“Cloning Considered Harmful”
Considered Harmful

Cory Kapser and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
A Commonly Cited Belief

Cloning considered harmful

- Claims:
  - Eliminate duplication reduces maintenance cost
  - Extensions will take less time
Introduction

- What is a code clone?
  - A lot of definitions (Daugstthul)
    - Redundant code
    - Duplicated code (copy and paste)
    - Similar code
  - Current literature cites that on average 10-15% of code is similar (duplicated)
- Why do these clones exist?
The Negative Effects of Cloning

- It increases maintenance costs
  - Bugs can be duplicated or even introduced
  - Unnecessary code bloat
  - Understanding the differences in clones can be difficult
- Can be indications of “smelly” parts of your code
  - Duplication of complex code
  - Poor design
  - Poor interfaces require repetitive code
But is it all bad?

- Code duplication used to minimize risk in financial software [Cordy]
- Developers often use duplicating as a starting point for new code [Kim et al]
- Duplication can be a useful architectural artifact
Reasons to Duplicate

- Clones reduce risk exposure
- Feature “springboard”- evolution intended to diverge
- There may be access restraints
- Abstractions can make code complex
- Abstractions can introduce unwanted architectural dependencies
Patterns of Cloning

- Typical ways in which duplication of code is used in software
- Based on several case studies
  - Linux, Apache, Postgresql, Columba, Gnumeric
- Defined by what is duplicated and why
- Why patterns?
  - Patterns create a framework of documentation
  - Lead to the crystallization of vocabulary
  - Initial steps toward formal definitions of real life phenomenon
  - Ideally leading to automatic detection and classification
Patterns of Cloning

- **Categories**
  - What
  - How
  - Management

- **Patterns**
  - Name
  - Motivation
  - Advantages
  - Disadvantages
  - Management
  - Long term issues
  - Structural manifestations
  - Examples
Forking

• Forking
  - What
    • Portions of code that are intended to evolve independently
    • Duplication is a “springboard” for the new code
  - When
    • Commonalities and differences of end solutions are not clear
  - Management
    • When code matures, refactoring may be possible

• Examples
  - Hardware variations
  - Platform variations
  - Experimental variations
Forking - Hardware Variations

- Name:
  - Hardware Variations
- Motivation:
  - Similar hardware family exists
  - Often non-trivial differences in the functionality/features
  - Difficult and risky to modify the existing code while preserving compatibility for the original target
Forking - Hardware Variations

- **Advantages:**
  - Avoid retesting the driver on older hardware devices
- **Disadvantages:**
  - Propagation of bug fixes
  - Introduce unexpected feature interactions
  - Code growth
Forking - Hardware Variations

- **Management:**
  - Groups of cloned drivers should be clearly identified
  - Bug fixes should be investigated within the group
- **Long term issues:**
  - Dead code can slowly creep into the system
- **Structural manifestations:**
  - Developers usually copy the entire file and modify
Forking - Hardware Variations

- Examples:
  - NCR5380.c -> atari_NCR5380.c -> sun3_NCR5380.c
  - Documentation shows the trail
Customization

- **What?**
  - Code solves a similar problem but additional requirements exist

- **Why?**
  - Current code cannot be modified to encompass these concerns
    - Ex. code ownership, exposure to risk
  - Abstractions may be overly complicated

- **Management**
  - Form proper abstractions and remove if possible

- **Examples**
  - Bug workarounds
  - Replicate and specialize
Templating

- **Templating**
  - **What?**
    - Code embodying the desired behavior already exists
    - Parameterization
  - **Why?**
    - Language constraints prevent appropriate abstraction
  - **Management?**
    - Evolution is expected to be closely related, Linked Editing should be used
    - Machine generated code?
  - **Examples**
    - Boiler-plating due to language inexpressiveness
    - API/Library Protocols
    - General language or algorithmic idioms
Conclusion

Cloning considered harmful
Conclusion

“Cloning considered harmful” considered harmful
Conclusions

- Duplicating code can have positive effects on development
- Reporting metrics is simply not enough
- Management of clones is dependent on the pattern
A Patterns Wiki

