

BACHELORTHESES

Robustheit orientierender Messungen der Konzentration von Radon in Gebäuden

*Entwicklung eines Simulationstools
für virtuelle Messungskampagnen
anhand realer Messdaten*

Alexander Schoedon

Lesefassung A2
vom 29. Mai 2012

Freie wissenschaftliche Abschlussarbeit zur Erlangung
des akademischen Grades BACHELOR OF SCIENCE (B.Sc.)

29. Mai 2012

Autor der Thesis

Alexander Schoedon

Matrikel #0524846

Eingereicht im

Studiengang Betriebliche Umweltinformatik,
Fachbereich F2 - Ingenieurwissenschaften II,
Hochschule für Technik und Wirtschaft Berlin

Angefertigt im

Fachgebiet SW 1.1: Radon,
Abteilung SW 1: Überwachung der Radioaktivität in der Umwelt,
Fachbereich SW: Strahlenschutz und Umwelt,
Bundesamt für Strahlenschutz, Berlin

Betreuende Prüfer

Prof. Dr.-Ing. Jochen Wittmann

Hochschule für Technik und Wirtschaft Berlin

Dr. Bernd Hoffmann

Bundesamt für Strahlenschutz, Berlin

Lizenz

 – Dieses Werk wird unter den Bedingungen der Creative-Commons-Lizenz *Attribution-ShareAlike 3.0 (CC-BY-SA)* veröffentlicht und erfüllt die Grundlagen für den *Open-Access-Standard*, der freien Zugang zu wissenschaftlichen Informationen fördert.

Inhaltsverzeichnis

Inhaltsverzeichnis	i
Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Abkürzungsverzeichnis	ix
1 Einleitung	1
1.1 Problemdefinition	1
1.1.1 Radon in Gebäuden	1
1.1.2 EU-Norm zu Referenzwerten	2
1.2 Zielstellung	2
1.3 Das BUNDESAMT FÜR STRAHLENSCHUTZ	2
2 Edelgas Radon	5
2.1 Grundlegende Eigenschaften	5
2.2 Radon in Innenräumen	6
2.2.1 Quellen von Radon	6
2.2.2 Gesundheitliche Auswirkungen	7
2.2.3 Richt- und Grenzwerte	8
2.3 Messverfahren	9
2.3.1 Bewertende Messverfahren	10
2.3.2 Orientierende Messverfahren	10
2.4 Vorangegangene Studien	11

3 Modellbildung und Simulation	13
3.1 Systemdefinition	13
3.2 Modellbildung	14
3.3 Simulationstools	15
4 Anforderungsanalyse	17
4.1 Verarbeitung realer Messdaten	18
4.2 Generierung virtueller Messkampagnen	18
4.3 Statistische Auswertung	20
5 Spezifikation	23
5.1 Modellierung	23
5.2 Datenaufbereitung im CSV-Format	25
5.2.1 Vorgehensweise	25
5.2.2 Hinweise	27
5.3 Größenordnung der Simulationen	28
5.4 Statistische Grundlagen	30
6 Konzept	35
6.1 Plattform und Entwicklungsumgebung	35
6.1.1 Plattformunabhängige Entwicklung	35
6.1.2 Programmiersprache JAVA	36
6.1.3 Entwicklungsumgebung ECLIPSE	37
6.2 Vorgehensmodell	37
6.2.1 Möglichkeiten als Einzelentwickler	38
6.2.2 Das Prototyping	39
6.2.3 Open-Source-Softwareentwicklung	39
6.3 Versionierung	42
6.3.1 Release-Verwaltung	42
6.3.2 Etikettierung	43

7 Implementierung	45
7.1 Prototyp 0.1 - Konsolenanwendung	46
7.1.1 Domänenmodell	46
7.1.2 Simulationen	47
7.1.3 Protokollierung	48
7.1.4 Ergebnisse	50
7.1.5 Tests	52
7.2 Prototyp 0.2 - Datenbankanwendung	54
7.2.1 Überlegungen zur Persistenz	54
7.2.2 Statistik	57
7.2.3 Ergebnisse	59
7.2.4 Tests	60
7.3 Prototyp 0.3 - Desktopanwendung	67
7.3.1 Entwurf der Oberfläche	67
7.3.2 Nebenläufigkeit und Parallelisierung	70
7.3.3 Grafische Ausgabe der Ergebnisse	71
7.3.4 Tests	72
7.4 Prototyp 0.4 - öffentliche Testphase	73
7.4.1 ANT-Build-Skripte	73
7.4.2 Projektmanagement mit GITHUB	75
7.4.3 Aufgedeckte Probleme und Optimierungen	75
8 Fazit	77
8.1 Resultat	77
8.2 Ausblick	78
Quellenverzeichnis	A
A Literatur	A
B Internetverweise	D

Abbildungsverzeichnis

2.1	Radon im Periodensystem der Elemente	6
2.2	Eintrittsstellen für Radon in Gebäuden [76]	7
3.1	Das System „Wohnhaus“	14
3.2	Das Modell „Wohnaus“ mit bewohnten Räumen (R) und unbewohnten Kellerräumen (C)	14
3.3	Morphologischer Kasten für die Einordnung von Simulationstools [nach 41, S. 252]	15
3.4	Einordnung des OM-Simulationstools (violett) in den Morphologischen Kasten [nach 41, S. 252]	16
4.1	Komplexität möglicher Variationen von Messungskampagnen	20
4.2	Übersicht der zu berechnenden statistischen Größen	21
5.1	Ablaufdiagramm für das Simulationstool	24
5.2	Positionen der Kellermessungen $P = 7$ bei Gebäuden mit $R \geq 6$	28
5.3	Mögliche Kellermessungen $P = 6$ und Mehrfachmessungen $M = 5$ bei Gebäuden mit $R = 5$	29
5.4	Kellermessungen $P = 5$ und Mehrfachmessungen $M = 6$ mit $R = 4$	29
5.5	Positionen der Kellermessungen bei $R = 3$, $R = 2$ und $R = 1$	30
6.1	Trennung von Compiler und Interpreter in Java	36
6.2	Architektur der ECLIPSE-IDE (mit Java-Plugins, [nach 29, S. 7])	37
6.3	Grundprinzip der evolutionären Softwareentwicklung [nach 21, S. 115]	38
6.4	Unterschiedliche Softwarelizenzierungen [nach 59]	42
6.5	Der Edit-Compile-Test-Zyklus [nach 5, S. 155]	43

6.6	Etikettierungsschema des Simulationstools	43
7.1	Paketdiagramm des ersten Prototypen	46
7.2	Logviewer im Vergleich: LTFVIEWER und LESS	52
7.3	Klassendiagramm des ersten Prototypen	53
7.4	H2-Datenbank im eingebetteten Modus [66]	56
7.5	Die JVM überschreitet den zugesicherten Arbeitsspeicher und bricht ab	58
7.6	Paketdiagramm des zweiten Prototypen	59
7.7	Import-Ansicht	68
7.8	Datensatz-Ansicht	68
7.9	Simulations-Ansicht	69
7.10	Auswertungs-Ansicht	69
7.11	Testansicht	70
7.12	Verteilungsfunktion der arithmetischen Mittel der bewohnten Räume einer Simulation mit 10.000 generierten Messkampagnen	72
8.1	Screenshot des fertigen Simulationstools	77

