A Controlled Experiment on Inheritance Depth as a Cost Factor in Maintenance

Walter F. Tichy
University of Karlsruhe

Why is Inheritance Important?

- Inheritance and polymorphism is an essential part of OO-modeling
- Class hierarchies can structure a domain cleanly
- Reuse by extension is convenient
- How would you do certain designs without inheritance, such as template method, compositum, strategy, factory method, extensible frameworks, and others?

Problems with Inheritance

- Everybody has been lost in class libraries. Sun’s SWING and AWT library consists of over 800 classes; Java 1.4 contains over 2700 classes.
- OO design guidelines recommend using inheritance sparingly (choose classes first, structure class hierarchy later, prefer composition over inheritance, don’t use subclassing unnecessarily, inherit only interfaces, not implementations).
- Design pattern books use single-level inheritance only, inheriting mostly interfaces.

Inheritance Considered Harmful

- Difficulty of testing increases with inheritance depth (Basili et al, 1996).
- Sommerville, 1992: “…class inheritance is not essential and may sometimes confuse a design, because an object class cannot be understood on its own without reference to any superclass.”
- Wilde et al, 1993: “Understanding of a single line may require tracing a chain of method invocations through several different object classes and up and down the object hierarchy to find where the work is really getting done.”
- Soloway’s delocalized plans: Design decisions spread out over several locations in a program.
What do we make of this?

• Should we go back to programming with abstract data types without inheritance, only allow interface inheritance?

• But what about the potential benefits of inheritance:
  • clarity of conceptual modeling
  • design patterns (e.g. compositum, template method)
  • sharing of code
  • convenient extension
  • frameworks
  • Can you program GUIs reasonably without inheritance?

• Is there a serious problem lurking here?

Enter Experiments

• Experiment by Daly et al, 1996:
  • Task: adding new class to existing program
  • Two different programs:
    • program A: 3 vs 0 levels
    • program B: 5 vs 0 levels
  • 0-level programs were “flatteened” versions of 3 and 5 level programs.

Levels: an n-level program is one where the maximum number of edges traversed from root to leaves is n.

Results:

• Maintaining 5 levels is slower than 0 levels.

• Maintaining 0 levels is slower than 3 levels.

• Summary: 3 levels is best!

But wait!
3 levels seems arbitrary.
Can this be explained?
We’ve all programmed with way more than three levels (e.g., C++, Java, GUIs) and did not necessarily see difficulties because of inheritance.

Are there Replications?

• Cartwright, 1998: replicated 3 vs 0 levels with 5-person teams (undergraduates).
  • Result: 0 levels is faster than 3!

• Harrison, 1999: identify changes (undergraduates):
  • Result: 0 levels causes fewer mistakes than 3 and 5!

Something is wrong here, but what?

Experiment flawed?
Subjects inexperienced?
Program/task too simple?
Flattening artificial?
Random fluctuations?
The Karlsruhe Inheritance Depth Experiments

- **Hypothesis 1**: Given two functionally identical versions of a program that differ in inheritance depth, the version with deeper inheritance requires more time for maintenance.

- **Hypothesis 2**: Given two functionally identical versions of a program that differ in inheritance depth, the version with deeper inheritance causes lower quality during maintenance.

Major Changes in Design of Experiments

- Longer programs: Karlsruhe programs were 2 to 4 times longer than the prior programs (more realistic).
- More complicated and varied change tasks, more understanding of the program necessary (prior experiments only required the addition of a single class, similar to an existing one).
- Subjects were given source code plus class diagram (prior experiments provided only source code, no class diagram).
- Subjects decided on their own when finished (in prior experiments, instructor checked programs and asked subjects to continue when flaws were present. This is unrealistic in work environments.)

Experiment Details

- **Experiment 1 (Group G)**: 57 Graduate Computer Science Students out of a class of 70, selected if they achieved 75% of available points in Java course.
  - Had 8.1 years of programming experience on average.
  - Performed Task 1 and then Task 2a/2b

- **Experiment 2 (Group U)**: 58 Undergraduate Computer Science Students out of a class of 160.
  - Had 6.1 years of programming experience.
  - Performed Task 2a and then Task 2b

More Experiment Details

- Single program in 5-, 3-, and 0-level versions.
  - Original program had 5 levels, 28 classes, 80 method bodies, 1200 lines.
  - Equivalent 3 level program: 27 classes, 100 method bodies, 1340 lines (inherit only interfaces, never implementations, “design pattern style”)
  - Equivalent 0-level program: 20 classes, 160 method bodies, 2470 lines (expanded all inherited attributes, “C-language style”)
- The 3- and 0-level programs were reasonable design alternatives.
Program: Börse
Displays stock exchange data in various ways

More Details
- Materials: source, OMT diagram.
- Subjects decide when to hand in.
- Independent variable: inheritance depth (0, 3, 5).
- Subject ability was balanced according to score on previous assignment or largest program they had written.
- Dependent variables:
  - elapsed time (minutes),
  - correctness (discrete grading scale)

Class Hierarchy of Börse (5 levels)

Tasks
- Task 1: Y2K, i.e. convert program from 2-digit to 4-digit years.
  - Solution involved finding all occurrences of a method.
  - 5-level program had half the change locations as the others.
- Task 2a: add new type of display and extend menu.
  - Inherit (3, 5-level) or copy (0-level) part of solution.
  - adapt code from a chart class.
  - Overwrite behavior from superclass (5-level hardest).
- Task 2b: add another type of display.
  - solution can profit from 2a, but 5-level program is the simplest to handle (clone a class).
**Procedures**

- Subjects implemented their solutions using JDK 1.1 and tested them.
- Four parts:
  - questionnaire and pretest for Java’s inheritance rules
  - assignment 1
  - assignment 2
  - postmortem questionnaire
- Measured time between handing out assignment and handing in solution.
- Also intercepted each compilation run and time stamped.
- Grading on a fixed classification of error types (points).

