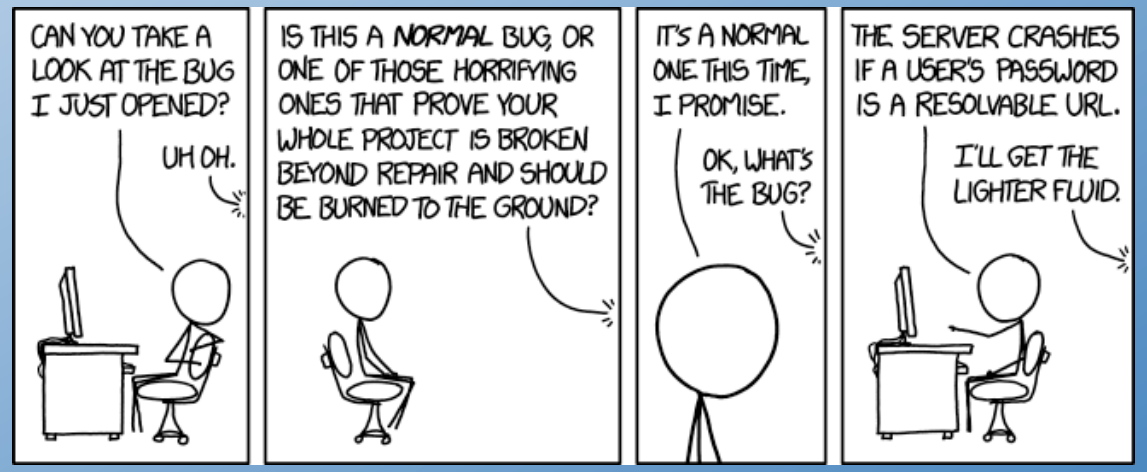
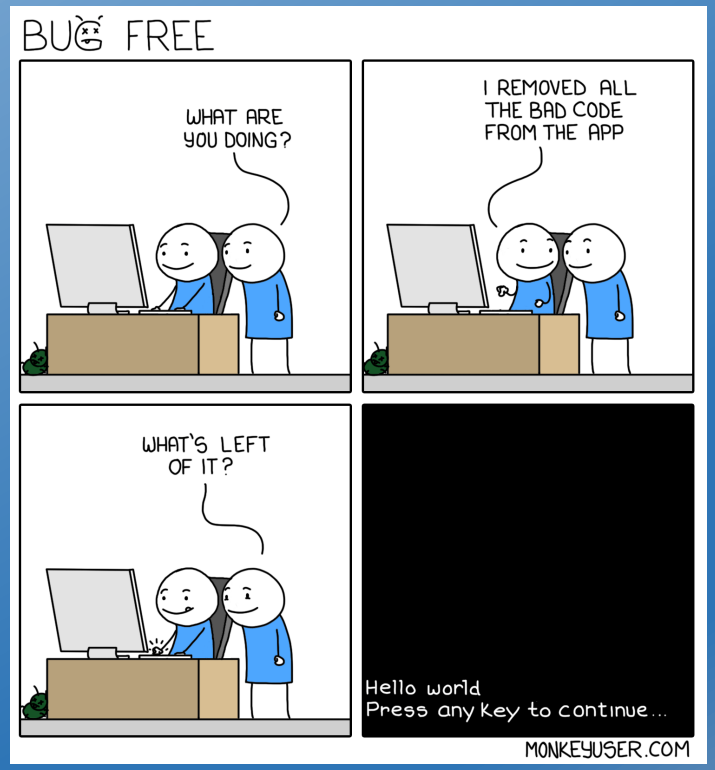


Maintenance II

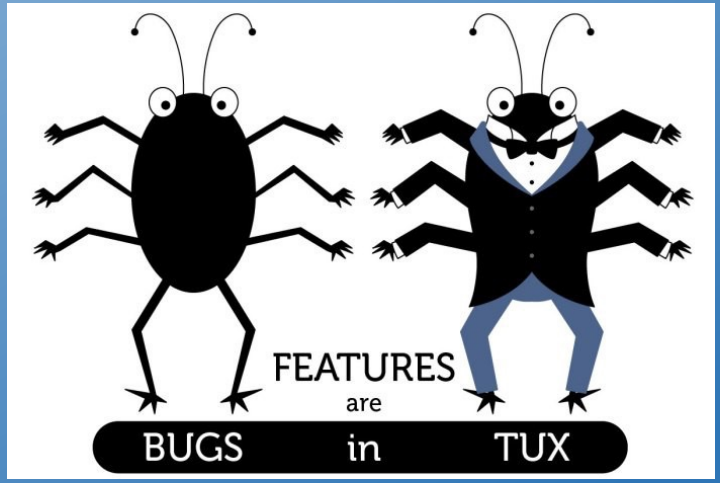
CSCI2340: Software Engineering of Large Systems