Tabellenverzeichnis

2.1	Eigenschaften der Edelgase [nach 7, S. 217][32, S. 393]	5
5.1	Anzahl möglicher Variationen bei $R \geq 6$	28
5.2	Anzahl möglicher Variationen in Abhängigkeit von der Anzahl der bewohnten Räume	30
6.1	Die populärsten Plattformen [82]	36
6.2	Die populärsten Programmiersprachen [79]	36
6.3	Vor- & Nachteile des Prototyping [nach 6, S. 23]	40
6.4	Wesentliche Vor- & Nachteile von Open-Source-Software für die behördliche Nutzung [nach 60]	41
7.1	20 CSV-Testdateien	61
7.2	Ergebnisse der Funktionstests	62
7.3	Benchmarks der systematischen Simulationen	64
7.4	Vorraussichtliche Systemlast	65
7.5	Ergebnisse der ersten Lasttests	65
7.6	Ergebnisse weiterer Lasttests	65
7.7	Ergebnisse weiterer Lasttests	66

Abkürzungsverzeichnis

6+1	Messprotokoll des Schnellverfahrens ‘6+1’ [25, S. 10]
AM	Arithmetischer Mittelwert
BfS	Bundesamt für Strahlenschutz
BMU	Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit
Bq	Becquerel - SI-Einheit der Radioaktivität
BUI	Studiengang Betriebliche Umweltinformatik
BUIS	Betriebliche Umweltinformationssysteme
BVA	Bundesverwaltungsamt
C	<i>en. cellar</i> - unbewohnte, bodennahe Keller-Räume in Gebäuden
CSV	<i>en. comma seperated values</i> - einfach strukturiertes Textformat
CV	<i>en. coefficient of variation</i> - Variationskoeffizient
Dev	<i>en. development</i> - Abkürzung wird benutzt um ungetestete Entwicklerversionen zu kennzeichnen
DNS	Desoxyribonukleinsäure, das <i>Erbgut</i> von Zellen
FSF	FREE SOFTWARE FOUNDATION
GM	Geometrischer Mittelwert
GPL	GNU GENERAL PUBLIC LICENCE
GSD	<i>en. geometric standard deviation</i> - Geometrische Standardabweichung
GUI	graphical user interface
HTW	Hochschule für Technik und Wirtschaft Berlin
ICRP	Internationale Strahlenschutzkommission
ID	Identifikator der Messreihen
IDE	<i>en. integrated development environment</i> - Integrierte Entwicklungsumgebung
JAR	<i>en. java archive</i> - bündelt Java-Quellen und -Bibliotheken in einer einzigen JAR-Datei
JDBC	<i>en. java database connectivity</i> - Treiber für die Verbindung von Java-Programmen mit gängigen relationalen Datenbanken
JDT	<i>en. Java Development Toolkit</i> - ECLIPSE-Plugin für Java
JVM	Java Virtual Machine
M	<i>en. miscellaneous</i> - sonstige, irrelevante Räume in Gebäuden
MAX	Maximum, Maxima
MDA	<i>en. minimal detectable activity</i> - kleinste messbare oder nachweisbare Aktivität

NoSQL	<i>en. not only SQL</i> - Oberbegriff für nicht-relationale Datenbanken
ORDBMS	Objektrelationale Datenbank-Managementsysteme
OSI	OPEN SOURCE INITIATIVE
OSS	<i>en. open source software</i> - quellenoffene, freie Software
PDE	<i>en. Plug-in Development Environment</i> - ECLIPSE-Plugin-Entwicklungsumgebung
PDF	<i>en. portable document format</i> - plattformunabhängiges Dateiformat für Dokumente
Q5	Quantil 5 (<i>auch: Perzentil 5</i>)
Q50	Quantil 50, Median
Q95	Quantil 95 (<i>auch: Perzentil 95</i>)
QD	<i>en. quantile deviation</i> - Perzentilsabweichung
R	<i>en. room</i> - bewohnte Räume in Gebäuden
RC	<i>en. release candidate</i> - Abkürzung wird benutzt um vollständig gestestete Vorabversionen zu kennzeichnen
RQD	<i>en. relative quantile deviation</i> - relative Perzentilsabweichung
SD	<i>en. standard deviation</i> - Arithmetische Standardabweichung
SQL	<i>en. structured query language</i> - eine Datenbanksprache für relationale Datenbanken
SSK	Deutsche Strahlenschutzkommission
SVN	<i>en. subversion</i> - Versionierungssoftware
SWT	Standard-Widget-Toolkit
TIFF	<i>en. tagged image file format</i> - Dateiformat zur Speicherung von Bildern
TOP	Tagesordnungspunkt
VAR	Varianz - die mittlere quadratische Abweichung
WHO	Weltgesundheitsorganisation
XLS	<i>en. excel spreadsheet</i> - MICROSOFT-Dateiformat zur Speicherung von Tabellen und Datenblättern
XLSQL	Excel-JDBC-Treiber, ein Datenbank-Treiber für Java, welcher Daten in XLS, XML oder CSV-Dateien ablegt
XLSX	<i>en. excel spreadsheet extended</i> - erweitertes MICROSOFT-Dateiformat zur Speicherung von Tabellen und Datenblättern
XML	<i>en. extensible markup language</i> - hierarchisch strukturiertes Textformat
XP	<i>en. extreme programming</i> - Vorgehensmodell in der agilen Softwareentwicklung

Kapitel 1

Einleitung

Das radioaktive Edelgas Radon stellt das zweitgrößte Lungenkrebsrisiko nach dem Tabakrauchen dar – für Nichtraucher sogar das größte [51, S. 3]. Diese Problematik ist aber in der Bevölkerung weitgehend unbekannt.

1.1 Problemdefinition

Da Radon geruchlos, geschmacklos und fabrlos ist, stellt es auch eine über längere Zeit eine unmerkliche Gefahr für den Organismus dar. So führt die ionisierende Strahlung durch den Zerfall von Radon und seinen Folgeprodukten in der Lunge zu Schädigungen von DNS und Zellstruktur und kann somit Krebs verursachen.

1.1.1 Radon in Gebäuden

Radon entweicht aus dem Boden und reichert sich in Gebäuden an. Die Konzentrationen von Radon in der Innenraumluft ist im Schnitt fünfmal höher als die der Außenluft [16, S. 17]. Dies ist vorwiegend in Wohngebäuden, aber auch am Arbeitsplatz eine ernstzunehmende Belastung.

Es gibt keine fest definierten Grenzwerte in der Europäischen Union. In Europa ist die Schweiz das einzige Land, dass Radon-Grenzwerte gesetzlich verankert hat und somit Besitzer von stark belasteten Gebäude zu verschiedenen radonsenkenden Maßnahmen zwingen kann.

Auch gibt es kaum Einigkeit bei der Bestimmung von sinnvollen Richtwerten. So schwanken die Empfehlungen verschiedener nationaler und internationaler Organisationen teilweise um den Faktor 10 und größer.

