software improvement and maintenance in open source software development.

<https://www.extremetech.com/computing/94026-linux-is-20-years-old-today>

Eva-Maria Schön, Jörg Thomaschewski, María José Escalona, "Agile Requirements Engineering: A systematic literature review," *Computer Standards & Interfaces*, Volume 49, 2017, Pages 79-91

T. A. Alspaugh and W. Scacchi, "Ongoing software development without classical requirements," *2013 21st IEEE International Requirements Engineering Conference (RE)*, Rio de Janeiro, 2013, pp. 165-174.

N. A. Ernst and G. C. Murphy, "Case studies in just-in-time requirements analysis," *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*, Chicago, IL, 2012, pp. 25-32.

A. Q. Do and T. Bhowmik, "Refinement and Resolution of Just-in-Time Requirements in Open Source Software: A Case Study," *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Lisbon, 2017, pp. 407-410.