Steven P. Reiss



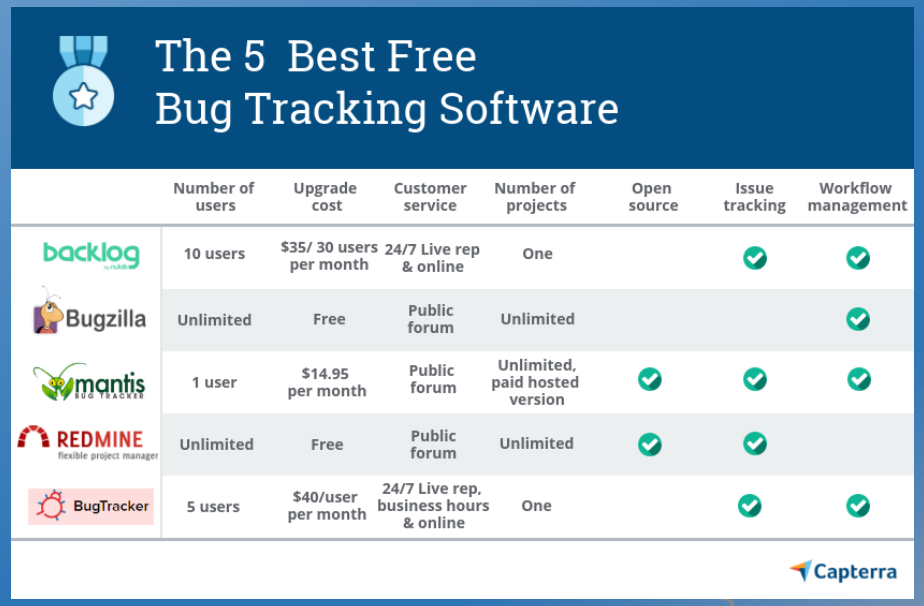
Bugs and Features

- Much of maintenance is bug and feature driven
  - You will find bugs
  - Other programmers will find bugs
  - Your QA team will find bugs
  - Users will find bugs
  - Your team want to add new features
  - Your management will want new features
  - Users will want new features
- You will need to track, prioritize, and address these issues
  - To know what to work on next
  - To understand the state of the system



Bug Tracking

- If there are a small number of issues (10s)
  - You can do it by hand or on paper
  - I use the Reminders app on the Mac for example
  - But this lacks history, is not shared or prioritized, …
- But a large system with a user base can have many more issues
  - 100s or 1000s outstanding is not unusual
  - Need something more sophisticated
- Tools exist for this purpose
  - Bug tracking or problem tracking system
  - JIRA, , YouTrack, GitHub Issues…
  - Organized database of bugs & features



The 5 Best Free Bug Tracking Software

	Number of users	Upgrade cost	Customer service	Number of projects	Open source	Issue tracking	Workflow management
backlog	10 users	$35/ 30 users per month	24/7 Live rep & online	One		✓	✓
Bugzilla	Unlimited	Free	Public forum	Unlimited			✓
mantis BUG TRACKER	1 user	$14.95 per month	Public forum	Unlimited, paid hosted version	✓	✓	✓
REDMINE flexible project manager	Unlimited	Free	Public forum	Unlimited	✓	✓	
BugTracker	5 users	$40/user per month	24/7 Live rep, business hours & online	One		✓	✓

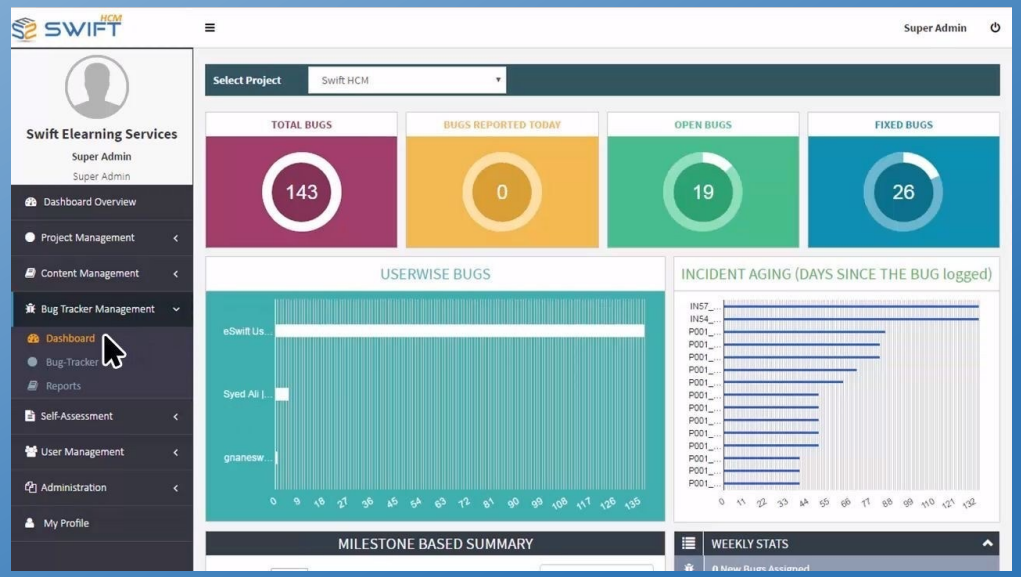
Capterra

Bug Tracking System Features

- Let you define bugs and feature requests
  - Stored in a permanent repository
- Assign these a unique id
  - That can be used in the code and in commits
  - That are permanent
- Link bugs with associated bugs (avoid duplicates)
- Give bugs a priority (critical, severe, … feature request)
- Assign bugs to a developer to be fixed or implemented
- Track status of entries (submitted, validated, fixing, done)
- Search for bugs using various criteria and terms
- View the history of a bug or feature request