1.1.2 EU-Norm zu Referenzwerten

Nun liegen aber Pläne der Europäischen Union (EU) vor, eine neue Richtlinie zu erlassen, die verschiedene Themen des Strahlenschutzes zusammenfassen und nach Stand der Wissenschaft und Technik neu formulieren. So werden unter anderem Radon-Referenzwerte neu definiert [17, S. 62, 75]. Mit dem Inkrafttreten dieser Richtlinie werden die Mitgliedsstaaten der EU dazu verpflichtet, diese Werte in nationales Recht zu übernehmen und Maßnahmen zur Messung-, Bewertung und Verminderung der Radonkonzentration in Gebäuden zu entwickeln und umzusetzen.

Dass dies nicht trivial ist, wird in Kapitel 2 diskutiert. Denn es gibt bis dato keine zuverlässigen Mess- und Bewertungsverfahren, die die Radonbelastung von Gebäuden in weniger als drei Monaten akkurat bestimmen können. Das Bundesamt für Strahlenschutz (BfS) untersucht zur Zeit die Aussagekraft und Zweckmäßigkeit eines neuen Messprotokolls für orientierende Messungen (OM), welches eine erste Aussage zur möglichen Radonbelastung eines Gebäudes innerhalb von sieben Tagen liefern soll.

1.2 Zielstellung

Zu diesem Zweck wird am Bundesamt für Strahlenschutz ein Simulationstool entwickelt, welches – anhand realer Messdaten von Radonkonzentrationen in Gebäuden – virtuelle Messkampagnen generieren soll, die dem Protokoll des Schnellverfahrens entsprechen. Anhand dieser Simulationen und der ausführlichen Statistik selbiger, soll eine Evaluierung der orientierenden Messungen ermöglicht werden.

Diese Bachelorthesis stellt den ersten Teil der Überprüfung der Robustheit des Messverfahrens da und bildet die grundlegende technische Dokumentation. Nach Abschluss dieser Arbeit steht dem Bundesamt für Strahlenschutz ein Software-Werkzeug zur Verfügung, mit welchem dann die eigentliche Evaluierung möglich ist.

1.3 Das BUNDESAMT FÜR STRAHLENSCHUTZ

Das Bundesamt für Strahlenschutz ist eine selbstständige, wissenschaftliche Bundesoberbehörde, die dem Geschäftsbereich des Bundesministeriums für Umwelt, Naturschutz und Reaktorsicherheit (BMU) unterstellt ist. Das BfS wurde 1989 gegründet und ist vorwiegend mit den Verwaltungsaufgaben des Bundes im Bereich des Strahlenschutzes betraut [14, S. 82ff].

Die Behörde ist in vier grundlegende Fachbereiche gegliedert, die auch die Kernkompetenzen wiederspiegeln [12, S. 8]:

- Sicherheit der Kerntechnik (SK)
- Sicherheit nuklearer Entsorgung (SE)
- Strahlenschutz und Gesundheit (SG)
- Strahlenschutz und Umwelt (SW)

Die vorliegende Arbeit wurde im Fachgebiet SW 1.1 angefertigt, welches sich mit der Überwachung der Radioaktivität der Umwelt – im besonderen mit Radon – auseinander setzt [12, S. 19f].

Kapitel 2

Edelgas Radon

Radon ist ein natürliches vorkommendes Edelgas und wurde 1900 von Friedrich Ernst DORN entdeckt [73], als er den Zerfall von Radium untersuchte. Edelgase bilden die Elemente der 8. Hauptgruppe im Periodensystem (vgl. Abbildung 2.1) und verfügen über die besondere Eigenschaft, dass sie in atomarer Form stabil¹ sind, während andere Elemente dazu neigen, Verbindungen einzugehen und Moleküle zu bilden [7, S. 8]. Dies röhrt daher, dass die Elemente der Edelgasgruppe eine voll besetzte äußere Elektronenschale besitzen und daher eine hohe Ionisierungsenergie aufgebracht werden muss, um Elektronen aus dieser Hülle zu lösen.

Edelgas	He	Ne	Ar	Kr	Xe	Rn	Uuo
Ordnungszahl	2	10	18	36	54	86	118
Atommasse [u]	4,003	20,18	39,95	83,80	131,29	[222]	[294]
Vorkommen [%]	5,24E-04	1,82E-03	9,34E-01	1,14E-04	8,7E-06	6,1E-15	<i>syn.</i>

Tabelle 2.1: Eigenschaften der Edelgase [nach 7, S. 217][32, S. 393]

2.1 Grundlegende Eigenschaften

Radon ist – wie alle anderen Edelgase auch – farblos, geruchlos und geschmacklos. Es entsteht hauptsächlich auf natürliche Weise durch spontanen Zerfall von Atomkernen aus den drei Zerfallsketten der Plutonium-Thorium-Reihe (Ausgangsnuklid ^{244}Pu), der Uran-Actinium-Reihe (Ausgangsnuklid ^{235}U) und der Uran-Radium-Reihe (Ausgangsnuklid ^{238}U) [7, S. 360]. In diesen drei natürlichen Zerfallsketten bilden sich ^{222}Rn aus der ^{238}U -Kette, ^{219}Rn aus der ^{235}U -Kette und ^{220}Rn aus der ^{244}Pu -Kette. Dies sind die drei häufigsten Radon-Isotope. ^{219}Rn – historisch auch Actinon (An) genannt – ist das kurzlebigste dieser drei Isotope mit einer Halbwertszeit von 3,96 Sekunden. ^{220}Rn – öfters auch als Thoron (Tn) bezeichnet – weist eine

¹Mit *stabil* sei hier die chemische Eigenschaft bezeichnet, die Elektronenaustausch und Molekülbinding aufgrund der hohen Ionisierungsenergie verhindert. Physikalisch betrachtet sind Radon und Ununoctium alles andere als *stabil*, da sie als Radionuklide nach einiger Zeit zerfallen.

Halbwertszeit von 55,6 Sekunden auf. Circa 90% aller Radon-Isotope bilden aber das Radionuklid ^{222}Rn welches mit 3,82 Tagen Halbwertszeit das langlebigste ist [16, S. 16]. Als einziges Isotop mit einer Halbwertszeit über mehrere Tage hat ^{222}Rn die Möglichkeit sich im Boden, in der Atmosphäre und in Gebäuden auszubreiten. Daher ist in dieser Thesis im folgenden immer, wenn von *Radon* die Rede ist, das Isotop ^{222}Rn gemeint.

