Managing Known Clones: Issues and Open Questions

Kostas Kontogiannis[1]

University of Waterloo, Department of Electrical & Computer Engineering, Waterloo, Ontario, N2L 3G1, Canada
kostas@swen.uwaterloo.ca

Abstract. Many software systems contained cloned code, i.e., segments of code that are highly similar to each other, typically because one has been copied from the other, and then possibly modified. In some contexts, clones are of interest because they are targets for refactoring. This paper summarizes the results of a working session in which the problems of merely managing clones that are already known to exist. Six key issues in the space are briefly reviewed, and open questions raised in the working session are listed.

Keywords. code clone, software evolution, change management, code visualization, redundancy, clone indent, metamodels, software management environments

1 Introduction

This paper reports on the proceedings of the breakout session within DRASIS [1] on the management of clones within software systems. The breakout session discussed several topics, including:

- Costs and benefits of redundancy and cloning.
- Strategies for clone removal.
- Taxonomy of intent and context of cloning.
- Clone metamodel and denotation of clones.
- Clone management environments.

Discussions regarding each of the above topics are summarized in the following sections, and current concepts and issues are relayed. Open research questions perceived and raised in the session are listed; these show a need that must be addressed in future theoretical and empirical research.

2 Costs & Benefits of Redundancy & Cloning

Cloning is a standard programming technique for assisting forward engineering. As such, it is a legitimate code authoring technique. Techniques that would avoid the creation of a clone beforehand are often too expensive to apply.

Dagstuhl Seminar Proceedings 06301
Duplication, Redundancy, and Similarity in Software
http://drops.dagstuhl.de/opus/volltexte/2007/967

Cloning may occur on the code level only, or manifest on the code level as a result of design-level "cloning." To ensure the coherence of the implementation, similar constructs on the design level should be implemented similarly, i.e. a design template is instantiated multiple times, which leads to code clones. Redundant or cloned code may appear as a result of specification-based generative programming. In such cases, the relationships between clones are known and relate to the generative programming and the specification in the generative language or environment. Thus there may be a relation between generative languages and the modeling of clone relationships and invariants.

Clones introduce implicit dependencies in the code, i.e. the same behaviour is located at multiple places in the code. In the further evolution of the code, this must be addressed to ensure convergent evolution of the code. If changes are applied to one clone instance only, divergent evolution occurs in contrast. This said, certain clones may result because copies are created of well tested code, a fact which may lead to the the concept of "trusted clones."

2.1 Open Questions

Costs.1: Under which conditions is the existence of clones an indicator for good or bad quality?

Costs.2: Can aspect-oriented programming be used for clone management ("Aspect-oriented clone management")?

Costs.3: How can *intentional* cloning be supported by tools?

3 Strategies for Clone Removal

The breakout group mainly discussed refactoring operations as a structured approach to removing clones. Relevant refactoring operations include the extraction of code blocks into methods, functions, procedures resp. methods. For more complex clones, higher order function templates might be used, which could be implemented through the Strategy pattern. It might not be good practice to eliminate all clones, e.g. for performance reasons.

3.1 Open Questions

Strategies.1: Which effects do different removal strategies have on the quality of the resulting system, e.g. time and resource efficiency, understandability?

4 Taxonomy of Intent and Context of Cloning

A priori clone avoidance is often noted as a strategy to ensure code quality and to remove the need to apply clone removal. Consequent use of aspects could be a means to achieve clone avoidance. However, the breakout group believes that it is more useful to understand the intent and context of the introduction of clones.

4.1 Intentions

The following lists intentions (or rationales) for creating clones were raised during discussion:

- To achieve a form of organized reuse (convergent or divergent).
- Plain copy and paste.
- Design reuse—what constitutes reuse at the design level (patterns)?
- Logic or functionality reuse.
- Meet some functional and/or nonfunctional requirement.
- Facilitate understanding—use of idioms—patterns.
- To overcome language limitations.
- General convenience.
- Easier to copy and adapt than to generalize.
- No permission to generalize.
- Reduce risk.
- To reduce explicit coupling (however, cloning introduces an implicit form of coupling instead).
- Experimental branch and merge (copy—enhance—experiment—test—integrate cycle).
- Adapt core functionality of a component (e.g. driver) to new "similar" but different environment.

4.2 Contexts

Understanding the context for cloning were considered next. The contexts are important to know because they relate to or establish:

- the relations between clones and features of the clones,
- the conditions under which a clone can be reused,
- the Pre/post conditions when instantiating a clone in a new environment, and
- the information on the impact clones have on non-functional and functional requirements.

Discussion turned to the contexts in which clones are created. The aspects of such contexts were enumerated in the group as follows:

- Location within the overall program code.
- Internal structure.
- Author information.
- Intention of its creation.

- Information on differences between instances of a clone group (syntactic / semantic).
  - Provides consistency constraints and invariants that have to hold in the presence of clone evolution on one clone group
- Classification.

Requirements for modeling such context and intention:

- Schema must be portable and usable in various tools (IDEs).
- Schema must be able to represent clone relations that are not equivalence relations.
- Provide information on the intended lifecycle of the clone.
- Provide information on neighboring clones, overlapping clones, nested clones.

4.3 Open Questions

Context.1: Under which conditions is cloning used as the solution to some programming situation?

Context.2: In particular, why is copy and paste used instead some other language based mechanism?

Context.3: Which intentions create how many clones?

5 Metamodel and Denotation of Clones

The group discussed further problems of modeling clones and clone relationships. The concerns included: formalisms and techniques for modeling clones, clone relationships, and relationships to intent; Tool support; and markup languages.

6 Clone Management Environments

One part of the discussion focussed on operational primitives that represent use cases that should be supported by clone management environments. The discussion generated a list of such operational primitives as a suitable starting point for management environment requirements. These included:

- create clone
  - cut and paste
  - auto-complete
  - instantiate from template
- search candidate clones
  - list all possible clone groups
  - identify clone based on query

- query – exact
- query – template or specification type
- explain the intent for a clone,
- identify impact of a clone,
- remove clone
  - generalize (abstract parameterize)
  - componentize
  - merge fragments that are similar to a "service"

Additional requirements:

- version control system for clones
- maintaining consistency between clone classes—synchronizing clones
- maintaining statistics and providing links to version control system

Clone management environments need to access functionality that has been implemented in various existing tools, most importantly clone detectors. These existing tools embody different allocations of functionality, which makes it difficult to integrate them into environments that enable the use of multiple alternative techniques, e.g. for clone detection or removal. A reference architecture that identifies the main components and allocations of functionality to these components would enable the independent development of techniques that are easier to integrate. Furthermore, more detailed technical standard interface descriptions for the interoperation of such components would further ease interoperation and reuse. A reference implementation of a clone management environment, e.g. based on Eclipse, could be used within empirical research on clones.

6.1 Open Questions

Environments.1: Which components and interfaces should be described by a reference architecture for clone management environments?

References

1. Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, Dagstuhl, Germany, Dagstuhl (2006) ISSN 1682–4405.