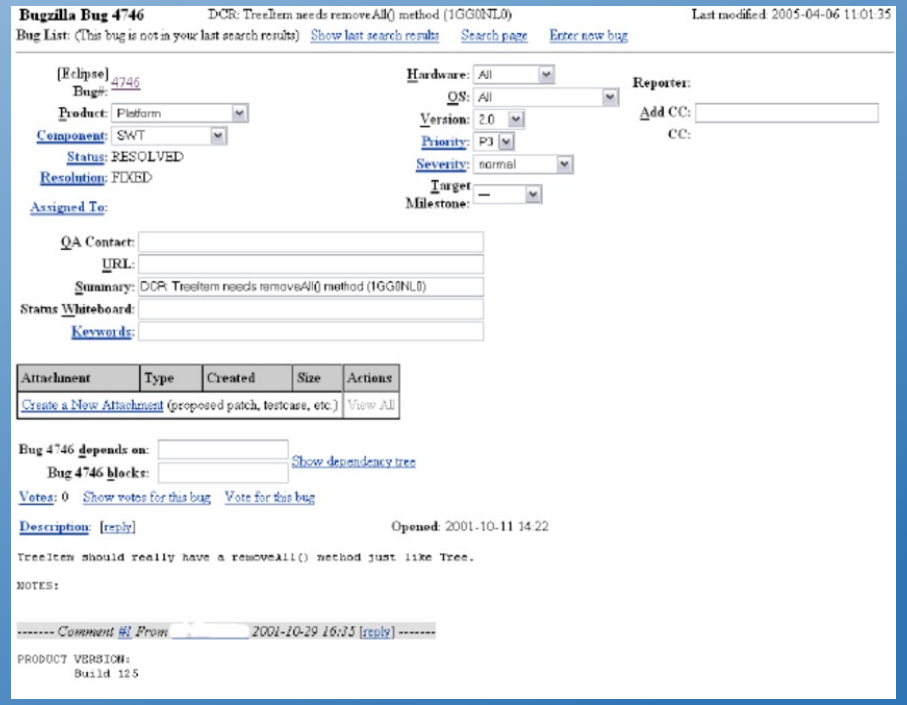
Bug Tracking System Features

- Add comment, notes, etc. to entries
  - Questions & answers as well
- Sending email on status changes
- Statistics (for management)
  - Track the state of the system
  - Track progress
- User-accessible portal
  - Users can comment on bugs
  - Users can report bugs
  - Users can vote bugs up or down



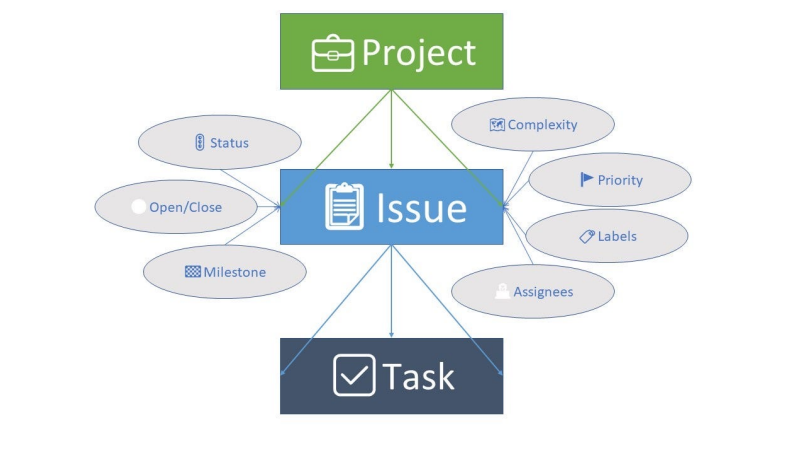
Bug Tracking System Problems

- A bit of overhead is involved
  - Can take 5-10 minutes to enter a new bug
    - Or feature request
    - Or more, depending on problem
    - Entering all the necessary data
    - Ensuring it is a new bug, not a duplicate
  - Modern systems have made this easier
    - By providing more flexibility
- Geared toward larger organizations
  - Seems like overkill for an individual or smaller project
- Not that flexible or easy to adapt to your flow
  - Difficult to customize the interface to your system



Modern Systems: GitHub Issues

- Simple interface for a bug database
  - Not structured like other systems, more freeform
  - Wiki-like description, comments, feedback on bug
  - Issues have a number
    - Number only, starting at 1
  - Issues have labels or tags
    - Default is type of bug.
    - Can be expanded to include priority
  - Issues can be assigned to a developer
  - Issues are associated with a project and a branch
  - Issues have a state (open/closed)
- Interface can be customized to project/team
  - For example, the set of tags available
- Notifications for status changes



Homework

- If you don’t do so already, try using GitHub Issues to track bugs in your project.
  - Ensure you have at least one issue (even if it is ”Start using GitHub Issues”)

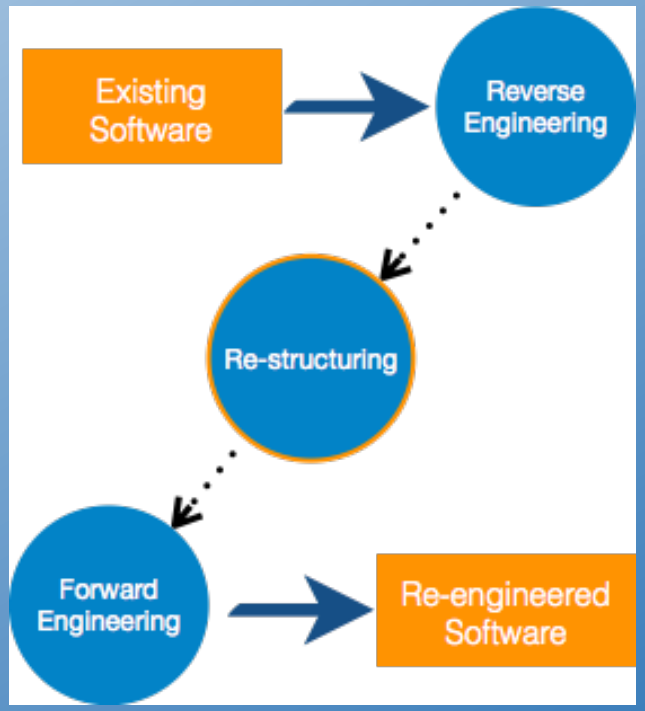
Maintenance Programming

- Maintenance programmers work on other people's code
  - When assigned a bug or feature to address
  - Possibly on the same project, possibly on a different one
  - Possibly on open-source code
- Maintenance programming is different from initial coding
  - And from maintaining your own code
  - In how you approach the code
  - In how you write the code