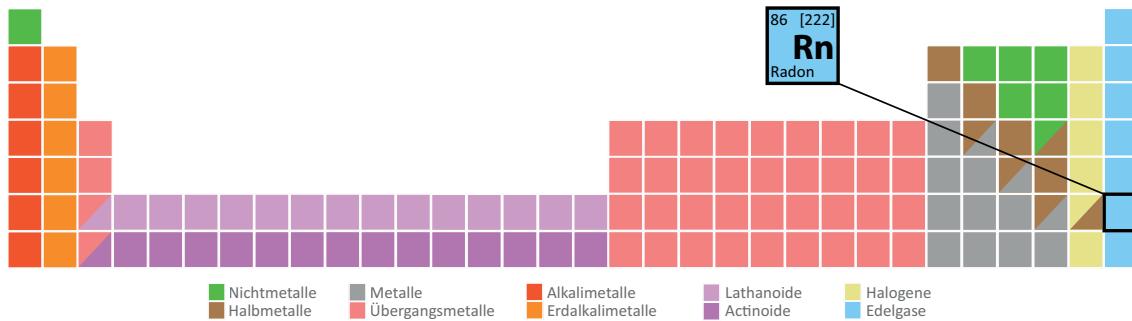


Abbildung 2.1: Radon im Periodensystem der Elemente

2.2 Radon in Innenräumen

Da Radon sehr flüchtig ist, kann es durch Poren und Risse in Mauerwerk oder Fundament sowie durch kleinste undichte Stellen in Rohr- oder Kabelschächten in Gebäude eindringen und sich dort anreichern. Daher weist oft die Raumluft im Mittel fünfmal höhere Radonkonzentrationen auf, als die Außenluft [16, S. 17], da in der Atmosphäre das Radon schnell durch Wind verbreitet und verdünnt wird.

2.2.1 Quellen von Radon

Radon ist meist direktes Produkt des Zerfalls von Radium, welches ein Erdalkalimetall ist und in Boden und Gestein vorkommt [56]. Bei dem Alphazerfall von Radium entsteht Helium, Radon und ionisierende Strahlung, die sogenannte Radioaktivität. Diese wurde 1896 von dem französischen Physiker Antoine Henri BECQUEREL entdeckt, welcher 1903 dafür den Nobelpreis für Physik zusammen mit Pierre und Marie CURIE erhielt [7, S. 359].

Das nun entstandene Radon entweicht dem Boden und dringt in die bodennahe Luft, aber auch in Gebäude ein [32, S. 393]. Da Radon mit $9,73 \frac{\text{kg}}{\text{m}^3}$ eine größere Dichte als das normale Luftgemisch der Erdatmosphäre ($1,20 \frac{\text{kg}}{\text{m}^3}$) aufweist, reichert es sich vorwiegend in Kellern und Erdgeschossen mit Bodenkontakt an [7, S. 218] (vgl. Abbildung 2.2).



Abbildung 2.2: Eintrittsstellen für Radon in Gebäuden [76]

2.2.2 Gesundheitliche Auswirkungen

Alle Menschen, aber auch Tiere und Pflanzen sind fast ausnahmslos immer einer geringen Radiaktivität ausgesetzt. Diese kann verschiedenste Quellen haben. Ein Mensch in der Bundesrepublik Deutschland lebt mit einer durchschnittlichen Jahresbelastung von circa $2,1\text{mSv}$, wovon mehr als die Hälfte ($1,1\text{mSv}$) auf Radon und Radon-Folgeprodukte zurückzuführen ist. Aufgrund der höheren Radonkonzentration in der Innenraumluft sowie der geringeren Aufenthaltsdauer von Menschen in der Außenluft beträgt die jährliche Strahlendosis von Radon in Gebäuden im Schnitt $0,9\text{mSv}$ [14, S. 72]. Zum Vergleich: Das lebenswichtige Kalium (^{40}K), welches über die Nahrung aufgenommen wird, stellt mit 8000 bis 9000 Zerfällen pro Sekunde im Körper eine Jahresbelastung von $0,3\text{mSv}$ dar [16, S. 16ff]

Ionisierende Strahlung geringer Intensität ist in der Hinsicht gefährlich, weil über längere Zeit unmerklich DNS und Zellstrukturen geschädigt werden und somit Krebs, Leukämie, Geburtsfehler und Säuglingssterben verursacht werden können [7, S. 362]. 1,5% aller Krebsursachen lassen sich auf natürliche Radioaktivität zurückführen [24, S. 107f].

Radon wird über die Atemluft aufgenommen, stellt aber selbst das geringere Risiko dar, da es sich nicht oder nur schlecht chemisch im Körper bindet und als Gas einfach wieder ausgeatmet wird [32, S. 393]. Problematisch sind vor allem die Radon-Folgeprodukte, die kurzlebigen Isotope Polonium (^{218}Po , ^{214}Po , ^{210}Po), Bismut (^{214}Bi , ^{210}Bi) und Blei (^{214}Pb , ^{210}Pb), die sich beim Zerfall von Radon bilden und sich im Lungengewebe absetzen können. Bei dem weiterem Zerfall dieser Metalle direkt auf dem Gewebe ist das Risiko der Schädigung der Zellstruktur durch die ionisierende Strahlung hoch [24, S. 97].

Epidemiologische Studien bestätigen das Lungenkrebsrisiko von Radon in Gebäuden bei der Bevölkerung. So beschreibt die Weltgesundheitsorganisation (WHO), dass – je nach Berechnungsgrundlage – etwa 3-14% aller Lungenkrebsfälle auf Radon

zurückzuführen sind [51, S. 3]. Damit stellt Radon das zweithäufigste Lungenkrebsrisiko nach dem Tabakrauchen dar. Das Bundesamt für Strahlenschutz beziffert in Deutschland die Todesfälle aufgrund radonbedingtem Lungenkrebs auf circa 1.900 pro Jahr [15, S. 240, @TODO: Primärquelle?].

2.2.3 Richt- und Grenzwerte

Als Maß für die Aktivität von Radon in der Luft wird die Einheit BECQUEREL pro Kubikmeter ($\frac{Bq}{m^3}$) verwendet. Eine Aktivität von $1\frac{Bq}{m^3}$ entspricht einem radioaktivem Zerfall pro Sekunde und Kubikmeter Luftvolumen. Im Boden kann Radon Konzentrationen von mehreren $100.000\frac{Bq}{m^3}$ erreichen, in Gebäuden liegen die Konzentrationen meist zwischen 10 und $1.000\frac{Bq}{m^3}$, in Einzelfällen auch über $10.000\frac{Bq}{m^3}$.

Es gibt laut Weltgesundheitsorganisation keine bekannten Untergrenzen, bei denen Radon kein Risiko darstellt [51, S. 3]. Demnach besteht auch ein Lungenkrebsrisiko bei Werten von unter $200\frac{Bq}{m^3}$, welcher vielen Staaten als Referenzwert dient. Nach der Auswertung von 22 Radon-Langzeitstudien aus Nordamerika, Europa und China wurde ein Anstieg des Krebsrisikos von 10% pro $100\frac{Bq}{m^3}$ Aktivität ermittelt [51, S. 12].

Es gibt bis dato keine einheitlichen Richt- oder Grenzwerte, weder auf nationaler noch auf internationaler Ebene. Im folgenden seien einige Richtwertempfehlungen verschiedener nationaler und internationaler Organisationen aufgeführt.

