3. Compile-Time Variability with Clone-and-Own – Handout

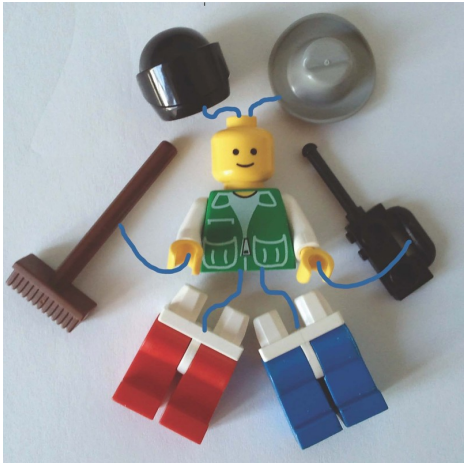
3. Compile-Time Variability with Clone-and-Own

3a. Compile-Time Variability and Clone-and-Own

3b. Clone-and-Own with Version Control

3c. Clone-and-Own with Build Systems

Recap: How to Implement Software Product Lines?



Key Issues

- Systematic reuse of implementation artifacts
- Explicit handling of variability

Variability [Apel et al. 2013, p. 48]

"**Variability** is the ability to derive different products from a common set of artifacts."

Variability-Intensive System

Any software product line is a variability-intensive system.

Recap: Variability and Binding Times

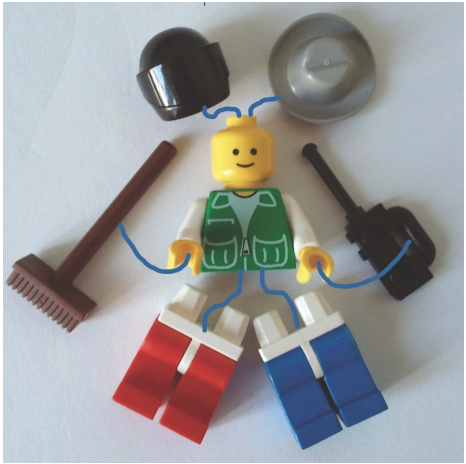
Binding Time [Apel et al. 2013, p. 48]

- Variability offers choices
- Derivation of a product requires to make decisions (aka. binding)
- Decisions may be bound at different binding times

When? By whom? How?

Lecture 2a: **when** and **by whom**

Lecture 2b: **how**



Recap: Runtime Variability

Basic Principles

Variability with Configuration Options:

- Conditional statements controlled by configuration options
- Global variables vs. method parameters

Object-Orientation and Design Patterns:

- Template Method
- Abstract Factory
- Decorator

Problems of Runtime Variability

Conditional Statements:

- Code scattering, tangling, and replication

Design Patterns for Variability:

- Trade-offs and potential negative side effects
- Constraints that may restrict their usage

In General:

- Variable parts are always delivered
- Not well-suited for compile-time binding

Compile-Time Variability

Problems of Runtime Variability

Conditional Statements:

- Code scattering, tangling, and replication

Design Patterns for Variability:

- Trade-offs and potential negative side effects
- Constraints that may restrict their usage

In General:

- Variable parts are always delivered
- Not well-suited for compile-time binding

Compile-Time Variability [Apel et al. 2013, p. 49]

"Compile-time variability is decided before or at compile time."

Goals:

- Only required source code is compiled
- Smaller and highly optimized variants

Challenge:

- How to implement options and alternatives (i.e., variability)?

In this Lecture

Simple concepts and techniques for a few variants

Ad-Hoc Clone-and-Own

Clone-and-Own [Northrop et al. 2012, p. 7]

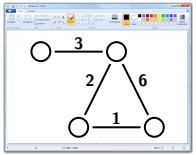
"Suppose you are developing a new system that seems very similar to one you have built before. You borrow what you can from your previous effort, modify it as necessary, add whatever it takes, and field the product, which then assumes its own maintenance trajectory separate from that of the first product. What you have done is what is called **clone and own**. You certainly have taken economic advantage of previous work; you have reused a part of another system. But now you have two entirely different systems, not two systems built from the same base. This is again **ad hoc reuse**."