Approaching the Code

- Determine where the change should be made
  - This is fault localization
  - Experimentation – find the code that causes the bug
    - Easier if there is a test case you can use in the debugger
  - By searching over the source
    - Looking for error messages, names, stack traces
  - By asking others or looking at (non-existent) documentation
  - Like fixing your own bugs, BUT…
- Don't attempt to understand the whole system
  - Concentrate on the problem at hand
    - Only understand what the minimum needed to locate the problem
  - Try to find pieces of the code that might be relevant
    - Using names, coverage, …
    - Using the IDE (name search, calls to a method, …)
    - Using documentation
- Fault localization is often the most difficult part of maintenance programming

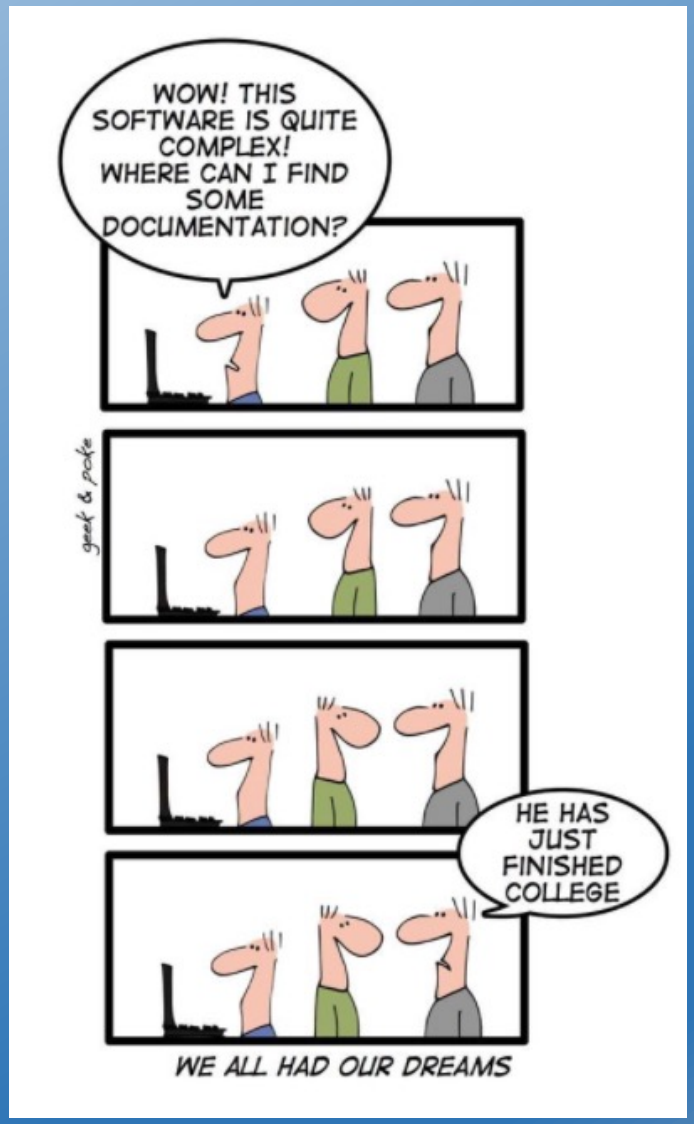
Fixing the Code



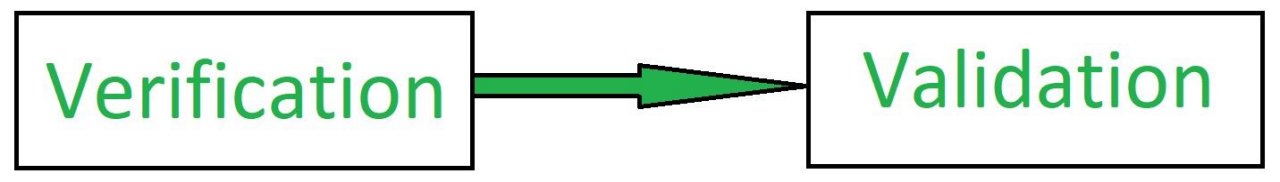
- Develop a fix that addresses the problem
  - Like fixing your own bugs, BUT …
  - With a minimal effect on anything else
    - Better not to try to fix the rest of the system
    - Concentrate on the problem at hand
  - But be general enough to handle related problems
    - If you can prove such problems exist
  - Understand the full effect of the code
    - What it does to the local function
    - If those changes flow out of the function, understand what it does to callers
    - Propagate this to callers of these as well – but try to minimize the effects
- Validate the fix manually
  - Check anything else done in the local area
    - Think about possible consequences of the fix
  - Find a fix that causes **minimal changes** to the function and to its return values
  - Check all call sites and assumptions
  - Check what other conditions might apply
  - Ask why it worked in general and failed in this case
  - Satisfy (prove to) yourself that the fix works and doesn't affect other cases

Writing the Code

- Use the style of the existing code
  - Not your personal style
  - Not your project's style if it is a library
- Add a comment for the fix
  - With your name, date
  - With the bug id if there is one
  - With information about the fix
- Write good code
  - Others will read it and judge you by it



