Erstellen Sie „richtige“ Tests

```
bool LevenshteinDistWordsOneEmpty() {
    vector<string> s1{""};
    vector<string> s2{"Hello", "I", "am", "a", "Test"};
    Levenshtein dist(s1, s2);
    string expected = "Subs Ins Ins Ins Ins ";
    string received = dist.backtrace();
    assert(expected == received);
    cout << "Needed steps: " << received;
    return (5 == dist.CalcLevenshteinDistance());
}
```

- Tests liefern true/false
- Test können unabhängig voneinander in jeder Reihenfolge ausgeführt werden
- Kein „assert“, wenn die zu testende Funktion was Unerwartetes liefert
- Möglichst keine Ausgabe (wenn dann in Datei umlenken oder nur im Fehlerfall

Beispiel für Test

```
// We calculate the Levenshtein distance of Tier and Tor which should be 2
    TEST(LevenshteinTest, LevenshteinDistTierTor)  {
        vector<string> s1{ "T", "i", "e", "r" };
        vector<string> s2{ "T", "o", "r" };
        Levenshtein dist(s1, s2);
        EXPECT_EQ(2, dist.CalcLevenshteinDistance());
         TRACE("Matrix of Tier/Tor\n" << dist.GetMatrixAsString());
        TRACE("Needed steps: " << dist.backtrace());
        return true;
    }
```

```
#define EXPECT_EQ(r, e) \
        if ((r) != (e) ) {\
              std::cerr << "Test failed for: " << \\
                #r << "==" << #e << std::endl; \
              return false; \
              }
```

Warnungs-Level von 3 auf 4 setzen bei VS

int mi_zCnt_m1 = mstr2.size() + 1;
Ergibt Warnung

Besser:
int mi_zCnt_m1 = static_cast<int>(mstr2.size()) + 1;