- Die Internationale Strahlenschutzkommision (ICRP) empfiehlt Maßnahmen ab $600\frac{Bq}{m^3}$ in Wohnungen und ab $1.500\frac{Bq}{m^3}$ am Arbeitsplatz zur Radonsenkung [4, S. 4].
- Die WHO empfiehlt aus gesundheitlicher Sicht Referenzwerte von $100\frac{Bq}{m^3}$ nicht zu überschreiten. Ist dies nicht möglich, sollten aber Werte über $300\frac{Bq}{m^3}$ vermieden werden [51, S. 90].
- Die Europäische Union empfiehlt bisher Planungswerte von $200\frac{Bq}{m^3}$ für Neubauten und Referenzwerte von $400\frac{Bq}{m^3}$ in bereits existierenden Gebäuden [vgl. 37].
- Das Bundesamt für Strahlenschutz empfiehlt radonsenkende Maßnahmen bei Jahresmittelwerten von über $100\frac{Bq}{m^3}$ [16, S. 17].
- Die Deutsche Strahlenschutzkommision (SSK) hält keine Maßnahmen für Werte unter $250\frac{Bq}{m^3}$ für notwendig [4, S. 4].

Als einziges Land hat die Schweiz verbindliche Grenzwerte eingeführt. Demnach schreibt die schweizerische Gesetzgebung Maßnahmen zur Senkung der Radonkonzentration ab $1.000\frac{Bq}{m^3}$ vor [26, S. 63ff].

Zur Zeit bereitet die EU eine neue Richtlinie „*zur Festlegung grundlegender Sicherheitsnormen für den Schutz vor Gefahren einer Exposition gegenüber ionisierender Strahlung*“ vor, welche auch neue Referenzwerte für Radon in Wohnungen, öffentlichen Gebäuden und am Arbeitsplatz vorsieht. Dieser Vorschlag wurde am 25. November 2011 der 890. Sitzung des Deutschen Bundesrates (TOP 44) zur Stellung vorgelegt [vgl. 17]. Diese Richtlinie empfiehlt folgende Referenzwerte, die nach Beschluss durch das Europäische Parlament auch in nationales Recht umgesetzt werden müssen.

- **Artikel 53 Absatz 1** [17, S. 62]:

„[...] nationale Referenzwerte für die Radonkonzentration [...] dürfen einen jährlichen Durchschnittswert von 1000 Bq m^{-3} am Arbeitsplatz nicht überschreiten.“

- **Artikel 74 Absatz 1** [17, S. 75]:

„Im Rahmen des in Artikel 103 genannten Maßnahmenplans legen die Mitgliedsstaaten nationale Referenzwerte für die Radonkonzentration in Gebäuden fest, die (im jährlichen Durchschnitt) folgende Werte nicht überschreiten dürfen:

- (a) 200 Bq m^{-3} in neuen Wohnräumen und neuen öffentlich zugänglichen Gebäuden;
- (b) 300 Bq m^{-3} in bestehenden Wohnräumen;
- (c) 300 Bq m^{-3} in bestehenden öffentlich zugänglichen Gebäuden.
In bestimmten Fällen, in denen die Aufenthaltsdauer nur kurz ist, kann ein Referenzwert von bis zu 1000 Bq m^{-3} festgelegt werden.“

- **Artikel 100 Absatz 1** [17, S. 88]:

„Die Mitgliedsstaaten sorgen dafür, dass Programme eingeführt werden, um bestehende Expositionssituationen zu ermitteln und zu bewerten [...]“

Der Artikel 100 ist von zentraler Bedeutung und hebt die Relevanz der folgenden Arbeit hervor. Das Bundesamt für Strahlenschutz wird mit Inkrafttreten dieser Richtlinie früher oder später die Aufgabe übernehmen müssen, bestehende Expositionssituationen unter anderem von Radon in Gebäuden zu ermitteln und zu bewerten, das heißt, auch geeignete und effiziente Mess- und Bewertungsverfahren zu entwickeln. Dass diese Aufgabe nicht trivial ist, wird im folgenden Abschnitt näher erläutert.

2.3 Messverfahren

Die Radonkonzentrationen in Gebäuden schwanken täglich, saisonal und sogar in Jahr-zu-Jahr-Zyklen [51, S. 7], was Mess- und Bewertungsverfahren vor große Hürden stellt. Wesentliche Faktoren, die die Konzentration in der Raumluft beeinflussen,

sind vielfältig und üben oft gegensätzliche schwer vorhersehbare Effekte aus. So ist zum Beispiel im Sommer aufgrund der höheren Temperaturen die Bodenmigration von Radon höher, aber dadurch, dass im Sommer öfter gelüftet wird, findet in den Gebäuden einen höheren Luftwechsel statt, was die Radonkonzentration wieder sinken lassen kann [24, S. 91]. Die Konsequenz aus diesem Problem ist, dass Gebäude hinsichtlich ihrer Radonbelastung nur um so besser bewertet werden können, um so länger die Gebäude gemessen werden.

2.3.1 Bewertende Messverfahren

Es hat sich daher durchgesetzt, bewertende Langzeitmessungen von mindestens drei, aber optimaler Weise zwölf Monaten durchzuführen und damit Jahresmittelwerte zu ermitteln. Für Langzeitmessungen werden meist passive Exposimeter mit Kernspur- oder Elektretdetektoren verwendet [13].

2.3.2 Orientierende Messverfahren

Es gibt zur Zeit keinerlei Kurzzeitmessverfahren, die es ermöglichen akkurate Rückschlüsse auf Jahresmittelwerte zu schließen. Kurzzeitmessungen können auch bei wiederholten Messungen zu Ergebnissen kommen, die um den Faktor zwei oder mehr von den Ergebnissen von Langzeitmessungen abweichen können [51, S. 28] und sind daher sehr unzuverlässig, es sei denn die Radonwerte sind extrem hoch.

Ein von mitteleuropäischen Forschern entwickeltes Protokoll für orientierende Messungen (OM) soll nun Abhilfe schaffen. Dabei wird über den Zeitraum von sieben Tagen eine Abschätzungen zur Radonbelastung eines Gebäudes ermöglicht [25, S. 10] [vgl. auch 26, S. 63ff]. Dieses Messprotokoll ist wie folgt definiert.

- Das Gebäude muss während der Messperiode normal bewohnt werden. Vor Beginn der Messungen soll das Gebäude gründlich durchgelüftet werden, während der Messungen sollte aber auf geschlossene Fenster und Türen geachtet werden.
- Es wird mit einem aktiven Messgerät zeitauf lösend die Radonkonzentration in dem Gebäude über einen Zeitraum von sieben Tagen gemessen. Die Auflösung der Messung soll mindestens 10 bis maximal 60 Minuten betragen.
- Das Messgerät wird alle 24 Stunden in einen anderen Raum gestellt, so dass
 - an sechs Tagen in bewohnten Räumen wie zum Beispiel Wohnzimmern, Schlafzimmern oder Kinderzimmern gemessen wird.
 - an einem Tag in einem unbewohnten Raum oder Keller mit höchst annehmender Radonkonzentration gemessen wird. Dieser Raum muss Bodenkontakt besitzen und somit die vermutliche Eintrittsstelle von Radon im Haus sein.