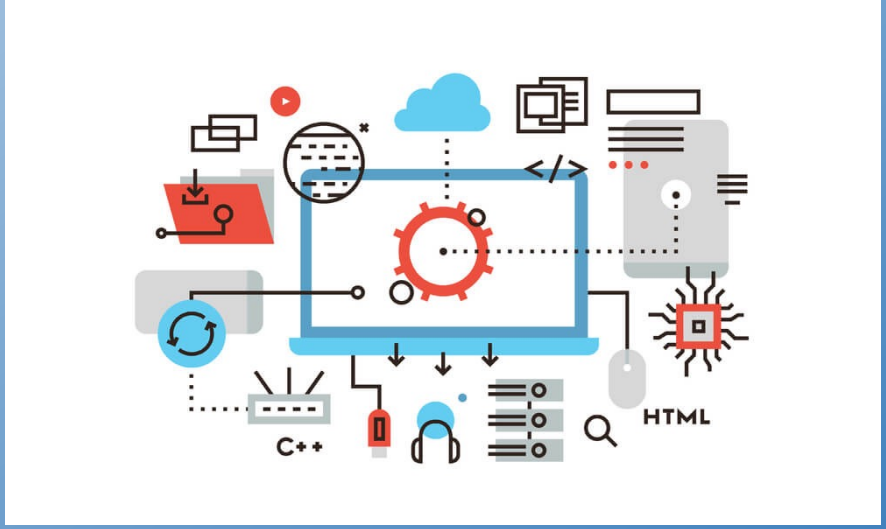
Validate the Code



- On paper – manually check the actual code
  - Prove to yourself that the code works
  - Double check and triple check
- With the original problem
  - Create a test case if you didn't start with one
  - Add to your test suite if possible
- With other test cases
  - Using the existing test suite
  - Adding other test cases that you see that might be at issue
    - Based on your analysis of what other things might be affected
- Maintenance program is fun and challenging
  - What is the minimum you need to understand of the system to fix the bug

Monitoring Your System

- You want to know about your system
  - Is it being used
  - How is it being used
  - What are the problems users are having
  - What are the problems the system is having
  - Are there performance issues
  - Are there security issues
  - Has your data been compromised
  - Where should you put your future efforts
- How can this be done
  - By asking your users
    - Pop-ups, surveys, feedback requests
  - Automatically
    - In various ways

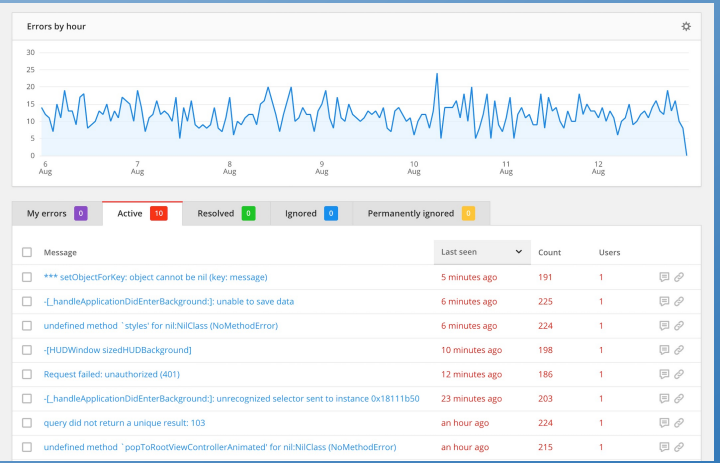


Automatic Monitoring: Analyzing Logs

- Of URL requests to the server (RESTful and otherwise)
  - Log analysis tools exist to help here
- Of command sequences (without sensitive data)
  - Operating system logs; self-generated logs
  - Log analysis tools can help here if logs are structured
  - Code Bubbles sends anonymized command info to server
    - These can then be analyzed off-line
- From your applications logging
  - You are writing log files – make them easy to interpret
    - Easy to find relevant information in
    - Structured in some way
  - Our loggers have <module>:<severity>:<thread>:<message>
  - Other loggers have their own standard or structured format

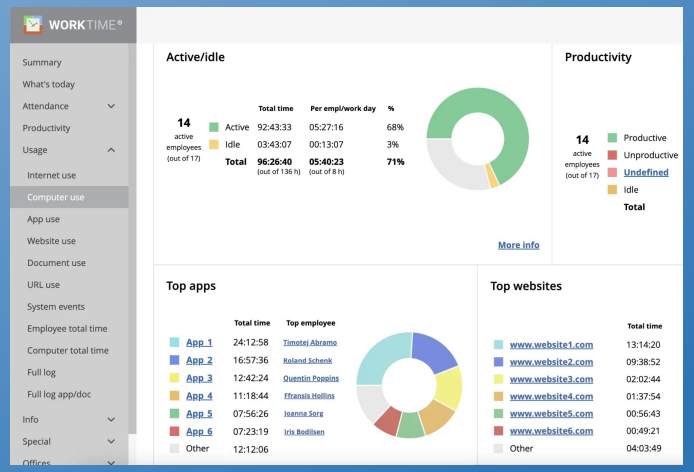
Automatic Monitoring: Bug Tracking

- Crash reports sent automatically
  - Many applications request permission to do so
  - On force quit
  - On internal error
  - Tools exist to analyze these if they are structured
- Even if the system can recover from the error
  - You can send information about the problem
  - Need to get user permission for this (at installation)
  - Code Bubbles detects and sends problems to our server
    - Which we check periodically