**Results**

Inconclusive for hypothesis 1.

Modest support for hypotheses 2.

- 0-level program requires more modifications than 5-level, but is almost as fast to do as 5-level program (difference $p=0.2$).
- 3- and 5-level program have about the same number of mods, but 3-level slower to modify ($p=0.075$).
- 0-level programs received most points.

0-level significantly faster to do than the others.
3-level and 5-level about the same.
Correctness about equal in all three.
Most code to be added in level 0, but it is apparently easier to copy and modify than to trace up and down the inheritance hierarchy.

Weak support for both hypotheses.

0-level and 3-level programs about the same, faster to modify than 5-level.
Correctness decreases with greater inheritance depth.
Time difference between 0-level and 5-level is significant ($p=0.038$).
Results

5-level program significantly faster to change than the others (p<0.01).
3-level and 0-level were about the same (p=0.26).
No significant differences in correctness.
However, task 2b is exceptionally simple for 5-level program.

Results Task 1(Y2K), Group G

- 0-level program requires more mods than 5-level, but is almost as fast to do as 5-level program (difference p=0.2).
- 3- and 5-level program have about the same number of mods, but 3-level slower to modify (p=0.075).
- 0-level programs received most points.

Inconclusive for hypothesis 1.
Modest support for hypotheses 2.

Results Task 2a+2b, Group G

- 0-level significantly faster to do than the others.
- 3-level and 5-level about the same.
- Correctness about equal in all three.
- Most code to be added in level 0, but it is apparently easier to copy and modify than to trace up and down the inheritance hierarchy.

Modest support for hypothesis 1.
Inconclusive for hypothesis 2.

Results Task 2a, Group U

- 0-level and 3-level programs about the same, faster to modify than 5-level.
- Correctness decreases with greater inheritance depth.
- Time difference between 0-level and 5-level is significant (p=0.038).

Weak support for both hypotheses.
Task 2b, Group U

- 5-level program significantly faster to change than the others (p=0.006, 0.004).
- 3-level and 0-level were about the same (p=0.26).
- No significant differences in correctness.
- However, task 2b is exceptionally simple for 5-level program: clone an entire existing class, no actual method changes.

Result Summary

- Modest Support for the hypothesis that 0-level is faster and better to maintain than 3-level or 5-level programs.
- Stark contrast to experiment by Daly.
- In line with experiments by Cartwright.
- If correct, should we flatten all OO-programs?

More useful question: Is there a better explanation than using inheritance depth?

What other Variables affect Maintenance Time?

- Figure out what needs to be done for a “default” solution strategy for each task.
- Test a number of potential predictors, such as classes, method bodies, lines, investigated methods, hierarchy changes, solution methods; build linear regression model.
- Best predictor in linear regression model:
  - the number of methods investigated in default solution strategy for each task.
  - Explains 84% of variation
- This is true when using ALL data (Daly’s, Cartwright’s, and our own) for the “add class” tasks: 16 groups.

Several Models

- \( T = 6.9m + 28 \) [min], \( R^2=0.84 \)
  - \( m \) is number of methods to be understood
- \( T = 4.5h + 61 \) [min], \( R^2=0.55 \)
  - \( h \) is number of hierarchy changes
- \( T = 7.8d + 70 \) [min], \( R^2=0.10 \)
  - \( d \) is the inheritance depth

R\(^2\) is the variance in average group work time explained by the model. \( R^2 =1 \) means perfect linear correlation; \( R^2 =0 \) means there is none.

Clearly, inheritance depth is a poor predictor!
Two Better Models

- \( T = 8.9m_{\text{exp=L}} + 5.9m_{\text{exp=H}} + 24, \quad R^2=0.94 \)
- \( T = 9.3m_{\text{exp=L}} + 6.3m_{\text{exp=H}} - 3.3d + 27, \quad R^2=0.96 \)

*takes experience into account (Low, High)*

Only after methods examined and experience level are factored into the model, does inheritance depth per se make a small contribution.

Conclusions: There is no Simple Answer

- In experiments, increased inheritance depth slightly increased maintenance time and errors.
- Positive effects of inheritance (less code duplication, less consistency checking) is perhaps visible in larger programs with complex modifications.
- Methods to be examined is a much better predictor for maintenance effort than inheritance depth.

Inheritance depth is not in itself an important factor for maintenance effort.

The real cost drivers may be number of methods analyzed, prior knowledge of the program, the degree of delocalization of information, or other aspects.
Threats to Validity

Internal Validity:
• The conversion to level-3 and level-0 programs may have introduced differences unrelated to inheritance depth (unlikely, since straightforward conversion).

External Validity/Generalizability:
• Students have less experience than professionals. Professionals may do better. However, not length of experience, but experience with application is important.
• The maintenance tasks may not be typical.
• The programs may not be typical (real programs are larger, may have different structural complexity, may have poorer documentation.)