- Während der Messphase müssen täglich Wetterdaten wie Außentemperatur, Niederschläge oder Wind protokolliert werden.

Dieses Messprotokoll wird im folgenden auch als Protokoll „6+1“ referenziert und dient als Grundlage für diese Arbeit, da es grundlegend durch Computersimulationen auf Robustheit untersucht werden soll.

2.4 Vorangegangene Studien

Für diese Arbeit von zentraler Bedeutung ist ein Forschungsvorhaben aus dem Jahre 2009 vom BUNDESMINISTERIUM FÜR UMWELT, NATURSCHUTZ UND REAKTORSICHERHEIT (BMU), welches vom BUNDESAMT FÜR STRAHLENSCHUTZ (BfS) begleitet wurde. Diese Studie hatte die „*Erarbeitung fachlicher Grundlagen für die Entwicklung zeit- und kosteneffektiver Verfahren zur Bestimmung von Strahlenexpositionen durch Radon in Wohnungen*“ (im Folgenden „Vorhaben St.Sch. 4534“ genannt) zum Ziel [vgl. 15, S. 7ff].

Dabei wurden 10 Gebäude in Deutschland ausgewählt und detailliert nach verschiedenen Verfahren untersucht. Die Komplexität der Messungen ist in dieser Studie bis dato einzigartig. So wurden parallel

- in den 10 Objekten zeitauf lösende Kurzzeitmessungen durchgeführt. Dabei wurden in allen relevant erscheinenden Räumen aktive Messgeräte für ein bis zwei Wochen aufgestellt. Diese Messungen wurden in zwei verschiedenen Jahreszeiten wiederholt.
- in möglichst jedem Raum Radon-Langzeitmessungen zur Bestimmung von Jahresmittelwerten durchgeführt.
- auf den Grundstücken Bodenluftmessungen durchgeführt und Bodenproben zur Untersuchung auf relevante Radionuklide entnommen.
- Messungen zur Bestimmung der Luftwechsel innerhalb der Gebäude durchgeführt.

Die ausführlichen Daten aus dem Abschlussbericht zum Vorhaben St.Sch. 4534 [3] – vor allem die der zeitauf lösenden Kurzzeitmessungen – eignen sich hervorragend für die Überprüfung der Robustheit orientierender Messungen und dienen somit dieser Thesis als Arbeitsgrundlage.

Kapitel 3

Modellbildung und Simulation

Seit dem Menschen denken können, werden komplexe Systeme, deren Elemente und deren Wirkungsstrukturen in Denkmodellen vereinfacht und in Gedanken durchgespielt. Derartige Gedankenspiele waren schon immer einfache Hilfsmittel zum Verständnis der Welt [8, S. 5]. Dies sind die ersten Vorläufer der Modellbildung und Simulation, die wir heute durch Computermodelle und -simulationen ersetzt haben, um die höhere Komplexität von Systemen begreifen zu können [8, S. 13]. Denn computergestützte Prozesse können schnell und genau jede Formulierung in beliebiger Kombination simulieren. Durch Modellbildung und Simulation eröffnet sich eine experimentelle Technologie zur Modellierung von virtuellen oder realen Systemen, welche wir hinsichtlich Beschaffenheit und Verhaltensweise verstehen wollen [40, S. 3].

In den folgenden Abschnitten sollen die Begriffe System, Modell und Simulation von einander abgegrenzt werden und anhand des hier behandelten Problems – eines radonbelasteten Gebäudes – erläutert werden.

3.1 Systemdefinition

Ein System ist ein Teilbereich der Realität, der näher betrachtet oder auf bestimmte Eigenschaften und Verhaltensweisen hin untersucht werden soll. Er wird durch Systemgrenzen zur Systemumwelt abgetrennt [40, S. 4f].

Abbildung 3.1 verdeutlicht das System des Wohnhauses. Die angrenzende Atmosphäre und der Boden sind die Systemumwelt, die nicht zum eigentlichen System gehören, aber eine Möglichkeit zur Interaktion bieten. Die Außenwände des Gebäudes bilden die Systemgrenzen.

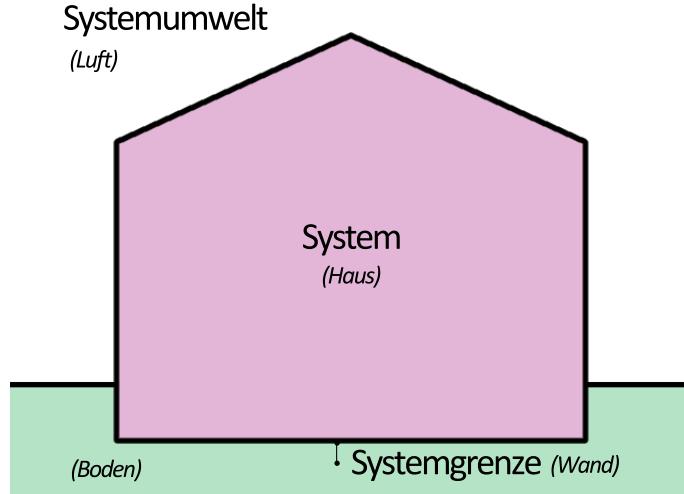


Abbildung 3.1: Das System „Wohnhaus“

3.2 Modellbildung

Ein Modell ist eine Abbildung eines Systems, welche einen definierten Bestimmungsgrund aufweist oder auf spezielle, experimentelle Fragestellungen zugeschnitten ist [40, S. 5f]. Im vorliegenden Problemfall des radonbelasteten Wohnhauses besteht das Modell aus bewohnten Räumen und unbewohnten Kellerräumen (Abbildung 3.2). Die Modellbildung orientiert sich hierbei an dem durch ein von mitteleuropäischen Forschern entwickeltes Protokoll „6+1“ für orientierende Messverfahren [25, S. 10] (vgl. Abschnitt 2.3.2 *Orientierende Messverfahren*).

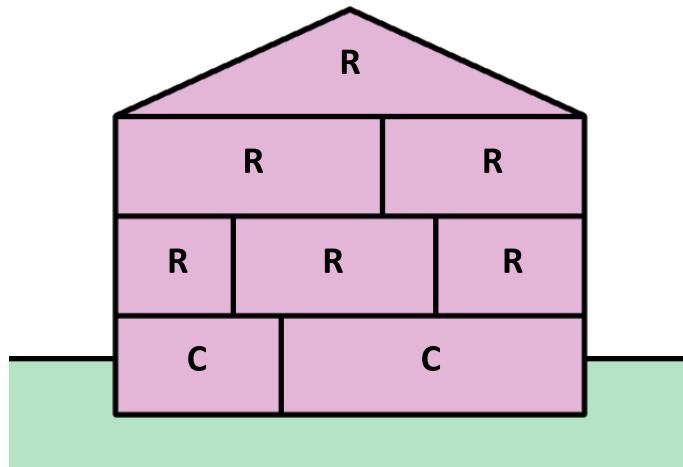


Abbildung 3.2: Das Modell „Wohnhaus“ mit bewohnten Räumen (*R*) und unbewohnten Kellerräumen (*C*)

Hierbei handelt es sich um ein statisches, systemerklärendes Modell, welches das Verhalten eines realen Systems reproduziert [8, S. 21], da es sich an realen Messdaten bedient (vgl. Abschnitt 4.1 *Verarbeitung realer Messdaten*).

