

Publishing Source Code for Reuse and Maintenance

Jim Fawcett

11 October 2019

Publishing Code Topics

- Software Salvage and Reuse
- Publishing Code Components
- Reuse Prologue
- Website Structure
- Quality Control
- Experiments
- Conclusions

Software Salvage and Reuse

- Software Components
 - Software package that has a single purpose, few dependencies, and is useful for building software systems – example: blocking queue
- Salvage (good)
 - Using an existing component with minor modifications
 - That creates another component that must be configured and managed
- Reuse (better)
 - Reuse an existing component with **no modification**
 - Used by composing, using as template argument, or using as base for derived classes

Software Reuse – Good news and bad news

- Reuse of compiler libraries has been spectacularly successful
 - Each language defines a set of libraries that support building projects
 - Updated with each new standardization of the language.
- Software reuse in the academic, industrial, and commercial domains has been disappointing
 - Typical use is:
 - Grab the last relevant project(s)
 - Attempt to throw away the unneeded parts
 - Sometimes we keep unneeded parts because too much breaks if we remove
 - That causes maintenance problems
 - Add needed new parts
 - Spend a lot of time fixing breakage

Application Domain Targets – Candidates for Support?

- Academic Research: 5 – 10 code developers
 - Code life-time: 3 – 5 years
 - Example: Natural Language Processing (NLP)
- Open Source Development: 5 – 1000 active developers, many casual contributors
 - Code life-time: 10 - 20 years
 - Example: Linux
- Industrial Development: 5 – 10 developers
 - Code life-time: 5 – 20 years
 - Example: Machine Tool Control
- Commercial Products: 10 – 30 developers
 - Code life-time: 20 – 30 years
 - Example: Microsoft Word
- Aerospace Programs: 5 – 200 developers
 - Code life-time: 20 – 30 years
 - Example: Submarine control, Area surveillance, ..

Supporting Software Reuse

- For reuse a software component should be designed so that reusing in a new project is quicker and easier than creating from scratch
 - How to do that is a topic for another story.
 - There are many good examples, e.g., the C++ Standard Template Library and Apache HttpComponents.
- Components must be available
 - That implies some cloud-based repository
- Components need to be documented
 - Concept, use statement, use examples, design, status


Publishing Code Components for (Re)use

- The goal of this site is to improve that process by publishing code in an effective way
- Publishing some code artifact is the act of making it available, in usable form.
- Five main facets:
 - Containment
 - Delivery
 - Location
 - Interpretation
 - Quality Control

The Issues

- Source code containment and delivery are solved problems
 - Cloud-based facilities like github do that very well
- The issues are finding and understanding code relevant to a need
 - We want most code repositories to be large – to support broad reuse
 - How do we find, in a large repository, code that fills some need?
- Website documentation, colocated with source code - a good option
 - To support both salvage and reuse, documentation needs to provide information about the component's concept, design, and typical use