Cloning Whole Products

Clone-and-Own

- New variants of a software system are created by copying and adapting an existing variant
- Afterwards, cloned variants evolve independently of each other

Example for Ad-Hoc Clone-and-Own




```
class Graph {
  List nodes = new ArrayList();
  List edges = new ArrayList();

  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    e.weight = new Weight();
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  void print() {
    for (int i = 0; i < edges.size(); i++) {
      ((Edge) edges.get(i)).print();
    }
  }
}
```

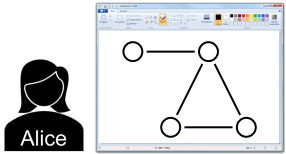
initial graph implementation providing weighted graphs

```
class Edge {
  Node a, b;
  Weight weight = new Weight();

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Weight {
  void print() {...}
}
```

```
public class Node {
  int id = 0;

  void print() {
    System.out.print(id);
  }
}
```

Alice's Clone: Unweighted Graphs




```
class Graph {
  List nodes = new ArrayList();
  List edges = new ArrayList();

  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    e.weight = new Weight();
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    Edge e = new Edge(n, m);
    e.weight = w;
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  void print() {
    for (int i = 0; i < edges.size(); i++) {
      ((Edge) edges.get(i)).print();
    }
  }
}
```

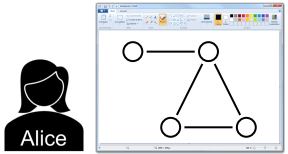
Alice works with unweighted graphs: she copies and adapts the basic implementation

```
class Edge {
  Node a, b;
  Weight weight = new Weight();

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Weight {
  void print() {...}
}
```

```
public class Node {
  int id = 0;

  void print() {
    System.out.print(id);
  }
}
```

Alice's Clone: Unweighted Graphs



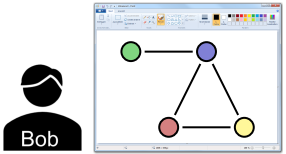

```
class Graph {
  List nodes = new ArrayList();
  List edges = new ArrayList();

  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  void print() {
    for (int i = 0; i < edges.size(); i++) {
      ((Edge) edges.get(i)).print();
    }
  }
}
```

Alice works with unweighted graphs: she copies and adapts the basic implementation

```
class Edge {
  Node a, b;

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
public class Node {
  int id = 0;

  void print() {
    System.out.print(id);
  }
}
```

Bob's Clone: Colored Graphs



Bob works with colored graphs: he is a colleague of Alice and knows her variant, so he copies and adapts Alice's variant

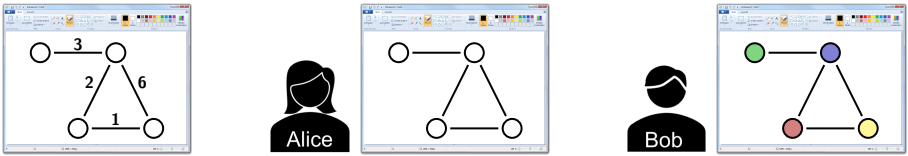
```
public class Graph {
  List nodes = new ArrayList();
  List edges = new ArrayList();

  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
  void print() {
    for (int i = 0; i < edges.size(); i++) {
      ((Edge) edges.get(i)).print();
    }
  }
}
```

```
class Edge {
  Node a, b;

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

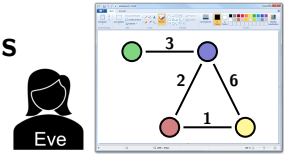
```
public class Node {
  int id = 0;
  Color color = new Color();