3.3 Simulationstools

Einer der frühesten Anwendungsfälle der Informationstechnologie sind Simulationen. Diese sind Modellierungen dynamischer Prozesse in real existierenden Systemen, welche auf realen Daten basieren, um – im Verlauf der Zeit – Vorraussagen über das Verhalten treffen und Änderungen des Systems beobachten zu können [40, S. 9ff]. Die grundlegende Motivation stellt die daraus gewonnene wissenschaftliche Erkenntnis dar, die nicht aus der bloßen Systemkenntnis gewonnen werden kann [8, S. 17].

Merkmal	Ausprägung			
Anwendungsebene	Programmiersprachen	Modelle (grafisch, textlich)	Simulationsysteme	
Systemtyp	diskret	kontinuierlich	hybrid	
Sichtweise	ereignisorientiert	transaktionsorientiert	prozessorientiert	aktivitätsorientiert
Modellierungskonzept	Modell-Bausteine	Automatentheorie	wissensbasierte Systeme	
	Warteschlangenszenarien	Objektorientierung	grafische Formalismen	
Grad der Spezialisierung	universal	domänenspezifisch	Teilbereich einer Anwendung	

Abbildung 3.3: Morphologischer Kasten für die Einordnung von Simulationstools [nach 41, S. 252]

Um ein Simulationstool zu klassifizieren, wird ein Morphologischer Kasten verwendet, welcher Ausprägungen verschiedener Merkmale einer Software gegenüberstellt [41, S. 252] (Abbildung 3.3). Im folgende wird das zu entwickelnde OM-Simulationstool entsprechend eingeordnet (Abbildung 3.4).

- Bei dem hier zu entwickelnden Tool handelt es sich um die **Anwendungsebene** eines fertigen Simulationssystems. Der Anwender muss keinerlei Programmierkenntnisse besitzen und auch die Modellbildung ist bereits abgeschlossen, da das Modell „6+1“ komplett in das Tool hinein programmiert sein wird.
- Durch den Radonkonzentrationsverlauf – abgebildet auf die Zeit – wird ein stetiger Prozess simuliert. Daher ist der **Systemtyp** kontinuierlich, weil durch eine Abstraktion oder Terminierung der Zeitkomponente die Simulationen keinen fachlichen Wert mehr aufweisen würden.
- Mit dem Merkmal der **Sichtweise** weist der Morphologische Kasten für die Klassifizierung von Simulationssoftware noch eine Schwachstelle auf. So sind

die vier Eigenschaften der Ereignis-, Transaktions-, Prozess- und Aktivitätsorientierung alle samt Ausprägungen ereignis- und zeitdiskreter Simulationen [47, S. 6]. Somit trifft keine der Ausprägung auf das hier zu entwickelnde Tool zu.

- Das **Modellierungskonzept** entspricht dem der Objektorientierung. Alle Systemkomponenten der Simulationen werden als Objekte in der Software angelegt: Gebäude, Räume, Messkampagnen und sogar gesamte Simulationsläufe (vgl. Abschnitt 7.2.3).
- Der **Grad der Spezialisierung** ist hier maximal. Das Software-Werkzeug ist einzig für den konkreten Anwendungsfall der orientierenden Messungen bestimmt. Um das Tool für andere Anwendungsbereiche zu verwenden, muss es auf der Ebene der Programmiersprache weiter oder gar neu entwickelt werden.

Merkmal	Ausprägung			
Anwendungsebene	Programmiersprachen	Modelle (grafisch, textlich)	Simulationsysteme	
Systemtyp	diskret	kontinuierlich		hybrid
Sichtweise	ereignisorientiert	transaktionsorientiert	prozessorientiert	aktivitätsorientiert
Modellierungskonzept	Modell-Bausteine	Automatentheorie		wissensbasierte Systeme
	Warteschlangenszenarien	Objektorientierung		grafische Formalismen
Grad der Spezialisierung	universal	domänenspezifisch		Teilbereich einer Anwendung

Abbildung 3.4: Einordnung des OM-Simulationstools (violett) in den Morphologischen Kasten [nach 41, S. 252]

Wesentliche Vorteile der Simulationstechnologie sind sowohl die finanziellen und zeitlichen Einsparmöglichkeiten als auch die Minimierung von bestimmten Risiken, die auftreten könnten, wenn reale Versuche gestartet werden [38, S. 471]. Wichtig für erfolgreiche Simulationsprojekte sind unter anderem eine konkrete statistische Planung sowie die detaillierte Dokumentation der Vorgehensweise und Ergebnisse [38, S. 476]. Das hier vorliegende Werk dient somit der kompletten technischen Dokumentation des Simulationstools.

Kapitel 4

Anforderungsanalyse

Das Messprotokoll „6+1“ sieht vor, durch sechs Messungen an sechs Tagen in bewohnten Räumen und einer Messung an einem Tag in einem unbewohnten Raum mit höchst anzunehmender Radon-Konzentration einen ersten Eindruck zu bekommen, ob das Gebäude ein Radon-Problem hat. Dazu wird jeweils der Mittelwert und das Maxima für alle bewohnten Räume sowie für den unbewohnten Raum ermittelt [25, S. 10].

Für das zu entwickelnde Software-Werkzeug wurden folgende Anforderungen grundlegend definiert.

- Vorliegende Messdaten aus Gebäuden mit ihren verschiedenen Räumen und den jeweiligen Messreihen müssen so auf- und vorbereitet werden, dass diese einer automatisierten Auswertung durch die Software zugeführt werden können.
- Es gilt eine automatisierte Routine zu schreiben, die alle möglichen Variationen gemäß des Messprotokolls „6+1“ für die vorhandenen Daten erstellt und gegebenenfalls virtuelle Messkampagnen simuliert.
- Bei der Auswertung der simulierten Messkampagnen sind einfache statistische Kenngrößen jeder einzelnen Variation sowie über alle Variationen hinweg zu erfassen und geeignet auszugeben.
- Das Programm soll es zudem ermöglichen, von gezielt ausgewählten Raumvariationen den Konzentrationsverlauf und statistische Kenngrößen grafisch darzustellen und in gängigen Dateiformaten, zum Beispiel PDF oder TIFF, auszugeben. Des Weiteren sollen die dieser Auswertung zu Grunde liegenden, kombinierten Daten in sinnvollen Formaten exportierbar sein.
- Import und Auswertung zukünftig erhobener Messdaten soll anwenderfreundlich möglich sein.

