1 Erkundung von fremdem Quellcode

Wir wollen zum Einen die Architektur bzw. den Kontext des Programms verstehen und zum Anderen auch die einzelnen Funktionseinheiten. Anschließend können wir dann sicher den Einstiegspunkt für unsere Änderung bestimmen.

1.1 Architektur verstehen

Grundlegend muss man verstanden haben, was der Zweck der Anwendung ist. Man sollte wenigstens einmal damit gearbeitet haben. Oft gibt es auch Dokumente die zeigen wie die Architektur aussieht. Eine Beschreibung von einem erfahrenen Entwickler ist meistens aktueller und es bietet sich die Möglichkeit Rückfragen zu stellen.

1.1.1 Die Geschichte des Systems erzählen

Voraussetzung: ich und ein System erfahrener Entwickler

1. Ich stelle Frage "Welche Architektur hat das System?"
2. Entwickler erklärt mit nur wenigen Konzepten (2-3) ganz grob das System (die Komponenten des Designs und ihre Interaktionen)
3. Ich stelle Frage "Ist das alles"? bzw. "Wie sieht es im Bereich X aus?"
4. Entwickler erklärt die nächstwichtigen Dinge kurz und knapp über das System
5. Gehe wieder zu Schritt 3 bis Ich alles Wichtige erfahren habe

1.2 Funktionseinheit verstehen

1.2.1 Notizen/Skizzen

Stellt sich beim Lesen von Code Verwirrung ein, lohnt es sich, Skizzen zu zeichnen und Notizen zu machen. z.B.

1. Schreibe Namen der letzten wichtigen Sache
2. Notiere Namen der vorletzten wichtigen Sache
3. Gibt es eine Beziehung? Ziehe Verbindungslinie

1.2.2 Vergleich Erwartung und Ergebnis

Voraussetzung: Debugger

1. Zerlege Funktion in Codeabschnitte
2. Notiere erwartetes Verhalten eines Codeabschnittes (in 3, 4 Worten)
3. Debugge die Anwendung und prüfe ob das Ergebnis mit der Erwartung übereinstimmt
4. Notiere das tatsächliche Verhalten neben deinen Erwartungen
5. Wiederhole ab Schritt 2 bis du alle Codeabschnitte bearbeitet hast
6. Schaue dir Erwartung und Ergebnis an, falls nicht übereinstimmend finde heraus warum

1.2.3 Listing Markup

Voraussetzung: Der Code muss frei markierbar sein. Entweder ausdrucken oder als Bild vorliegen haben. Anschließend muss man sich bewusst machen, was man möchte.

Aufgabe trennen Markiere zusammengehörige Code-Fragmente mit einer Farbe. Diese können auch verteilt liegen. Das Ziel ist hierbei, die Aufgaben aus ineinander verwobenen Quellcodeteilen zu extrahieren.

Struktur verstehen Ziehe Linien von dem Anfang des Blocks bis zum Ende, von dem innersten Block zum Äußersten. Das Ziel ist hierbei die Struktur offenzulegen und für jeden Block zu bestimmen, was genau dieser macht. Bestimmung der Blöcke bei C++ z.B.

1. Gehe vom Anfang über alle öffnenden bis zur ersten schließenden geschweiften Klammer
2. Markiere die Zeile und ziehe Linie zurück bis zur öffnenden geschweiften Klammer
3. Gehe weiter zur nächsten schließenden geschweiften Klammer
4. Wiederhole ab Schritt 3 bis am Ende der Funktion

Auswirkung der Änderung bestimmen In kleineren Quellcodeteilen reichen Markierungen aus. Das Ziel ist hierbei einen schnellen Überblick über das Ausmaß der gewünschten Änderungen in kleinem Rahmen zu erhalten.

1. Markiere die zu ändernden Codezeilen
2. Markiere alle Variablen und Methoden, die von den Änderungen beeinflusst werden
3. Markiere alle Variablen und Methoden, die von den Änderungen beeinflusst werden, die gerade markiert worden sind (neue Farbe)
4. Wiederhole Schritt 3 so oft, bis klar ist wie sich die Änderungen in dem Code fortpflanzen

Bei größeren Quellcodeteilen sind Effektskizzen besser geeignet.

Beispiel Auswirkung der Änderung des *elements* von `list<>` zu `vector<>`.