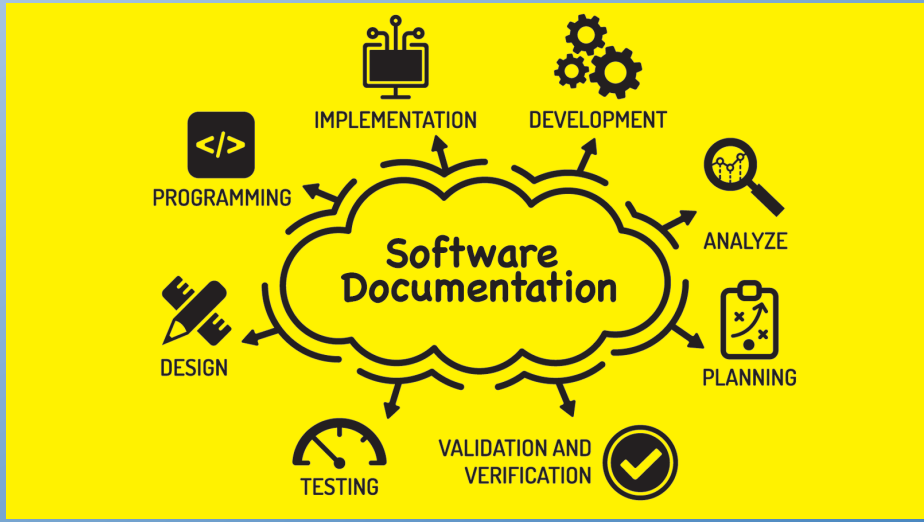


Automatic Monitoring: Usage

- You can request permission to track usage
  - To help improve your system
  - From user at installation
- Need to instrument the software
  - To provide the necessary data
  - Better to do it at random for short periods
    - You can still get a good sample
    - And the user shouldn't notice the instrumentation
- Output the information so it can be analyzed
  - Standard (structured) format usable with log analysis tools
  - Code Bubbles command logs

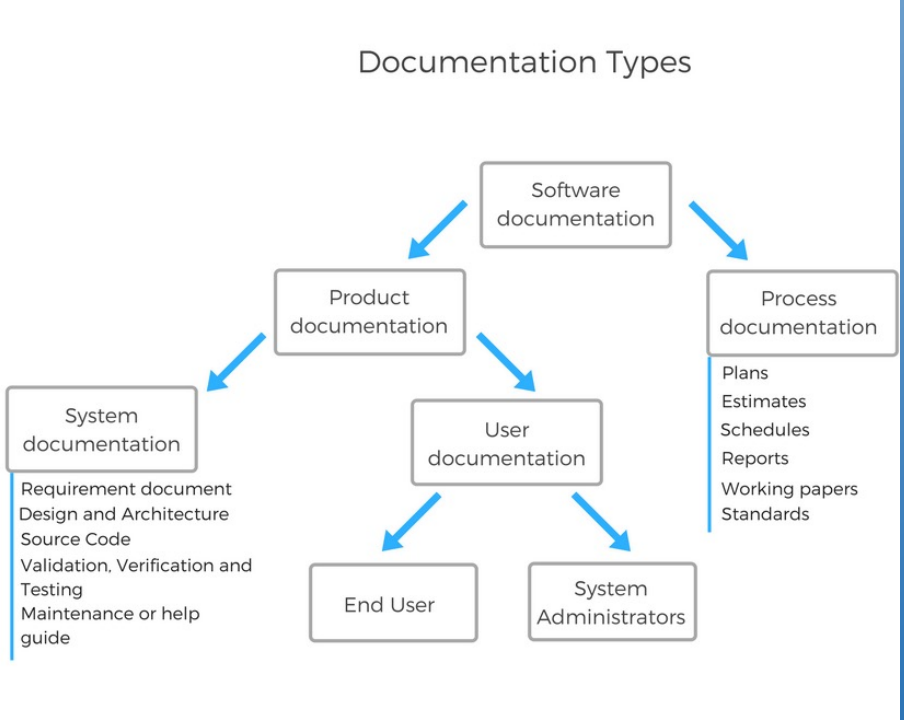


Documentation



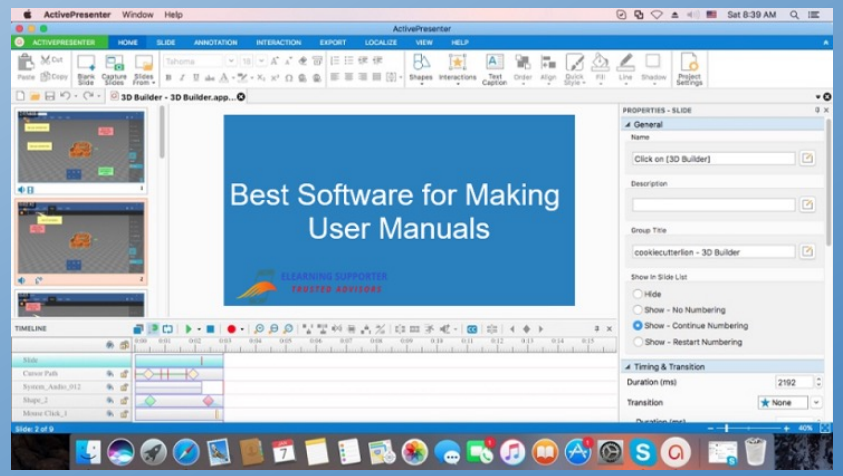
- Users expect documentation
  - Even if they don't use it
    - A system that needs documentation is a faulty system (poor user interface)
    - How often do you read documentation
  - Actually, most people do read documentation
    - But in subtle forms
- Programmers hope for documentation
  - When they must maintain the code
  - When they want to understand or reuse the code
  - And they hope the documentation is accurate and up-to-date
    - But you should be skeptical

Forms of Documentation



- User Documentation
  - Manuals
  - Tutorials
  - Help systems (searchable manuals)
  - Tool tips
- Code Documentation
  - API descriptions (JavaDoc)
  - Usage examples
  - In line and block comments
  - Design documents, interfaces, facades, UML
  - **Get in the habit of documenting as you write and rewrite**
- Note that documentation tends to get out of date as code evolves

