Methodology for Assessing the State of the Practice for Domain X

Spencer Smith
McMaster University, Canada
smiths@mcmaster.ca

Jacques Carette
McMaster University, Canada
carette@mcmaster.ca

Olu Owojaiye
McMaster University, Canada
owojaiyo@mcmaster.ca

Peter Michalski
McMaster University, Canada
michap@mcmaster.ca

Ao Dong
McMaster University, Canada
donga9@mcmaster.ca

Abstract

...

2012 ACM Subject Classification Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases Author: Please fill in \keywords macro

Contents

1 Introduction

Purpose and scope of the document. [Needs to be filled in. Should reference the overall research proposal, and the "state of the practice" exercise in particular. —SS]

2 Overview of Steps in Assessing Quality of the Domain Software

1. Identify domain. (Provide criteria on a candidate domain.)
2.

3 Identify Candidate Software

1. Must be open source.
2. Must have GitHub repository.

4 Domain Analysis

Commonality analysis. Follow as for mesh generator (likely with less detail).

Commonality analysis document Steps:

1. Introduction
2. Overview of Domain
3. Add Commonalities - Split into simulation, input, output, and nonfunctional requirements
4. Add Variabilites - Split into simulation, input, output, system constraints, and nonfunctional requirements
5. Add Parameters of Variation - Split into simulation, input, output, system constraints, and nonfunctional requirements
6. Add Terminology, Definitions, Acronyms

Commonality analysis for Lattice Boltzmann Solvers can be found here.

5 Empirical Measures

5.1 Raw Data

Measures that can be extracted from on-line repos.

[Still at brainstorm stage. —AD]

- number of contributors
- number of watches
- number of stars
- number of forks
- number of clones
- number of commits
- number of total/code/document files
- lines of total/logical/comment code
- lines/pages of documents (can pdf be extracted?)
- number of total/open/closed/merged pull requests
- number of total/open/closed issues
- number of total/open/closed issues with assignees

Instead of only focus on the current status of the above numbers, we may find the time history of them to be more valuable. For example, the number of contributors over time, the number of lines of code over time, the number of open issues over time, etc.

5.2 Processed Data

Metrics that can be calculated from the raw data.

[Still at brainstorm stage. —AD]

- percentage of total/open/closed issues with assignees - Visibility/Transparency
- lines of new code produced per person-month - Productivity
- lines/pages of new documents produced per person-month - Productivity

- number of issues closed per person-month - Productivity
- percentage of comment lines in the code - maintainability [Not Ao's qualities —AD]

In the above calculations, a month can be determined to be 30 days.

5.3 Tool Tests

[This section is currently a note of unorganized contents. Most parts will beremoved or relocated. —AD]

[This citation needs to be deleted later. It's here because my compiler doesn't work with 0 citations —AD] Emms [2019]

Most tests were done targeting to the repo of 3D Slicer GitHub repo

5.3.1 git-stats

GitHub repo

Test results: http://git-stats-slicer.ao9.io/ the results are output as webpages, so I hosted for you to check. Data can be downloaded as spreadsheets.

5.3.2 git-of-theseus

GitHub repo

Test results: It took about 100 minutes for one repo on a 8 core 16G ram Linux machine. It only outputs graphs.

5.3.3 hercules

GitHub repo

Test results: this one seems to be promising, but the installation is complicated with various errors.

5.3.4 git-repo-analysis

GitHub repo

5.3.5 HubListener

GitHub repo

The data that HubListener can extract.

Raw:

- Number of Files
- Number of Lines
- Number of Logical Lines
- Number of Comments

Cyclomatic: Intro

- Cyclomatic Complexity

Halstead: Intro

- Halstead Effort
- Halstead Bugs
- Halstead Length

- Halstead Difficulty
- Halstead Time
- Halstead Vocabulary
- Halstead Volume

Test results: HubListener works well on the repo of itself, but it did not work well on some other repos.

5.3.6 gitinspector

GitHub repo

Test results: it doesn't work well. Instead of creating output results, it prints the results directly in the console.