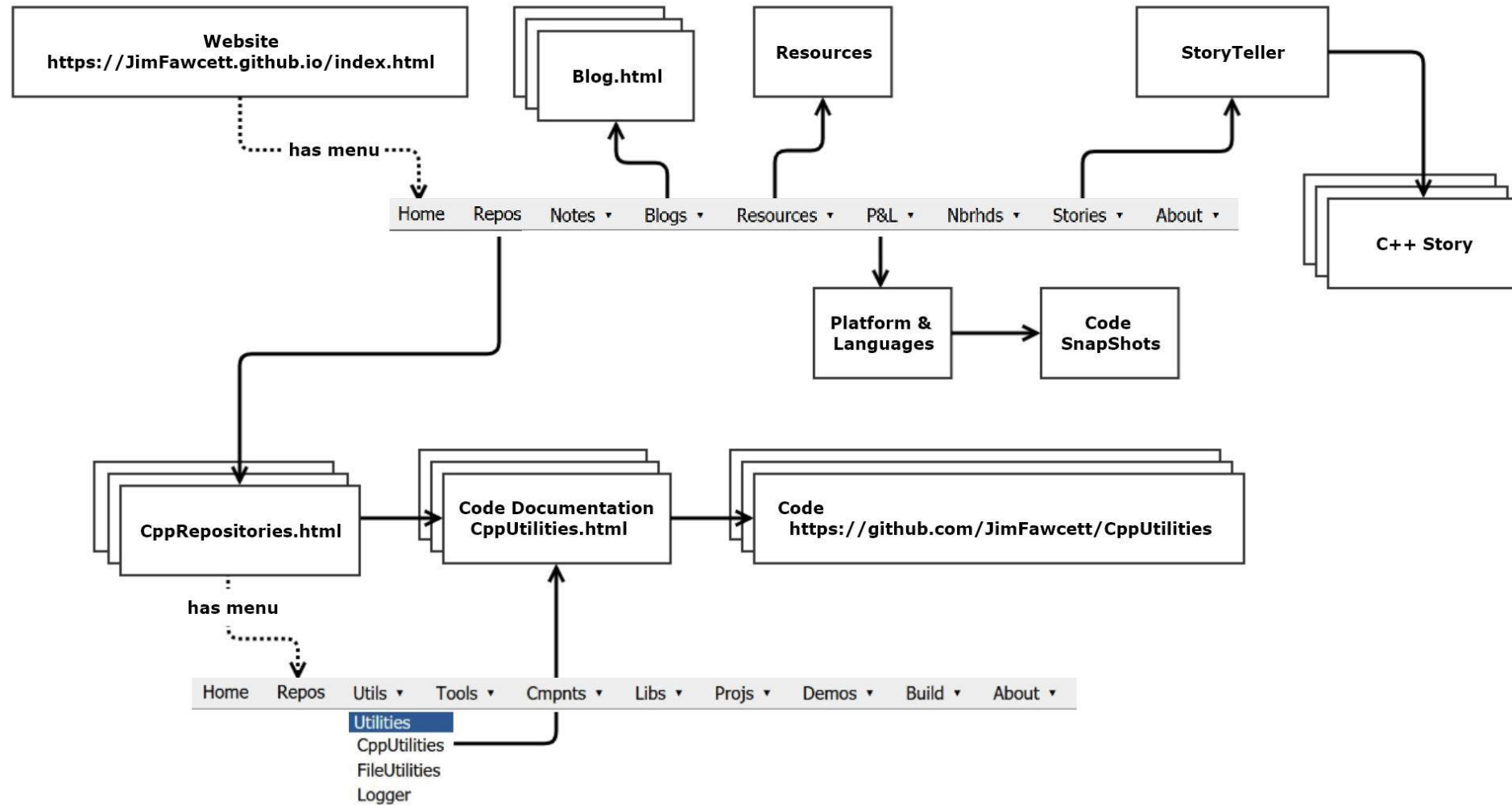
Story Prologue

- This is a presentation of goals and features of a website designed to publish source code in support of software reuse.
 - <https://JimFawcett.github.io>  Take a quick tour
- The site is a second-generation facility based on experience with an academic website:
 - <https://ecs.syr.edu/faculty/fawcett/handouts/Webpages/fawcettHome.htm>
- I used that site for graduate software design courses taught at Syracuse University for many years.
 - Published lecture content
 - Provided access to code components students used for class projects

Website for Publishing Reusable Components

- A site for publishing source code will need:
 - A structure for holding organized collections of code
 - An intuitive navigation process to find specific code and resources
 - Documentation for each component
 - Concept, Design, Usage, Status
 - Additional resources to help users understand relevant technologies
 - Language and platform references
 - Brief code snapshots with commentary, provided as webpages
 - Related blogs and opinion pieces
 - Code standards
 - Stories and Videos
 - Discussions, each focused on a single theme

