The rise of frameworks

Greg Hartman
gghartma@cs.cmu.edu

24 March 2005
Classical approach to reuse

Sequential completion

1. Choose an existing program
2. Remove the unwanted code
3. Insert new code to create a “new” program
4. Repeat 1-3 until you have all of the programs that you need

What happens if you’re trying to build something like this?
Parnas applied stepwise refinement...

1. Start with a complete abstract program

2. Fill in some implementation details

3. Repeat until the program can execute

- Application to programming: reuse one of the abstract programs rather than a complete program
  - Saves time understanding the code
  - Less chance of ending up with more software than you need
...and module specification...

- Hide things that are likely to change inside modules

- Application to reuse:
  - form a tree of abstract designs
  - Order designs so the most general (embody ideas common to more potential programs) appear near the root

- This tree generates a set of programs called a family
D. L. Parnas. *On the design and development of program families.* 1976

- Enhanced method
  - An enhanced method is described for the design of reusable software based on analyzing the common features of the family with step-wise refinement and information hiding.

- Correlation to real-world setting
  - Reuse along the lines of the classical model was happening. The examples chosen were plausible programs.

- Hypothesis: stepwise refinement is always profitable. Module specifications can be profitable if family members will vary widely.

- Proof: feasibility demonstrated by examples. Profitability argued from the definition of specifications: they are so clear that the programmer doesn’t need to look at most of the code

- Do I believe it? Creating and maintaining specifications like this is extremely expensive, so stepwise refinement can be unprofitable
These ideas contributed to approaches...

- 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

- Frameworks aren’t mentioned
... that didn’t contain software costs

- Project size was increasing

- A line of code (LOC) from a large project is more expensive than a LOC from a small project (Lewis, 1995)

- Reuse libraries didn’t solve the problem
  - Finding the pieces takes time
  - Programmers would rather code (Constantine, 2001)
    - Especially since they frequently needed to write code to glue the pieces together
GUIs exacerbated the crisis

• GUIs add bulk to existing applications
  
  – As much of 60% of the time and code to develop a GUI application is devoted to the user interface (Myers, 1992)
  
  – This ratio is much lower for applications with non-GUI interfaces

• OS vendors (like Apple) needed 3rd party developers

• “Developing software for the Macintosh is a bitch.” (Dash Chang, President of Chang Labs)
Industrial frameworks adapted ideas

• From software engineering
  – Reuse of code
  – Reuse of design
  – Patterns

• From object orientation programming
  – Most research produced pure OO languages like Smalltalk
    – Industry retrofitted objects into existing programming languages (Clascal, Object Pascal, Objective C, C++)

• And from Smalltalk itself
  – There are few publications from Xerox, this is guesswork
    – Had a concept of “teams” for things like model-view-controller
    – The Smalltalk programming environment didn’t make the boundary between the teams and application clear
Frameworks became a reuse strategy

In 1988, Johnson and Foote:
- Describe a generic concept of frameworks
  - A later paper has the title frameworks = (components + patterns)

- Create a simple taxonomy of frameworks
  - White box: the programmer must understand the implementation details of a framework to use it
  - Black box: the programmer only needs to understand an interface
    - These tend to have inversion of control

- Describe a development methodology to create new frameworks
The first frameworks didn’t work well

- Apple developed a framework for the Lisa in 1983

- Very few developers outside of Apple used it
  - The market for $10,000 machines was small

  - The framework was the only reuse mechanism
    - The framework was complicated by the need to support corner cases

  - The application needed to know a lot about the framework internals (white box)

  - A somewhat complication OO language called Clascal

- The Mac shipped without a framework in 1984

(Rosson, 1996)
The second framework was better

- MacApp was released in 1986

- It benefited from
  - Prior experience with the Lisa
  - Knowledge gleaned by Apple during the development of several applications
  - The presence of a reuse library (the Toolbox) to handle the corner cases
  - A simplified hybrid OO language (Object Pascal)

- The developers were happier
  - “I was able to build a working prototype of my entire application in less than five months. This would never have completed this without MacApp.”
Leading to more ambitious proposals


- Unvalidated enhanced method (a.k.a. whitepaper)
  - An enhanced method is proposed for reducing the cost of extending product lines based on a combination of development tools, standard components, frameworks, and models.

- Correlation to real-world setting
  - Reuse is still a problem, but it has been solved in other industries. This is easier to see if we focus on economies of scope (mass design), not scale (mass production).

- Hypothesis: the upfront costs of this method can be contained with improved abstraction techniques and CASE tools. More specialized factories will be more valuable.

- Proof: persuasion only

- Do I believe it? I’m least convinced by the argument about costs. Most of the other points here look convincing.
More information could improve decisions

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

- The earliest framework-specific study seems to be Moser, in 1996.
  - He compares results across 30 industrial framework projects.
  - This may explain the 8 year gap.

- I’m planning to look at empirical methods of determining programmer productivity for my research method.