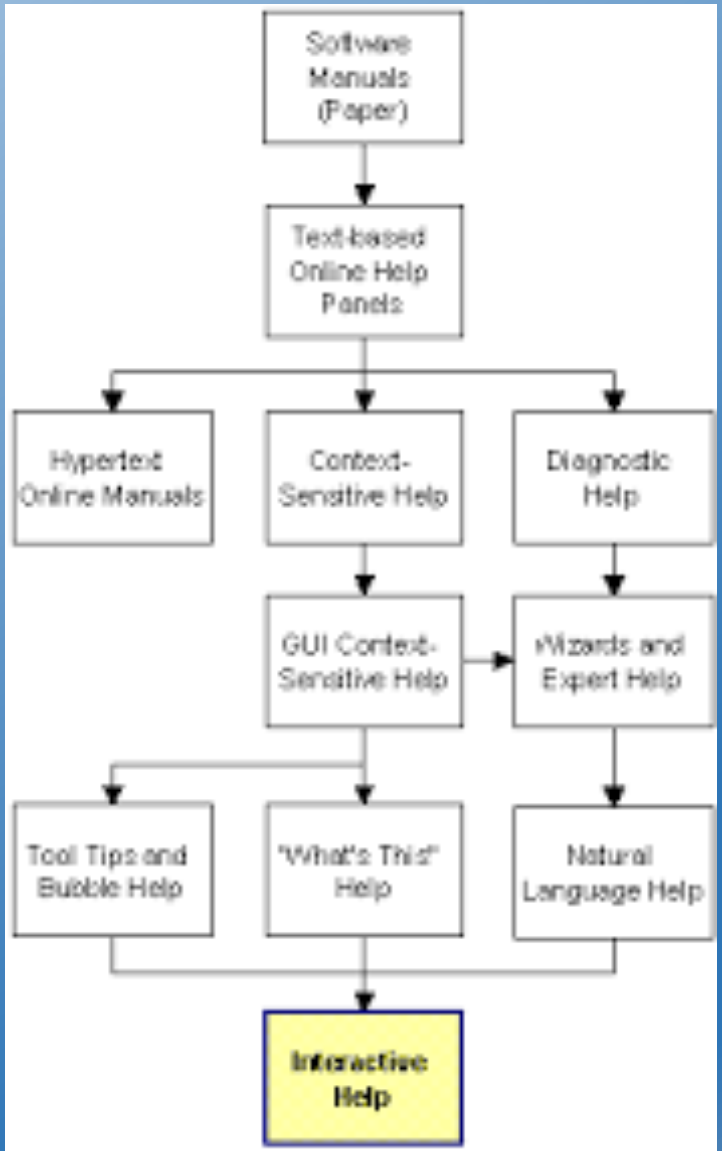
User Manuals



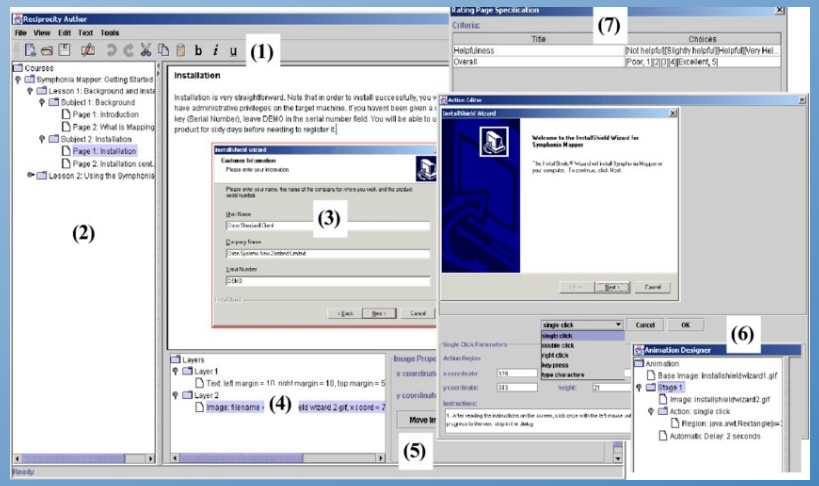
- Written so they can be used interactively
  - Small units, properly labeled
  - Indexed
  - Tools exist for this (EBT, FrameMaker)
  - Many of the descriptions are too simplistic to be useful for difficult problems
    - Manuals are not detailed enough to be helpful
- Difficult (tedious) to write
  - Most people only read a small portion
    - Need to be concise but complete
  - Determining what is relevant or important
  - Documentation writers might not know software that well
    - Programmers don’t have time/knowledge/ability to write manual well
  - Indexing needs to handle different vocabularies
    - User terminology and programmer terminology often differ

Interactive Help

- Context-sensitive help systems
  - F1 brings up help based on current location
- Other forms of help
  - Code Bubbles help (DEMO)