Marker	Quellcode

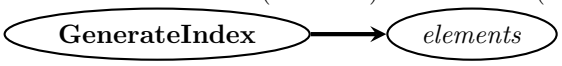
```
class InMemoryDirectory {
public:
  size_t GetElementCount() {return elements.size();}
  Element* GetElement(string name) {
    for(Element* e : elements) {
        if(e->GetName() == name)
          return e;
    }
    return nullptr;
  }
  void AddElement(Element* newElement) {
    elements.push_back(newElement);
  }
  void GenerateIndex() {
    Element* index = Element("index");
    for(Element* e : elements) {
      index.AddText(e->GetName() + "\n");
    }
    AddElement(index);
  }
private:
  list<Element*> elements = {};
};
```

1.3 Effektskizzen

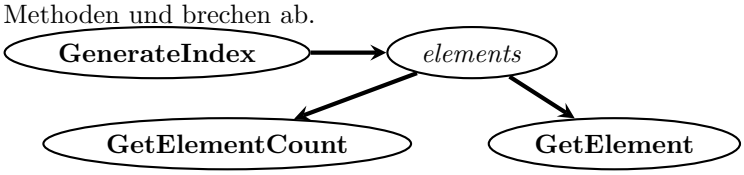
Definition. Die **Effektskizze** zeigt die Auswirkung bei Änderung von Programmteilen. Wir zeichnen einen Kreis um Dinge, die sich ändern. Von jeder dieser Änderungen (Auslöser) wird ein Pfeil zu allen Dingen, deren Wert sich zur Laufzeit dadurch ändert (Auswirkung), gezeichnet.

Abfolge am Beispiel Wir wollen das die Pflege des Index ein Nebeneffekt der Aktion, Element hinzufügen, wird.

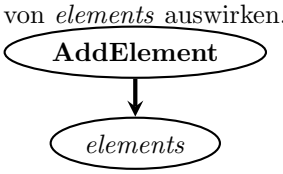
1. Änderungspunkte bestimmen
   - **GenerateIndex**
   - **AddElement**
2. Für jeden Änderungspunkt Effektskizze erstellen
   - In **GenerateIndex** ändern wir zur Laufzeit durch *AddElement* die Variable *elements*. Also ziehen wir ein Pfeil von **GenerateIndex** (Auslöser) zu *elements* (Auswirkung)

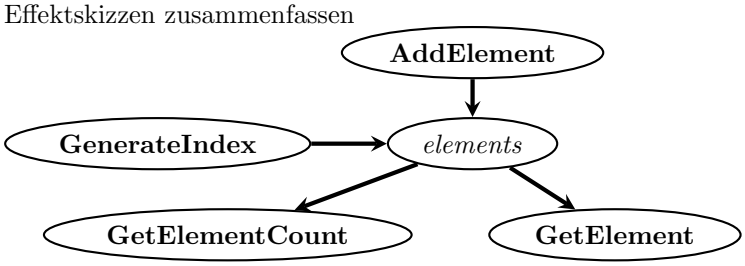
   - *elements* beeinflusst zur Laufzeit **GetElementCount** und **GetElement**. Also ziehen wir Pfeile von *elements* (Auslöser) zu **GetElementCount** und **GetElement** (Auswirkung). Hier ignorieren wir die Aufrufer der beiden Methoden und brechen ab.

   - Der zweite Änderungspunkt **AddElement** ändert zur Laufzeit *elements*.Also ziehen wir ein Pfeil von **AddElement** (Auslöser) zu *elements* (Auswirkung). Wir brechen hier ab, weil wir bereits sehen wie sich Änderungen von *elements* auswirken.

3. Effektskizzen zusammenfassen

1.3.1 Effektfortpflanzung

Es gibt drei grundlegende Formen, wie sich Effekte in Code fortpflanzen:

- Rückgabewerte, die von einem Aufrufer verwendet werden
- Änderung von Objekten, die als Parameter übergeben und später verwendet werden
- Änderung von statischen oder globalen Daten, die später verwendet werden

Mit folgender Heuristik kann man Effekte aufspüren:

1. Identifizieren die zu ändernde Methode
2. Wenn die Methode einen Rückgabewert hat, analysiere die Aufrufer
3. Modifiziert die Methode Werte? Prüfe Methoden die diese Werte verwenden
4. Prüfe etwaige Ober- und Unterklassen die diese Instanzvariablen und die Methode ebenfalls verwenden
5. Schaue nach Parametern der Methoden die du in der zu ändernde Methode aufrufst
6. Suche nach globalen Variablen, die von der Methode geändert werden