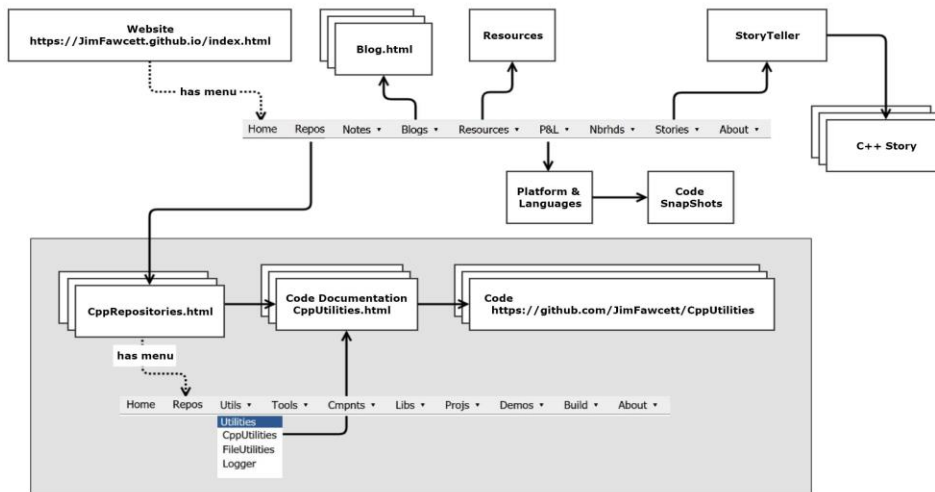
Site Structure – <https://JimFawcett.github.io>



Site Structure - Code Repos

<https://JimFawcett.github.io/Repositories.html>

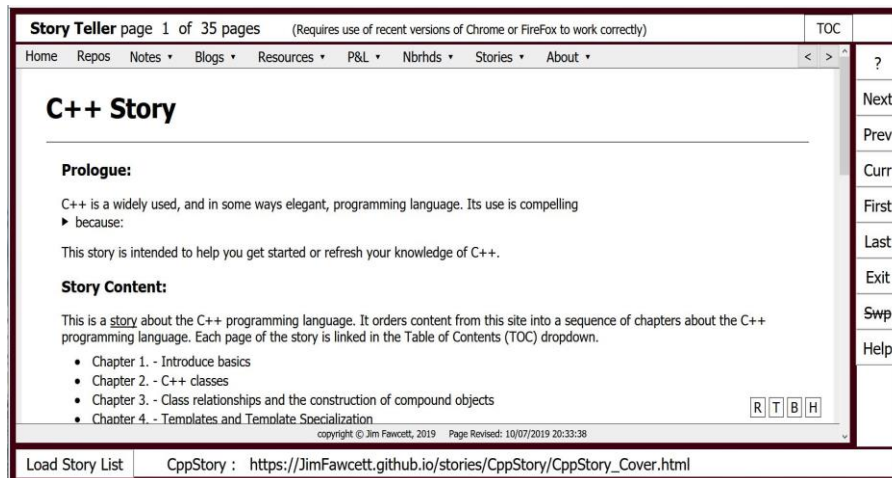
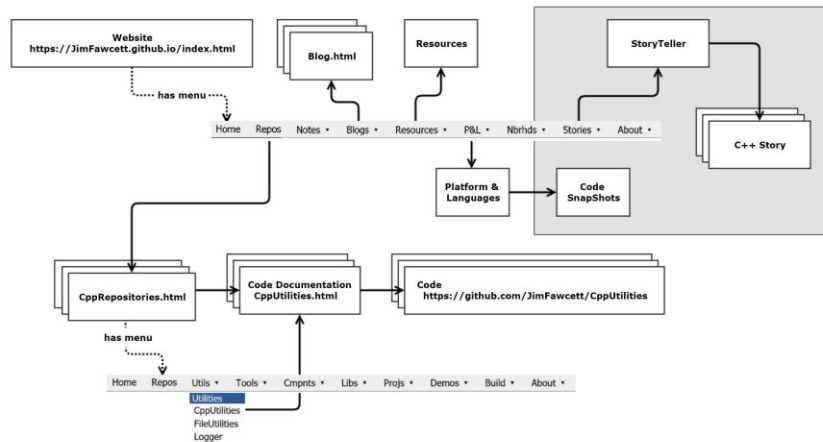
- Code Repositories are the most important part
- Goal is to have large collections to promote broad reuse
- Navigation is an issue. Solution:
 - Link from Home menu
 - Factor into (language or product specific) collections
 - Factor each collection into individual repositories (utilities, tools, ...)
 - Link first to documentation which then links to code folder