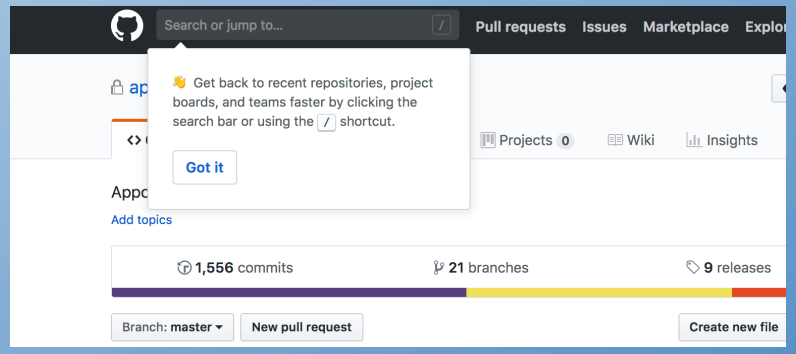


Tutorials



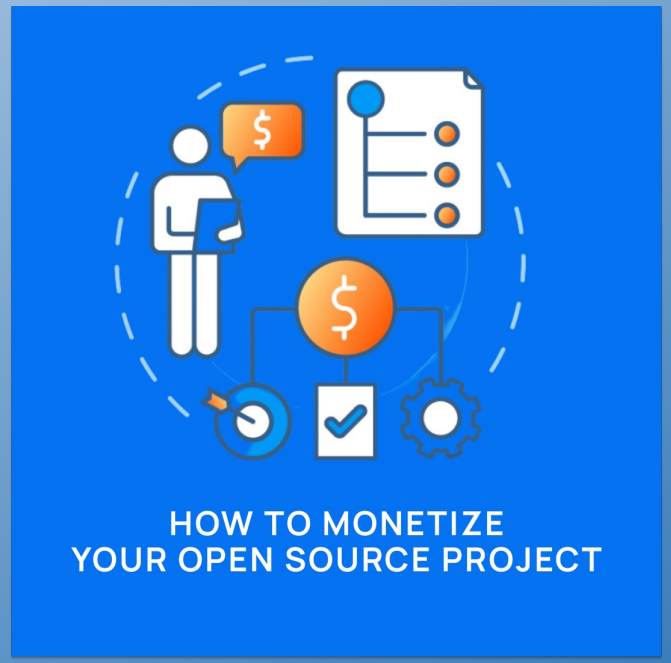
- A tutorial is a good start for understanding
  - Attempt to illustrate main features
  - Show off the system capabilities
  - Give users enough knowledge to start doing their own work
- A video of the tutorial (or the system in action) can help
  - To promote the system
  - To help users understand how to use it
  - To help understand the tutorial
  - Necessary for research software
- A hands-on tutorial (possibly with a video) is better
- Tutorials should be easy for the user to install and use
  - Consider the docker tutorial you did earlier

Tool Tips



- These are the most convenient documentation tools
  - Don't require any effort on the user's part
  - Always available
  - Users are used to them
  - Available in most front-end and web tool kits
- You should provide tool tips for all UI features
  - Every button, icon, text field, …
  - Any place else where additional information can be helpful
    - Bubbles code editor – elision, line number, debugger context, variable values, …
- Tool tips can be more than just pop-ups
  - Consider hints in VS-Code
  - Consider the help pop-ups in code bubbles
- Be careful that tool tips don't get in the user's way

Monetizing Your Project

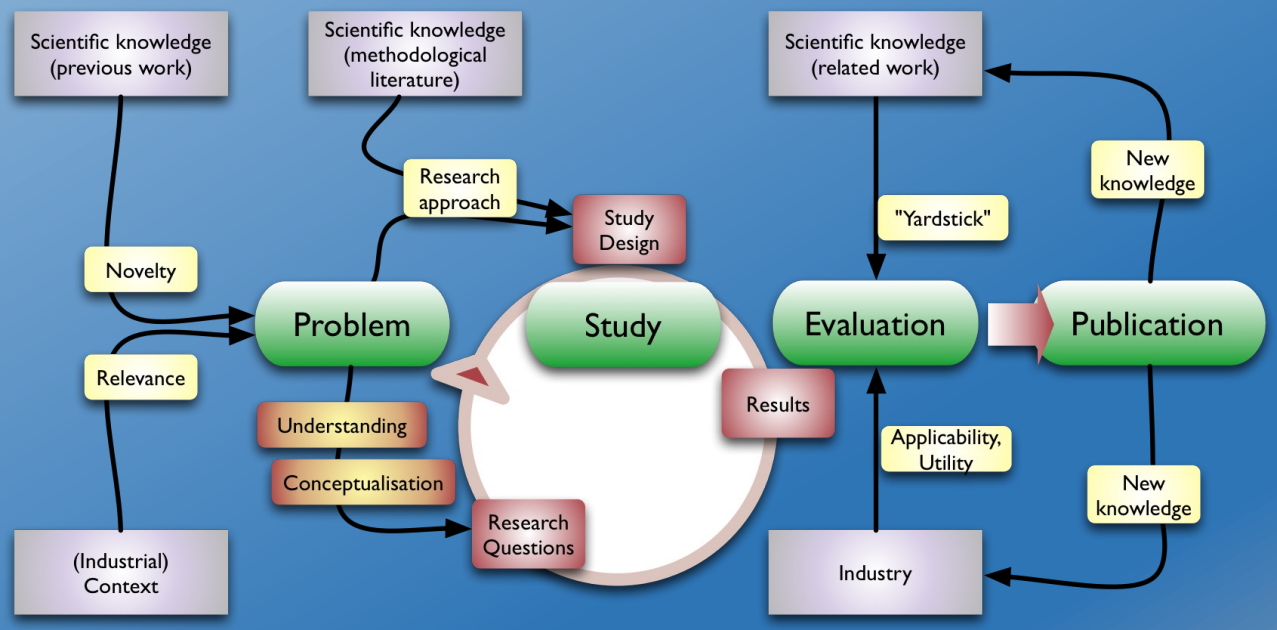


- Software has costs to maintain
  - VMs are not free
  - Need hardware and software maintenance
  - Need programmers
- You need a business plan to support long-lived software
  - Ads, contributions, subscriptions
  - Selling data
  - Supported by company
  - These might need to be part of requirements/specifications
  - Support does not mean profit (but it can)

Course Review

Software Engineering Research

- If anyone is interested in research
  - On any of the topics covered in the course
  - Or related topics
- Drop me an email
  - Or we can schedule a meeting



THANK YOU!!!

PROJECT Presentations

- Presentation Schedule
  - 11/26: User Interface Generation
  - 12/03: Speech
  - 12/03: Sense IoT
  - 12/05: DJ Mix
  - 12/05: LLAMA
  - 12/10: Agentic
  - 12/10: Accessibility
- Final Demonstrations 12/12
  - 10 minutes each as needed (optional)
- Each project gets up to 40 minutes (1/2 class)
  - Present the problem (requirements, specifications)
  - Present the solution (design)
  - Present the system (implementation)
  - Demo or video of the system in action (live preferred on the 12th)
  - What you learned
  - Future plans
  - Questions
- Feel free to invite your friends and sponsors and anyone interested in the project