  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
public class Color {
  static void setDisplayColor(
          Color c) {...}
}
```

Why is Clone-and-Own Problematic?

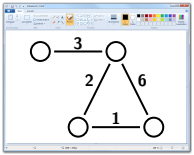


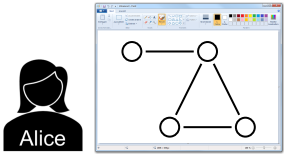
Clone-and-Own Problems: Feature Combinations

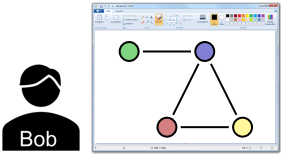


Eve has a new requirement: she wants to work with graphs which are both colored and weighted

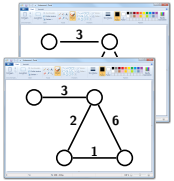
- Where to start from?
- Does Eve know about Bob's and Alice's variants?
- If so, how to avoid repeating the work that has been already done by Alice and Bob, respectively?







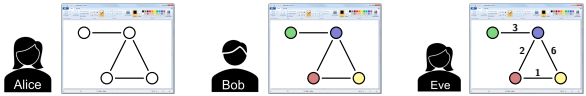
Clone-and-Own Problems: Evolution & Maintenance



Maintainers of the initial variant refactor the code of the basic implementation