The screenshot shows the 'ThreadPool Repository' page. The header includes 'ThreadPool code' and 'ThreadPool Repository' with the subtitle 'Implements Tasks using a thread-pool'. The main content area is divided into sections: 'Concept:', 'Operation:', 'Design:', 'Implementation:', 'Build:', and 'Status:'. The 'Concept:' section describes the ThreadPool as a thread-safe blocking queue. The 'Operation:' section describes the thread execution loop. The 'Design:' section describes the workitem type and number of threads. The 'Implementation:' section describes the ThreadPool and Task classes. The 'Build:' section describes the build process. The 'Status:' section describes the plan for a wrapper. The footer includes 'copyright © Jim Fawcett, 2019 Page Revised: 10/13/2019 11:51:31' and a 'SRTBHPN' logo.

Site Structure – Stories

<https://JimFawcett.github.io/StoryTeller.html>



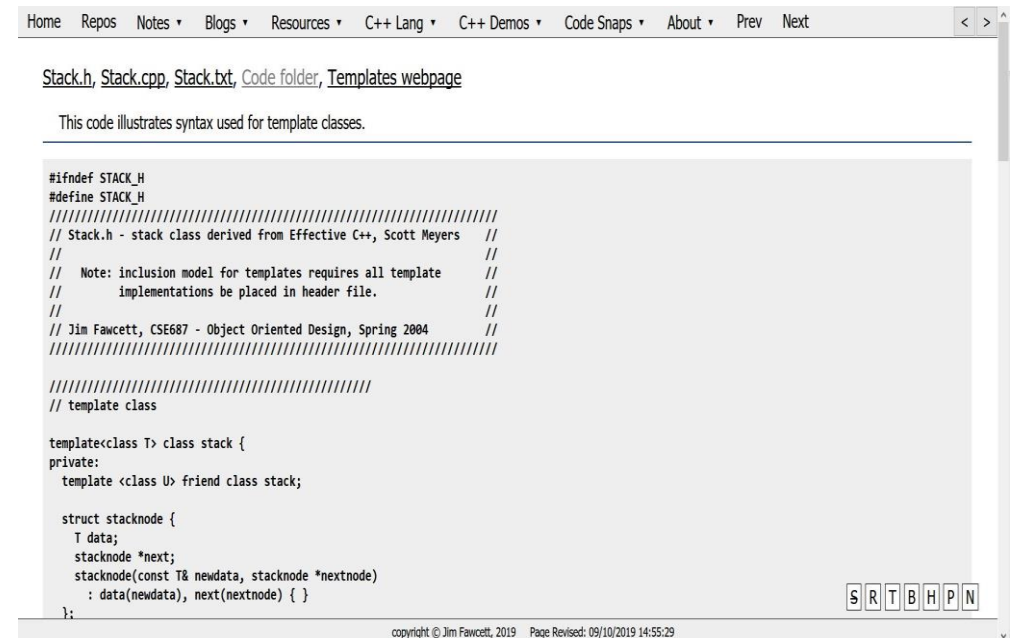
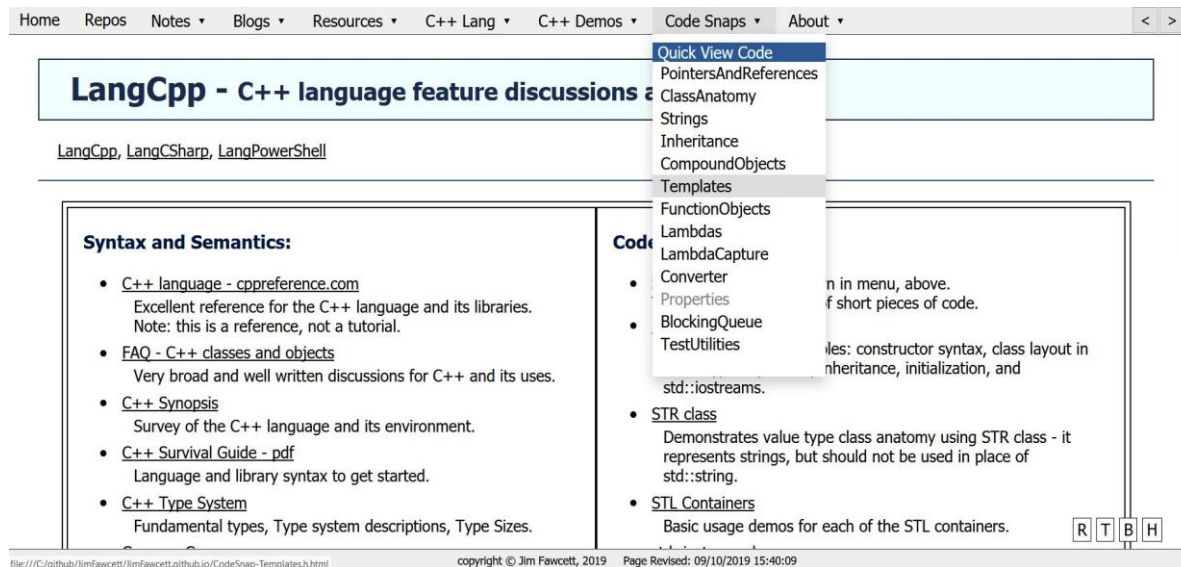
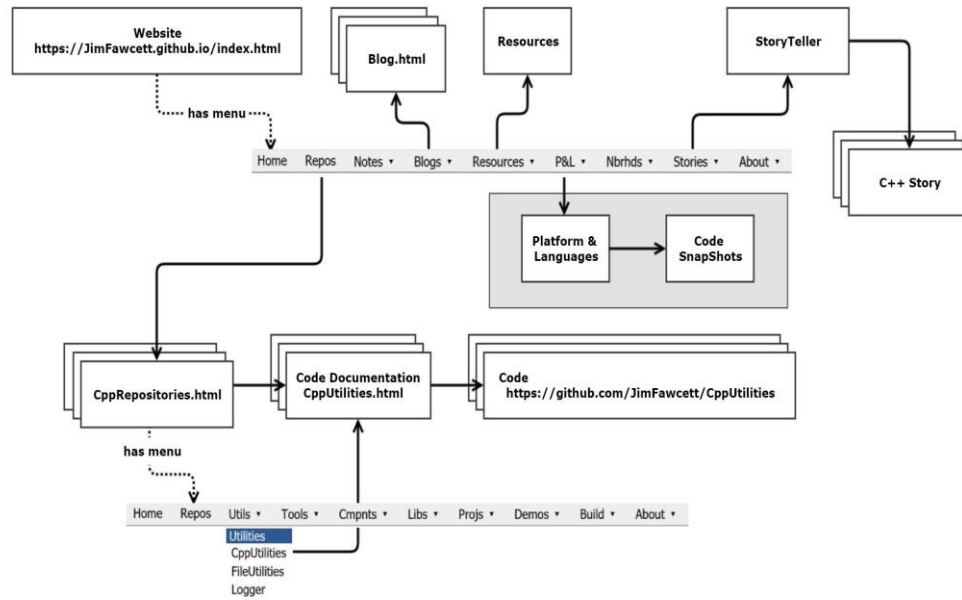
- Stories are organized collections of pages from the site that all focus on a single theme
- Intent is to help users grapple with site content
- Examples are:
 - C++ story – tutorial material
 - SiteStory – description of this site
 - MLiPS – guest story about ML
- Image to the left is StoryTeller interface