6 User Experiments

6.1 Usability Experiment

Steps:

1. Write research questions
2. Prepare logistic details(location, time table, recording setup, moderators etc)
3. Prepare survey questions
4. Select a list of projects
5. Design the tasks for the study subjects to perform (tasks can be defined based on user category, at least one task to modify the software)
6. Select participants
7. Conduct usability session
8. Survey the study subject to collect pre-experiment data
9. Perform tasks
10. Observe the study subjects (take notes, record sessions, watch out for body language and verbal cues)
11. Survey the study subjects to collect final feedback
12. Prepare experiment report
13. Perform pairwise comparison analysis
14. Prepare analysis report

6.2 Empirical Measures

1. Qualitative/Quantitative
2. Pairwise Comparison methodology needs to be described
3. Measure time to complete a task and compare actual with expected time of completion
4. From a list of given tasks, measure how many tasks were completed correctly
5. Learnability
6. Efficiency
7. Memorability
8. Errors
9. Satisfaction
10. Heuristic evaluation: Experts go through the interface to identify elements that violate usability heuristics.Popular because of its low cost and low time commitment

6.3 Nielsen's Heuristics

1. Consistency: Check if standards and conventions in product design are followed
   - Sequences of actions (skill acquisition).
   - Color (categorization).
   - Layout and position (spatial consistency).
   - Font, capitalization (levels of organization).
   - Terminology (delete, del, remove, rm) and language (words, phrases).
   - Standards (e.g., blue underlined text for unvisited hyper links).
2. Visibility: Users should be informed about what is going on with the system through appropriate feedback and display of information e.g.
   - What is the current state of the system?
   - What can be done at current state?
   - Where can users go?
   - What change is made after an action?
3. Match between system and world. The image of the system perceived by users should match the model the users have about the system.
   - User model matches system image.
   - Actions provided by the system should match actions performed by users.
   - Objects on the system should match objects of the task.
4. Minimalist: Any extraneous information is a distraction and a slow-down.
   - Less is more.
   - Simple is not equivalent to abstract and general.
   - Simple is efficient.
   - Progressive levels of detail.
5. Minimize memory load. Users should not be required to memorize a lot of information to carry out tasks. Memory load reduces users' capacity to carry out the main tasks.
   - Recognition vs. recall (e.g., menu vs. commands).
   - Externalize information through visualization.
   - Perceptual procedures.
   - Hierarchical structure.
   - Default values.
   - Concrete examples (DD/MM/YY, e.g., 10/20/99).
   - Generic rules and actions (e.g., drag objects).
   - Intuitive procedure
6. Informative feedback. Users should be given prompt and informative feedback about their actions.
   - Information that can be directly perceived, interpreted, and evaluated.
   - Levels of feedback (novice and expert).
   - Concrete and specific, not abstract and general.
   - Response time:
     -0.1 s for instantaneously reacting;
     -1.0 s for uninterrupted flow of thought;
     -10 s for the limit of attention.
7. Flexibility and efficiency. Users always learn and users are always different. Give users the flexibility of creating customization and shortcuts to accelerate their performance.
   - Shortcuts for experienced users.
   - Shortcuts or macros for frequently used operations.
   - Skill acquisition through chunking.

- Examples: - Abbreviations, function keys, hot keys, command keys, macros, aliases, templates, type-ahead, bookmarks, hot links, history, default values, etc.