- When have introduced a structure and a language, this is only the beginning
  - Need feedback
  - Community involvement
  - A wikipedia page?
Clarifications?
Conclusions

- Duplicating code can have positive effects on development
  - Facilitates quick development of new features
  - Reduces risk exposure
  - Decouples features/modules in the system
  - Sometimes the only alternative

- Management of clones is dependent on the pattern
  - Synchronous editing, refactoring, selective patching, simple programmer awareness
  - Refactoring is not always appropriate, but may be in time
Customization – Replicate and specialize

- Name: Replicate and specialize
- Motivation:
  - As developers implement solutions, they may find code in the software system that solves a similar problem to the one they are solving. However, this code may not be the exact solution, and modifications may be required. While the developer could generalize the original code, this may have a high cost in testing and refactoring in the short term. Code cloning may appear to be a more attractive alternative, and is commonly used in practice to minimize costs associated with risk.
Customization – Replicate and specialize

- **Advantages:**
  - Reduces immediate costs in testing and refactoring. Additionally, the high cognitive cost of developing the abstraction is avoided [29].
- **Disadvantages:**
  - Long term costs of finding and maintaining these duplicates could out-weigh the short term gains.
Customization – Replicate and specialize

- Management:
  - If an appropriate abstraction can be made, deprecating the original code and transitioning to the abstraction may defer testing costs and protect system stability. If the appropriate abstractions can not be made, explicitly linking the code clones through documentation or tool support will ensure consistent maintenance. Long term issues. Duplicated code can over time become more entrenched, with more of the software system dependent upon it. Over time, the cost of refactoring the code may rise. Differences in the code may make locating duplicates difficult, making maintenance of clones more costly.
Customization – Replicate and specialize

- Structural manifestations:
  - These code clones are often snippets or procedures located near each other, but can be more widely distributed as well. In some cases these clones can be particularly hard to detect due to the changes that have been made. Often the copied code contains control structures, suggesting that developers use duplication to reuse complex logic, an observation also noted by Kim et al. [26].
Customization – Replicate and specialize

- Examples:
  - This pattern is the most common type of cloning. In one example in Gnumeric, we see this pattern in use for developing the procedures that build the locale and character encoding selection menus. The procedures can be found in the files src/widgets/widget-charmap-selector.c and src/widgets/widget-locale-selector.c. The control flow of both procedures is very similar. However, how the items are chosen to be added to the menu differs, causing a minor change and addition of several lines. Another small difference is the way in which the menu title is made near the end of the procedure. In addition to these customizations, the data type containing the list of entities is also different, performed as a parametric change.
Forking - Hardware Variations

- Name:
  - Hardware Variations
- Motivation:
  - When creating a new driver for a hardware family, a similar hardware family may already have an existing driver. However, there are often non-trivial differences in the functionality/features between families of hardware, making it difficult and risky to modify the existing code while preserving compatibility for the original target.
Forking - Hardware Variations

- **Advantages:**
  - The risk of changing the existing driver is especially high in this situation as testing the driver on older hardware devices can be difficult and time consuming. Cloning the existing driver prevents the need for this type of testing.

- **Disadvantages:**
  - In addition to the general maintenance issues such as propagating bug fixes, cloned drivers may introduce unexpected feature interactions, in particular in the realm of resource management. Code growth can be a particular issue with this pattern of cloning because entire files or subsystems are copied.
Forking - Hardware Variations

- **Management:**
  - Groups of cloned drivers should be clearly identified, and potential bug fixes should be investigated within the group.

- **Long term issues:**
  - Dead code can slowly creep into the system unless care is taken to monitor which drivers are still actively supported.

- **Structural manifestations:**
  - Drivers are commonly packaged into a single file for simplicity of use within the system. Developers usually copy the entire file, and the duplicate is then modified to match the new device.
Forking - Hardware Variations

- Examples:
  - NCR5380.c -> atari_NCR5380.c -> sun3_NCR5380.c
  - Documentation shows the trail