Site Structure – Code Snaps

<https://JimFawcett.github.io/LangCpp.html>

- CodeSnaps

- Source code converted to HTML page
- Provides quick access to views of important code fragments
- No need to download from repository to explore



Quality Control

- Without quality control it's easy for repositories to become a sea of flotsam[1] and jetsam[2]
 - Some good content, but a lot of content that isn't ready for reuse
- To avoid that destination two things are necessary:
 - An effective structure for disclosing the contents of a repository
 - This site provides a candidate
 - Willingness of the site sponsor to invest in review and improvement
 - Standards components must meet to become candidates
 - Knowledgeable and productive person(s) to evaluate a component against the standards and admit or reject
 - Continuing background review activity – is this component still valuable?
- Note that github provides tools to help, e.g., wikis, charts, ...

[1] Flotsam - cargo that surfaces from a sunken ship

[2] Jetsam – cargo that is intentionally cast from a ship in distress.

Experiments

<https://JimFawcett.github.io/Tests.html>,
https://JimFawcett.github.io/SiteStory_4.html

- Experiments to make the site's publishing process more effective are continuing:
 - Developing UI widgets to use webpage real estate effectively
 - User driven diagram resizer
 - Slide-in panels
 - Slide show
 - Code blocks for presentation
 - Photo styling
 - Navigation schemes
 - Dropdown menus
 - Page sequences (defined by hidden links)
 - Navigation buttons and key presses
 - Continuing to think about site organization schemes

Status and Conclusions

- Status - <https://JimFawcett.github.io>:
 - Most of the structure described is in place
 - More than 40 repositories of mostly C++ code are published
 - Several stories are published
 - Other resources and CodeSnaps are available from the site
- Plans
 - Install more code repositories
 - Install starter site
 - Start to add videos
- Conclusions:
 - Static sites seem like a good tool for publishing
 - Main issue is text or keyword searches not available with static sites
 - Can be addressed with local mirror and tools for synchronizing and local searching
 - Required effort is reasonable for the expected payoff
 - This site went from zero to current status in four months of my time
 - However, I've built a site like this before.

Appendix - Domains

Application Domain Targets – Candidates for Support?

- Academic Research: 5 – 10 code developers
 - Code life-time: 3 – 5 years
 - Example: Natural Language Processing (NLP)
- Open Source Development: 5 – 1000 active developers, many casual contributors
 - Code life-time: 10 - 20 years
 - Example: Linux
- Industrial Development: 5 – 10 developers
 - Code life-time: 5 – 20 years
 - Example: Machine Tool Control
- Commercial Products: 10 – 30 developers
 - Code life-time: 20 – 30 years
 - Example: Microsoft Word
- Aerospace Programs: 5 – 200 developers
 - Code life-time: 20 – 30 years
 - Example: Submarine control, Area surveillance, ..

Open-Source Domain

Parts flow in, a product flows out

- Open-source projects like Linux
 - Locally developed code flows to the cloud
 - Local contributors push to remote development branch
 - A single large system flows from the cloud to local users
 - Download a distro
 - Open-source applications like node.js
 - Developed by small group of contributors
 - Users download and install system with additional imports
 - Open-source libraries like jQuery.js
 - Developed by small group of contributors
 - Users bind remote library to webpages with Content Delivery Network (CDN) links
- This domain already has a publishing model that has different goals than ours

Other Domains

Components installed, flow out to users

- Users search for a component to fill a current implementation need
 - Repository needs to be large, i.e., hold many components so there is a fairly good chance to find something useful
 - So search and interpretation need to be effective
 - For repositories with hundreds of components, that is not trivial
- The main issues are:

- Developing a useful search process that is intuitive and quick
 - Helping users interpret a found component
 - What does it do?
 - How is it designed?
 - How to integrate with existing code?
- These domains appear to be good candidates for the publication process proposed here

Academic Research Domain

- Academic Research: 5 – 10 code developers
 - Code life-time: 3 – 5 years
 - Example: Natural Language Processing (NLP)
- An academic researcher may have 2 or 3 doctoral candidates working on related parts of a research project.
- She may collaborate with two or three colleagues, perhaps at different universities.
 - Each of those colleagues may have a similar team working on related projects.
- There usually is continuing work on the same or related research projects for many years.
- It is quite common that this work develops software tools for gathering and analyzing data. Sometimes software is part of the end product, e.g., compilers for a new language, an architecture for streaming or classifying content, ...
- These software developments are rarely maintained well. Often documentation is nothing but code and the papers that describe research results and mention the software as an aside.
- Our methods are applicable.