8. Good error messages.
   - The messages should be informative enough such that users can understand the nature of errors, learn from errors, and recover from errors.
   - Phrased in clear language, avoid obscure codes. Example of obscure code: "system crashed, error code 147."
   - Precise, not vague or general. Example of general comment: "Cannot open document."
   - Constructive.
   - Polite. Examples of impolite message: "illegal user action," "job aborted," "system was crashed," "fatal error," etc.
9. Prevent errors. It is always better to design interfaces that prevent errors from happening in the first place.
   - Interfaces that make errors impossible.
   - Avoid modes (e.g., vi, text wrap). Or use informative feedback, e.g., different sounds.
   - Execution error vs. evaluation error.
   - Various types of slips and mistakes.
10. Clear closure. Every task has a beginning and an end. Users should be clearly notified about the completion of a task.
    - Clear beginning, middle, and end.
    - Complete 7-stages of actions.
    - Clear feedback to indicate goals are achieved and current stacks of goals can be released. Examples of good closures include many dialogues.
11. Reversible actions.
    - Users should be allowed to recover from errors. Reversible actions also encourage exploratory learning.
    - At different levels: a single action, a subtask, or a complete task.
    - Multiple steps.
    - Encourage exploratory learning.
    - Prevent serious errors.
12. Use users' language. The language should be always presented in a form understandable by the intended users.
    - Use standard meanings of words.
    - Specialized language for specialized group.
    - User defined aliases.
    - Users' perspective. Example: "we have bought four tickets for you" (bad) vs. "you bought four tickets" (good).
13. Help and documentation. Always provide help when needed.
    - Context-sensitive help.
    - Four types of help.
      -task-oriented;
      -alphabetically ordered;
      -semantically organized;
      -search.
    - Help embedded in contents.
14. Users in control. Do not give users that impression that they are controlled by the systems.
    - Users are initiators of actors, not responders to actions.

- Avoid surprising actions, unexpected outcomes, tedious sequences of actions, etc.

6.4 Tools

1. Downloaded OBS to record experiment sessions

6.5 To do

1. Detail plan
2. Describe Pairwise methodology
3. Define tasks
4. prepare survey questions

6.6 Task definition/selection criteria

1. Tasks based on difficulty levels (Difficult, Medium and easy)
2. Commonly performed tasks (applicable to all the selected software)
3. Task that would/should generate errors messages or error codes
4. Tasks to check standards and conventions
5. Tasks that should provide users with the system's current state and what actions can be done or where to go or what change was made
6. Tasks related to what users will do in the real world
7. End to End tasks that require users to go through sequential or hierarchical steps.
8. Tasks that allow users to reuse a previously defined value without having to memorize or jot it.
9. Tasks that should provide users with feedback
10. Tasks that allow users to customize and use shortcuts, bookmarks, command keys etc.
11. Tasks to verify if the system is capable of preventing and /or reversing errors
12. Tasks that might lead the user to use help or documentation

6.7 Usability Questionnaire

This questions need to be reworded to make them as open ended as possible
source https://www.usabilitest.com/sus-pdf-generator- 20-29

1. Overall, I am satisfied with how easy it is to use this system.
2. It was simple to use this system.
3. I could effectively complete the tasks and scenarios using this system.
4. I was able to complete the tasks and scenarios quickly using this system.
5. I was able to efficiently complete the tasks and scenarios using this system.
6. I felt comfortable using this system.
7. It was easy to learn to use this system.
8. I believe I could become productive quickly using this system.
9. The system gave error messages that clearly told me how to fix problems.
10. Whenever I made a mistake using the system, I could recover easily and quickly.
11. The information (such as online help, on-screen messages, and other documentation) provided with this system was clear.
12. It was easy to find the information I needed.
13. The information provided for the system was easy to understand.
14. The information was effective in helping me complete the tasks and scenarios.
15. The organization of information on the system screens was clear.

16. The interface of this system was pleasant.

17. I liked using the interface of this system.

18. This system has all the functions and capabilities I expect it to have.

19. Overall, I am satisfied with this system.

20. I think that I would like to use this system frequently.

21. I found the system unnecessarily complex.

22. I thought the system was easy to use.

23. I think that I would need the support of a technical person to be able to use this system.

24. I found the various functions in this system were well integrated.

25. I thought there was too much inconsistency in this system.

26. I would imagine that most people would learn to use this system very quickly.

27. I found the system very cumbersome to use.

28. I felt very confident using the system.

29. I needed to learn a lot of things before I could get going with this system.

7 Analytic Hierarchy Process

Describe process. Domain expert review.