```
public class Graph {
  ...
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    Edge e = add(n, m);
    e.weight = w;
    return e;
  }
  ...
}
```

- Who informs Alice, Bob and Eve about the improvement?
- How do they know whether the improvement is relevant for them?
- If so, how to propagate the improvement to their variant?

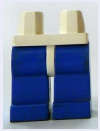


Recap: Software Clones [Lecture 1]

Software Clone [Rattan et al. 2013, p. 1166]

- = result of copying and pasting existing fragments of the software
- code clones = copied code fragments
- replicates need to be altered consistently
- for example: bugs need to be fixed in all replicated fragments
- in practice: a common source for inconsistencies and bugs

Cloning Parts of Software




Cloning Whole Products (Clone-and-Own)


Types of Software Clones

Types of Software Clones [Rattan et al. 2013, p. 1167]

- Type 1: identical except whitespaces and comments
- Type 2: syntactically similar (e.g., changed identifiers, . . .)
- Type 3: copied with modifications (e.g., inserted or removed statements)
- Type 4: similar functionality without textual similarities

Relevant Types for Clone-and-Own?

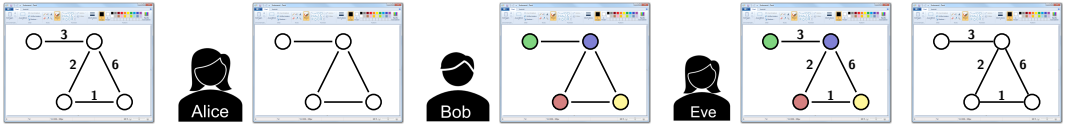
- Type 1: may happen if clones diverge and comments need to reflect actual changes
- Type 2: may happen if clones diverge and identifier names are not appropriate anymore
- Type 3: actually necessary for clone-and-own
- Type 4: may happen if same functionality is implemented again (simply unknown or merge/cherrypick infeasible) [see Lecture 4]

Every difference is an obstacle for future maintenance (cf. merge and cherrypick)

Cloning Parts of Software

Cloning Whole Products (Clone-and-Own)

Discussion of Clone-and-Own



Advantages

- Simple and straightforward approach
- Rapid exploration of new ideas
- No upfront investments

Disadvantages

- No structured and systematic reuse (copy & edit)
- No flexible combination of features
- Maintenance quickly becomes impractical

Towards Managed Clone-and-Own

- How can we better manage such clone-and-own development?
- The traditional answer: Software Configuration Management
- In the sequel: Software Configuration Management in practice

Compile-Time Variability and Clone-and-Own – Summary

Lessons Learned

- Compile-time variability is decided before or at compile time
- In clone-and-own, new variants of a software system are created by copying and adapting an existing variant
- Simple paradigm, but suffering from maintenance problems in the long run

Further Reading

- Apel et al. 2013, Section 3.1.1, pp. 48–49 — brief introduction of compile-time variability
- Rattan et al. 2013, Section 2, pp. 1167–1168 — brief introduction to software clones, their types, reasons, (dis)advantages
