Frameworks: research as reflection on practice

Greg Hartman
gghartma@cs.cmu.edu
28 April 2005

Frameworks weren’t widely known

- The 1984 issue of IEEE Transactions of Software Engineering is devoted to reuse
- Reusable building blocks are mentioned
  - Libraries of code
  - Libraries of designs
  - Libraries of specifications
- Also appearing:
  - General architectures are mentioned
  - Design patterns are mentioned
  - Code generators
  - Very high level languages
  - Classical reuse (reuse from finished applications)

But were evolving in GUI research

- 1976: MVC pattern observed in SmallTalk-76
  - But no reuse of code
- 1980: Reuse becomes a goal for SmallTalk-80
  - The framework was probably white-box (implement by inheriting)
  - This work is largely undocumented
  - I’d call this internal exploration
- 1981: Experience carried over into Lisa
  - All GUI problems are handled through a framework
    - This leads to complexity
      - The framework is still white-box
      - Apple discovers that external developers have trouble using it
      - I’d still call this internal exploration

Frameworks do go through the R&R steps

- But the steps are in a different order
- They move through the final steps very quickly

1976: Concept Formation
1981: Internal Exploration
1986: Artifact Popularization
1990: Technique Popularization

1976: Dev. & Extension
1981: Internal Exploration
1986: External Exploration
1986: Artifact Popularization
Frameworks become widely known
- 1984: Apple releases the Macintosh with no framework
  - However, they do have a huge API (the Toolbox)
    - Three volumes of API documentation
    - Hundreds of technical notes
    - This isn’t a great situation
  - Documentation filled with non-application specific glue code
  - Programmers can construct strange UIs
- 1986: Apple releases a framework called MacApp
  - It becomes wildly popular with developers
  - Usable without knowing the internals (black-box)
  - It ships with source code, allowing developers to tweak it
  - Other GUI system start working on frameworks
  - I’d call this external exploration/popularization

Frameworks are generalized
- In 1988, Johnson and Foote:
  - Describe a generic concept of frameworks
    - Like patterns, they describe a high-level structure of the application that is independent from the requirements
    - Like components, programmers reuse implementation
    - A later paper has the title frameworks = (components + patterns)
  - Describe a development methodology to create new frameworks
    - I’d call this concept formation
  - After this paper, we begin to see discussions about frameworks in domains beyond GUIs

Qualitative aspects of the stages
- Before concept formation, can’t tell the difference between technique (frameworks) and the artifact being produced (GUI libraries)
- Application of the technique to new areas often uncover problems, driving new research

Other projects encounter problems
- Estimating cost and benefit
  - MacApp had a huge market (all GUI applications) and greatly reduced the cost of development
  - In wider use, managers ask should we develop a framework that we will use (2, 5, 10) times?
- Composing multiple frameworks
  - What if you are a distributed GUI application? Which framework wins?
    - This is less of a problem with threading, since there is no such thing as inversion of control
- Understanding the true benefit of frameworks
  - Some companies have developed libraries of 100’s of frameworks
  - I don’t know of any that were widely adopted
  - How do developers know what frameworks to use?
How Reuse Influences Productivity

- Basili, Briand, Melo: 1996
- Type of paper: Experience Report (Case Study)
- Hypothesis:
  - Validation: Studied 8 systems written in C++ by students. Quantified the level of reuse, defect density, amount of rework, and time spent on the program. The reuse should be negatively correlated to the other variables.
- Result: Design of the study precludes concluding causation. Statistically significant correlations exist for defect density, rework not normalized to faults, and productivity. Reuse does not affect the amount of rework to fix an identified fault.
- Do I believe it? My only reservation is in the area of external validity: students may not be good substitutes for experienced programmers.

Quality, Productivity, and Learning...

- Morisio, Romano, Stamelos: 2002
- Type of paper: Experience Report (Case Study)
- Hypothesis: The use of frameworks improves productivity and quality. Net productivity = writes code faster (LOC/hour, not counting reuse), not just completes application faster. The developer will learn faster with the framework. Learning reduces the defect rate faster with frameworks.
- Validation: Hire an industrial programmer and have him complete six projects, some using a framework. Monitor the developer's productivity and look for learning effects.
- Result: Net productivity is higher, learning is faster in terms of productivity. Learning's influence on quality wasn't significant in frameworks, but the gap narrows with learning.
- Do I believe it? Yes, but I'm a bit surprised that net productivity was detectable. It would be helpful to have a deeper hypothesis of why this is the case.

Open areas of research in frameworks

- It would be helpful to quantify the cost of developing a framework and the benefit derived from using it.