8 Quality Specific Measures

8.1 Installability [owner —OO]

8.2 Correctness [owner —OO]

8.3 Verifiability/Testability [owner —OO]

8.4 Validatability [owner —OO]

8.5 Reliability [owner —OO]

8.6 Robustness [owner —PM]

8.7 Performance [owner —PM]

8.8 Usability [owner —JC]

8.9 Maintainability [owner —PM]

8.10 Reusability [owner —PM]

8.11 Portability [owner —PM]

8.12 Understandability [owner —JC]

8.13 Interoperability [owner —AD]

8.14 Visibility/Transparency [owner —AD]

8.15 Reproducibility [owner —SS]

8.16 Productivity [owner —AD]

8.17 Sustainability [owner —SS]

8.18 Completeness [owner —AD]

8.19 Consistency [owner —AD]

8.20 Modifiability [owner —JC]

8.21 Traceability [owner —JC]

8.22 Unambiguity [owner —SS]

8.23 Verifiability [owner —SS]

8.24 Abstract [owner —SS]

9 Using Data to Rank Family Members

Describe AHP process (or similar).

A Appendix

A.1 Survey for the Selected Projects

[Several questions are borrowed from Jegatheesan2016, and needed to be cited later. —AD]

A.1.1 Information about the developers and users

1. Interviewees' current position/title? degrees?
2. Interviewees' contribution to/relationship with the software?
3. Length of time the interviewee has been involved with this software?
4. How large is the development group?
5. What is the typical background of a developer?
6. How large is the user group?
7. What is the typical background of a user?

A.1.2 Information about the software

1. [General —AD] What is the most important software quality(ies) to your work? (set of selected qualities plus "else")
2. [General —AD] Are there any examples where the documentation helped? If yes, how it helped. (yes*, no)
3. [General —AD] Is there any documentation you feel you should produce and do not? If yes, what is it and why? (yes*, no)
4. [Completeness —AD] Do you address any of your quality concerns using documentation? If yes, what are the qualities and the documents. (yes*, no)
5. [Visibility/Transparency —AD] Is there a certain type of development methodologies used during the development? ({Waterfall, Scrum, Kanban, else})
6. [Visibility/Transparency —AD] Is there a clearly defined development process? If yes, what is it. ({yes*, no})
7. [Visibility/Transparency —AD] Are there any project management tools used during the development? If yes, what are they. ({yes*, no})
8. [Visibility/Transparency —AD] Going forward, will your approach to documentation of requirements and design change? If not, why not. ({yes, no*})
9. [Correctness and Verifiability —AD] During the process of development, what tools or techniques are used to build confidence of correctness? (string)
10. [Correctness and Verifiability —AD] Do you use any tools to support testing? If yes, what are they. (e.g. unit testing tools, regression testing suites) ({yes*, no})
11. [Correctness and Verifiability —AD] Is there any document about the requirements specifications of the program? If yes, what is it. ({yes*, no})
12. [Portability —AD] Do you think that portability has been achieved? If yes, how? ({yes*, no})
13. [Maintainability —AD] How was maintainability considered in the design? (string)
14. [Maintainability —AD] What is the maintenance type? (set of {corrective, adaptive, perfective, unclear})
15. [Reusability —AD] How was reusability considered in the design? (string)
16. [Reusability —AD] Are any portions of the software used by another package? If yes, how they are used. (yes*, no)
17. [Reproducibility —AD] Is reproducibility important to you? (yes*, no)
18. [Reproducibility —AD] Do you use tools to help reproduce previous software results? If yes, what are they. (e.g. version control, configuration management) (yes*, no)
19. [Completeness —AD] Is any of the following documents used during the development? (yes*, no)

20. [General —AD] Will this experience influence how you develop software? Do you see yourself maintaining the same level of documentation, tool support as you go forward? (string)
    - Module Guide
    - Module Interface Specification
    - Verification and Validation Plan
    - Verification and Validation Report

References

Steve Emms. 16 best free linux medical imaging software. https://www.linuxlinks.com/medicalimaging/, 2019. [Online; accessed 02-February-2020].