- Antkiewicz et al. 2014 — brief introduction to ad-hoc clone-and-own (L0)

Practice

- What are the reasons why clone-and-own is very popular in practice?
- What is the order of magnitude of the number of variants that can be reasonably maintained in clone-and-own?
- Have you ever applied the principle of clone-and-own? If so, where and how?

3. Compile-Time Variability with Clone-and-Own

3a. Compile-Time Variability and Clone-and-Own

3b. Clone-and-Own with Version Control

- Software Configuration Management
- Version Control Systems
- Variability with Version Control
- Discussion
- Summary

3c. Clone-and-Own with Build Systems

Excursus: Software Configuration Management

Software Configuration Management

Policies, processes, and tools for managing evolving software systems:

- **Version control**
- **System building**
- Release management
- Change management
- Collaborative work

No Software Configuration Management

Lecture 3a: Ad-Hoc Clone-and-Own

aka. unmanaged clone-and-own

Version Control

Lecture 3b: Clone-and-Own with Version Control

instance of managed clone-and-own

System Building

Lecture 3c: Clone-and-Own with Build Systems

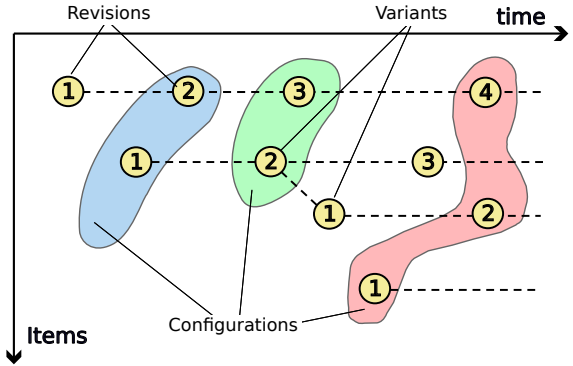
instance of managed clone-and-own

[Apel et al. 2013, pp. 100–101]

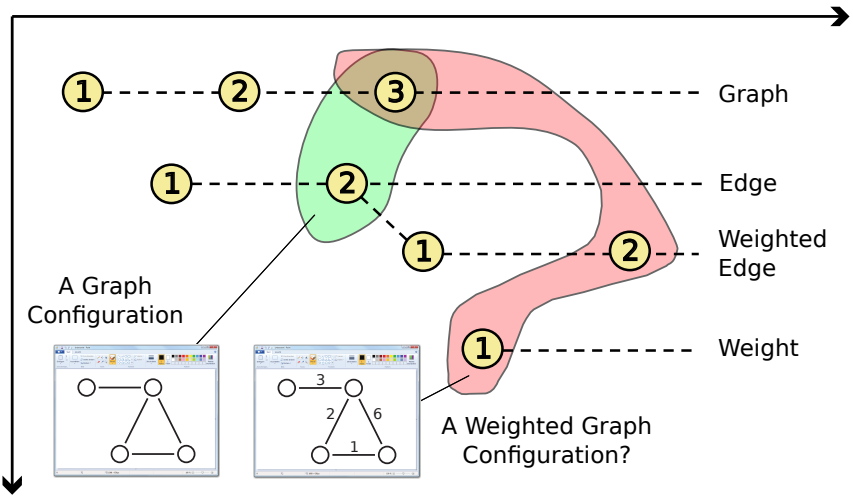
Excursus: Software Configuration Management

Basic Terms and Definitions

- **Software Item**: An (atomic) artifact that can be uniquely identified
- **Version**: A modified software item
  - **Revision**: A new version that replaces an old one
  - **Variant**: A version that co-exists with another one
- **Configuration**: A set of software items that together form a functioning (partial) system
- **Baseline**: A stable configuration that represents a point of reference for further development
- **Release**: A baseline delivered to customers

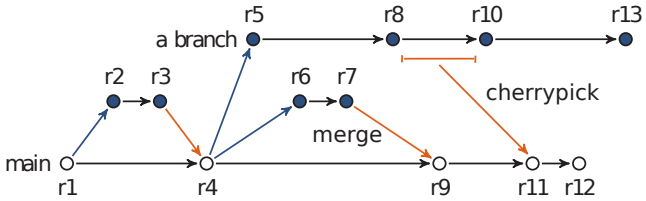


Example: A Conceptual Organization of our Graph Library



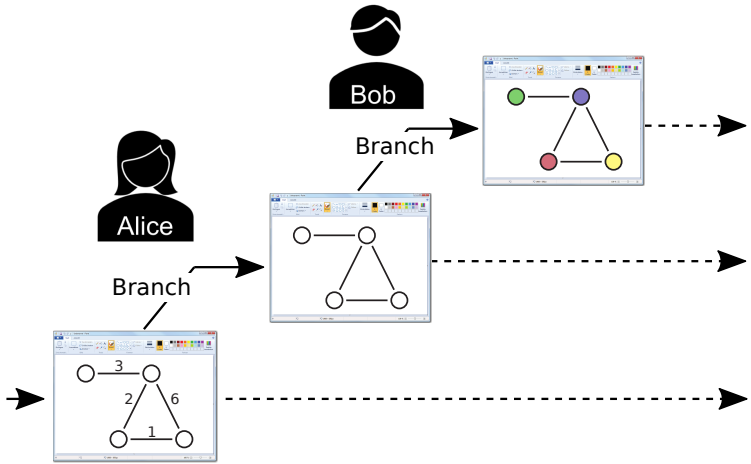
Tool Support: Version Control Systems

$$\text{cherrypick} := \text{patch}(\Delta(r8, r10), r11)$$

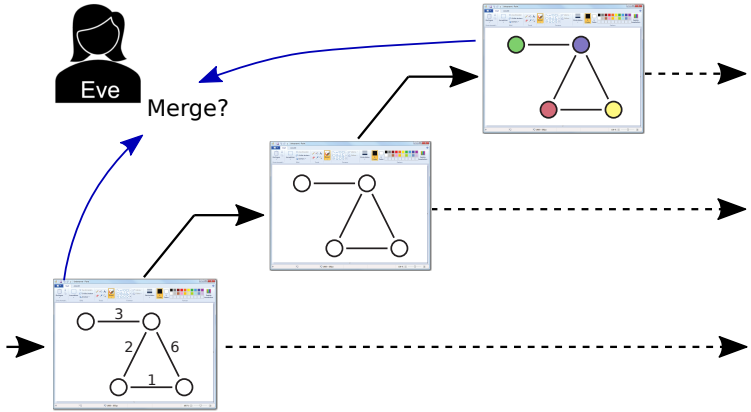


$$\text{merge} := \text{3-way-merge}(r4, \Delta(r4, r7), \Delta(r4, r9))$$

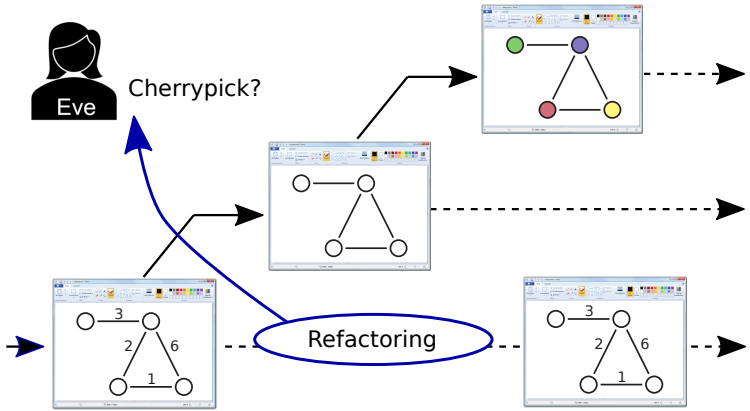
Example: Graph Library under Version Control



Example: Graph Library under Version Control



Example: Graph Library under Version Control



Clone-and-Own with Version Control

Observations

- Aka. **managed clone-and-own** (opposed to ad-hoc clone-and-own)
- Supports keeping track of revisions and variants – aka. **provenance**
- Creation of new variants is partially supported by merging of branches
- Propagation of changes between variants is supported by cherrypicking changes

However:

- Versioning is typically limited to entire system variants (i.e., branches)
- No flexible combination of software items

Advantages [Apel et al. 2013, p. 104]

- Well-established and stable systems
- Well-known known process
- Good tool integration

Disadvantages [Apel et al. 2013, pp. 104–105]

- Development of variants, not features: flexible combination of features not directly possible
- No structured reuse (copy & edit)
- Merging and cherrypicking not fully automated

No Version Control at All?

Revision Control $\subset$ Version Control

"Unless only few small variations are required for few customers, the use of version control systems should be restricted to **revision control**." [Apel et al. 2013, p. 104]

Clone-and-Own with Version Control – Summary

Lessons Learned

- Software configuration management as a traditional discipline of managing the evolution of variability-intensive systems
- Version control systems as a widespread tool supporting clone-and-own in practice

Further Reading

- Apel et al. 2013, Section 5.1, pp. 99–105 — introduction of variability with version control (not explicitly calling it clone-and-own)
- Staples and Hill 2004 — experience report on managed clone-and-own with version control and build systems
- Antkiewicz et al. 2014 — brief introduction to clone-and-own with version control (L1)

Practice

- Which software configuration management concepts are supported by version control systems?
- Do you know other version control systems than Git?
- If so, in which way are they different from Git?

3. Compile-Time Variability with Clone-and-Own

Recap: Software Configuration Management

Software Configuration Management

Policies, processes, and tools for managing evolving software systems:

- **Version control**
- **System building**
- Release management
- Change management
- Collaborative work

No Software Configuration Management

Lecture 3a: Ad-Hoc Clone-and-Own

aka. unmanaged clone-and-own

Version Control

Lecture 3b: Clone-and-Own with Version Control

instance of managed clone-and-own

System Building

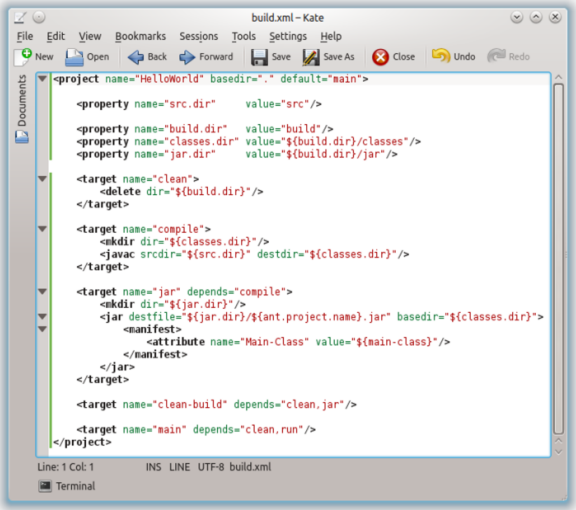
Lecture 3c: Clone-and-Own with Build Systems

instance of managed clone-and-own

Tool Support: Build Systems

Build Systems

- Automation of the build process through build scripts
- Multiple steps with dependencies/conditions
  - Copy files,
  - call compiler,
  - start other tools,
  - . . .
- Tools:
  - make
  - ant
  - maven . . .

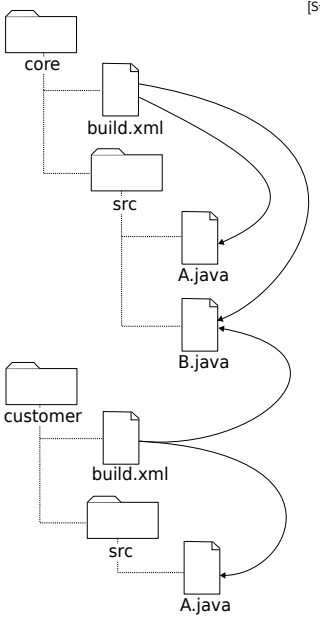


Variability with Build Systems

[Staples and Hill 2004]

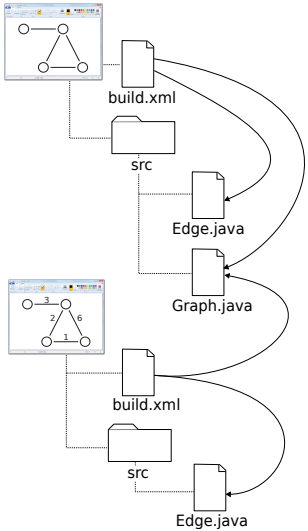
Basic Idea

- One build script per variant
- Include/exclude files when translating
- Overwrite variant-specific files



[Apel et al. 2013, p. 107]

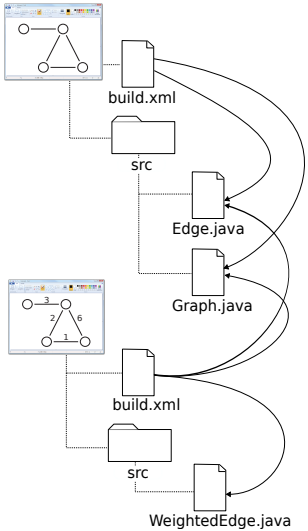
Example: Graph Library