Open Source Domain

- Open Source Development: 5 – 1000 active developers, many casual contributors
 - Code life-time: 10 - 20 years
 - Examples: Linux, Node.js, MongoDB
- Following an initial period of development, a project often settles into a maintenance mode:
 - Maintain the same mission and design.
 - Add new features and port to new platforms.
- Occasionally revise most of the code and support new missions.
- Documentation varies from poor to outstanding.
 - Linux documentation is one of the outstanding ones:
 - <https://www.kernel.org/doc/html/v4.10/index.html>
 - Some projects focus on api level documentation – how to use the code.
 - Some focus on maintenance – what are the parts, how are they related, code standards, ...
- Their goal is different than ours – our methods may not apply.

Industrial Development Domain

- Industrial Development: 5 – 10 developers
 - Code life-time: 5 – 20 years
 - Example: Machine Tool Control
- Code base starts with an initial product
- New products start from that initial code
 - Has a common code baseline been defined?
 - Code reviews are held during development
 - At product completion is code reviewed and refactored into reusable parts and product specific code?
 - Is the organization willing to provide overhead effort to evolve the common code base?
- Our methods are applicable. A scaled down version is probably appropriate.

Commercial Domain

- Commercial Products:
 - Code life-time: 20 – 30 years
 - Example: Microsoft Word
- Product may start with code framework, standards, and a specification
 - For Microsoft Office and many other products, the framework has been the Microsoft Component Object Model (COM) – technology was specifically designed for reuse
 - It appears that, moving forward, COM will be hidden with a wrapper technology - Windows RunTime (WinRT):
https://en.wikipedia.org/wiki/Windows_Runtime
 - Specification is developed by a Program Manager and reviewed by the development team
- Microsoft has spent a lot of resources on documentation, with mixed results
 - Executing a search in MSDN for WinRT results in 6,180,000 results.
 - That isn't a useful query – in fairness, the results are prioritized
 - It's quite common that when you arrive at a useful documentation page you get partial results with the remainder often linked to many pages scattered over the vast reaches of MSDN
- There are many Microsoft technologies each with its own products, APIs, code examples.
 - Some, like the .Net framework have outstanding organization and documentation.

Aerospace Domain

- Aerospace Programs: 5 – 200 developers
 - Code life-time: 20 – 30 years
 - Example: Submarine control, Area surveillance, ..
- Development is product oriented, starting with an initial contract, and continuing for possibly many years of enhancements and new contracts that build on the existing product technology
- An example is the development of area surveillance radar systems
 - Typical lifetime of a radar system is 20 years
 - The code has to be maintained over that lifetime, sometimes by the manufacturer, sometimes by the customer.
 - Once an initial contract has been successfully completed it is common for many contracts to be awarded for new versions.
 - Usually a large part of a new product is based on an existing one and will share a large fraction of its code
 - In most cases the new contract requires new features and enhancements – the customer looks at the original and decides how to embellish
- The Capability Maturity Model was developed beginning when the Air Force funded a study with the Software Engineering Institute. Its intent is to encourage DoD contractors to develop and maintain a consistent process for creation of software.
 - https://en.wikipedia.org/wiki/Capability_Maturity_Model
 - CMM provides guidelines, but it is a model and a yardstick for assessing capability of contractors
 - It does not provide specifics for tools and techniques that support reuse
- Our methods apply for individual product teams. Unknown how that would scale to corporation level.

That's all Folks!