Managing your software project

Killian Murphy <killian.murphy@york.ac.uk>

Background

- UNIX systems administration
- Legacy systems maintenance
- Full-stack development
- Business analysis
- Project management

Presentation structure

- Divided into three core aspects of software projects
- Each section will include an introduction, an informal approach, a formal approach, and attempt to evaluate some relevant technologies
- Happy to field questions at any point during the presentation
- Audience participation is particularly encouraged at the end of each section - I (and others) want to know how you manage your software projects!

Key facets of a software project

- Design:
  - Specification of the things required to achieve the intended project outcome
- Implementation:
  - Transformation of specification into code
- Maintainability:
  - Sustaining project health and promoting collaboration

Design

- Knowing where you're going and what the steps are that you need to take to get there
- Informal or formal, as appropriate
    - Pros and cons to both approaches, as you'd expect
    - Document either way!
- **Always** consider this aspect of your project
    - Can be invaluable to know why you chose to do something
    - Others may want/need rationale
    - Useful record for write-up

Informal approach to design

Formal approach to design

- Choose your approach to gathering requirements:
  - Enumeration
  - ‘Brainstorming’
  - Interviews
  - Inspecting other codebases
- Document your approach to gathering requirements
  - Project overview
  - Requirement analysis and validation techniques
  - Requirement prioritisation strategy
- Document important contacts
- Identify sources of requirements
- Derive a schedule

Formal approach to design

- Gather and document requirements from identified sources
- Analyse and validate your requirements
    - This can result in some iteration
- Prioritise your requirements
- Confirm requirements with collaborators

Formal approach to design

- Identify and document useful tools (libraries, methodologies, patterns, textbooks etc)
- Include rationale wherever appropriate
    - Even if you disagree with it later, it is likely to be useful to know why you chose to do something
- Consider your software architecture
    - Diagrams can be very useful here
    - Investigate architectures of similar projects, if they exist
    - Reflect on other projects that you have worked on

Supporting technologies

Google Drive

- Supports many types of document
- Straightforward to collaborate on
- Can be scripted (but you probably don’t want to)
- Painful to version control
- Requires Google account

Supporting technologies

Markdown

- Plain text!
- Straightforward to version control
- Convertible to many formats
- Simple syntax
- Learning curve
- No native support for diagrams
- Not WYSIWYG

Supporting technologies

Treesheets

- Supports all kinds of data organisation
- Very flexible
- Simple enough to learn
- Not straightforward to version control
- Cumbersome user experience

Design: audience experience

- How have you managed the design of your software projects?
- What has your experience been with your method of design?

Implementation

- Transformation of specification into code
- Very fun
  - Can be too much fun!
- Vast range of opinions on how implementation should be carried out
  - Often a very divisive subject
- Attention paid to project design pays off here

Informal approach to implementation

- Pay attention to any project specifications
    - Reviewing the spec > deviating from the spec during implementation
- **USE VERSION CONTROL**
    - Git is in widespread use, and works especially well in teams
    - Subversion and Mercurial have been historically popular, although appear to be in decline [1]
- Consider writing documentation as you write code
    - Either in-code or outside of code
    - Covered in ‘Maintainability’
- Build your implementation in small, testable chunks
    - Makes debugging (which you will be doing) more manageable
    - Can keep you motivated to know that things are on-track

Informal approach to implementation

- Consider inviting somebody to review your code
    - Things might make perfect sense to you, but not to others
- Don’t ‘reinvent the wheel’
    - Many problems have already been solved - use existing implementations (where appropriate)
- Work on your codebase regularly...
    - Can be difficult to ‘get back into’ a codebase after time away
- ...but don’t work on your codebase excessively
    - Match the implementation to your intended outcomes
    - Premature optimization is the root of all evil
    - Be wary of burning out

Formal approach to implementation

Test-driven development

- Very short development cycle
- Software grows with respect to new tests, which the software must pass
- Can foster very lean codebases
- Can foster ridiculous codebases
- Apply judiciously!

Formal approach to implementation

Test-driven development

- Add a test
  - Closely coupled with a project requirement, mandating a strong specification
- Validate the new test
- Write code to pass the test
  - The minimal solution to pass the test - elegance is not important yet
- Run tests
  - All tests must be passed - return to step 3 if there are failures
- Refactor codebase
  - Revise the code that simply passes the test - this forces the entire codebase to be regularly reviewed
  - All tests must still pass!

Supporting technologies

GNU Make

- Very flexible in its possible uses
- Well-documented
- Many examples available
- Syntax can be obtuse
- Doesn’t scale well
- Consider alternatives: Snakemake, Airflow etc

Supporting technologies

Vagrant

- Provides consistency throughout implementation
- Straightforward to configure
- Many ‘boxes’ available
- Can easily induce time wasting
- Update issues

VAGRANT

Supporting Technologies

GitHub

- Generally good user experience
- Straightforward to use
- Offers more than just repository hosting
- Free for academic use
- Can be overwhelming

Implementation: audience experience

- How do you go about implementation?
- Any favourite tools?

Maintainability

- Keeping the project healthy
- Promoting collaboration
- Simplifying debugging
- Supporting change
- **Future-proofing**

Informal approach to maintainability

- Coding style
  - Supported/community style guide
  - Consistency
  - Personal preference vs readability
- Documentation
  - ‘Self-documenting’ code
  - Documentation generation
  - Tutorials/examples

Informal approach to maintainability

- Consider issue tracking:
  - TODO
  - Separate document
  - Issue tracker
- Onboarding
  - How straightforward is it for somebody else to pick up your codebase?
  - Try it!
- **You never know when you might use your software again**

Informal approach to maintainability

- Well-considered build system can be key
  - Building on a range of systems
  - Building for a range of systems
- Factor maintainability into your project design, commit to maintainability during implementation
- Software Sustainability Institute provides useful reading material [2]

Formal approach to maintainability

Complexity Metrics

- Cyclomatic complexity
  - Linearly independent paths through code
  - Derive control flow graphs for functional units
  - $M = E - N + 2P$
  - Aim to minimise this number to a threshold, and refactor unit when threshold reached

Formal approach to maintainability

Complexity Metrics

- Halstead complexity measures
  - Extract measurable properties of software, computed statically from code
  - Evaluate relationships between properties, including
    - Program vocabulary
    - Program volume
    - Program difficulty
    - Estimate of delivered bugs

Supporting technologies

Doxygen

- Mature
- Feature-rich
- Useful syntax
- Drastically increases implementation time
- Default output not very pretty

Supporting technologies

GitHub

- Wiki included with every repository
- Integrated issue tracking
- Neat collaboration mechanisms
- Opinionated
- Time-consuming

Supporting technologies

Jupyter Notebooks

- Perfect for tutorials/worked examples
- Accessible
- Straightforward to integrate and share

Maintainability: audience experience

- Have you worked on a low-maintainability project?
- How do you foster maintainability within your projects?

Summary

- Make a map
- Follow the map, try not to let yourself get lost
- Ensure that others can both follow the map and make changes

Resources

- Google Drive
- Markdown
- Treesheets
- Git
- Subversion
- GNU Make
- Snakemake
- Airflow
- Vagrant
- GitHub
- Software Sustainability Institute
- Cyclomatic complexity
- Halstead complexity measures
- Doxygen
- Jupyter