```
class Edge {
  Node a, b;

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Edge {
  Node a, b;
  Weight weight = new Weight();

  Edge(Node a, Node b) {
    this.a = a; this.b = b;
  }
  void print() {
    a.print(); b.print();
    weight.print();
  }
}
```

Example: Graph Library




```
class Graph {
  EdgeFactory edgeFactory;
  ...
  Graph(EdgeFactory edgeFactory) {
    this.edgeFactory = edgeFactory;
  }
  Edge add(Node n, Node m) {
    Edge e = edgeFactory.createEdge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    return e;
  }
}
```

```
class Edge {
  Node a, b;
  ...
}
```

```
class WeightedEdge extends Edge {
  Weight weight = new Weight();
  ...
}
```

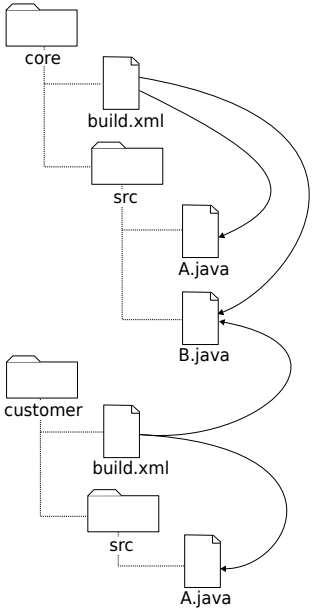
Clone-and-Own with Build Systems

Comparison to Version Control Systems

- Supports combination of more fine-grained software items (i.e., files)
- However: Only limited support for provenance

In General

- Combination of items (i.e., files) $\neq$ combination of features
- Changes to the basic variant may have undesired side-effects
  - Some variants are updated but do not need those changes
  - Some variants are updated but incompatible to those changes
  - Variants with copied files are not automatically updated



Clone-and-Own with Build Systems – Summary

Lessons Learned

- Variability through build scripts
- Granularity of clones: Individual files
- Combination of files
  $\neq$ combination of features

Further Reading

- Apel et al. 2013, Section 5.2.2 and 5.2.4, pp. 106–110 — brief introduction of clone-and-own with build systems (not explicitly calling it clone-and-own)
- Staples and Hill 2004 — experience report on managed clone-and-own with version control and build systems

Practice

- Which software configuration management concepts are supported by build systems?
- What are the commonalities and differences of clone-and-own with version control and clone-and-own with build systems?
- What are the strengths and weaknesses?

FAQ – 3. Compile-Time Variability with Clone-and-Own

Lecture 3a

- What are problems of runtime variability?
- What is compile-time variability?
- What is (ad-hoc) clone-and-own?
- How to develop new features or variants? Exemplify!
- What are (dis)advantages of (ad-hoc) clone-and-own?
- When (not) to use (ad-hoc) clone-and-own?
- What is better runtime or compile-time variability?

Lecture 3b

- What is software configuration management and version control (used for)?
- What is the difference between version, revision, variant, configuration, baseline, release, merge, cherrypick, revision control?
- How can version control be used for clone-and-own? Illustrate!
- How to develop new features or variants? Exemplify!
- What are (dis)advantages of version control for variability?
- When (not) to use clone-and-own with version control?
- Shall we use version control at all?

Lecture 3c

- What are build systems (used for)?
- How can build systems be used for clone-and-own? Illustrate!
- How to develop new features or variants? Exemplify!
- What are (dis)advantages of build systems for variability?
- When (not) to use clone-and-own with build systems?
- Shall we use build systems at all?
- What is the granularity of clones for all three techniques?
- What is the effort to migrate from ad-hoc clone-and-own to managed clone-and-own?