4.1 Verarbeitung realer Messdaten

Die zu simulierenden Messkampagnen sollen Abbilder möglicher realer Messungen sein, bei denen die Messgeräte entsprechend des Protokolls „6+1“ von Raum zu Raum bewegt werden. Während die Messkampagnen in der Software virtuell sind, sollen diese aber auf realen Messdaten aus radonbelasteten Gebäuden basieren, um die Vergleichbarkeit der Simulationsergebnisse mit bereits vorhandenen Evaluationsergebnissen zu gewährleisten. Dadurch wird auch die konzeptionelle Kluft zwischen Modell und Realität auf ein Minimum reduziert [40, S. 20]. Radonmesswerte und -messreihen sind per Definition Umweltinformationen nach Rautenstrauch [44, S. 8], da die Daten in Bezug zu Raum und Zeit stehen und einen fachlichen Kontext – den des radonbelasteten Gebäudes – gesetzt werden.

Die Daten, wie zum Beispiel im Abschlussbericht zum Vorhaben „StSch 4534“ [3], liegen vorwiegend im MICROSOFT-EXCEL-Format (XLS) vor. Sie sind meist auf viele einzelne Tabellen und Datenblätter aufgeteilt und weisen eine unterschiedliche Granularität der erfassten Messdaten vor. Es muss eine Datenaufbereitung stattfinden, die die Daten in ein geeignetes, einheitliches Format bringt und die Datensätze im Sinne der Datenmodellierung normalisiert.

Zu diesem Zeitpunkt denkbare Dateiformate für die sinnvolle Weiterverarbeitung von größeren, strukturierten Datenmengen wären unter anderem COMMA SEPARATED VALUES (CSV) oder EXTENSIBLE MARKUP LANGUAGE (XML).

- XML-Dateien ermöglichen das hierarchische Speichern von verschiedenen Daten und Datentypen in unterschiedlichen Entitäten [9].
- CSV-Dateien hingegen ermöglichen strukturiertes Speichern von tabellarisch vorliegenden Daten in einfachen Textdateien [46].

Welcher Datentyp der sinnvollste für dieses Vorhaben ist, wird weitestgehend im Kapitel 5 *Spezifikation* diskutiert.

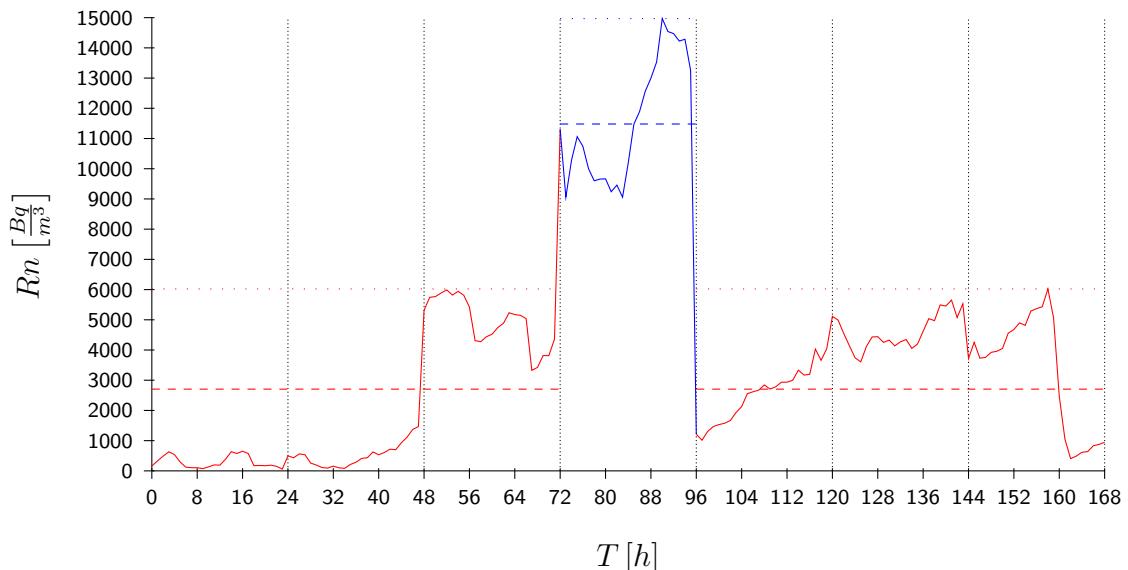
4.2 Generierung virtueller Messkampagnen

Wie bereits eingangs erwähnt setzen sich die simulierten virtuellen Messkampagnen aus sechs Messungen in regulär bewohnten Räumen und einer Messung in einem bodennahem Raum mit ansteigender Radon-Höchstkonzentration. Der Einfachheit halber werden diese Räume in diesem Dokument mit den Buchstaben **R** (*en. room*) und **C** (*en. cellar*) abgekürzt.

- Bewohnte Räume sind zum Beispiel Wohnzimmer, Schlafzimmer, Bäder, etc. in denen sich Anwohner täglich zum Teil über längere Zeit aufhalten.

- Bodennahe Räume beziehungsweise Kellerräume sind oft unbewohnte Räume, bei denen davon ausgegangen wird, dass in diesen durch das Mauerwerk, Fundament oder Rohrleitungen sowie Kabelschächte Radon direkt aus dem Boden in das Gebäude eindringen kann und sich dort sehr schnell anreichert.
- Vernachlässigt werden für die Messkampagnen Räume, in die nicht in der Nähe des Erdbodens liegen oder/und nicht direkt bewohnt sind und daher für die Simulationen nicht weiter relevant sind. Dazu zählen zum Beispiel Flure, Treppenhäuser oder Dachböden. (Diese Räume tauchen gelegentlich unter der Abkürzung M (*en. miscellaneous*) in dieser Arbeit auf.)

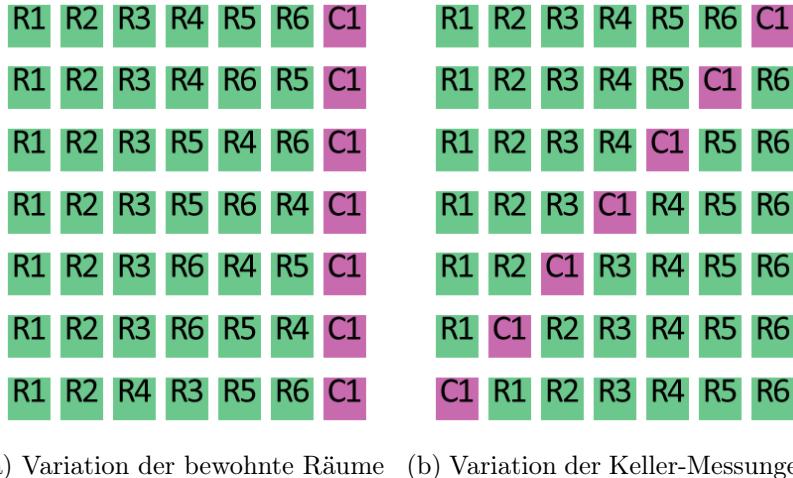
Funktion 4.1 veranschaulicht eine virtuelle Messkampagne. Am ersten Tag finden 24 Stunden lang Messungen im Esszimmer (*hier: R1*) statt, am zweiten Tag in der Waschküche im Keller (*C1*) und darauf jeweils Tagesmessungen im Arbeitszimmer (*R2*), Badezimmer (*R3*), Schlafzimmer (*R4*), in der Küche (*R5*), sowie im Wohnzimmer (*R6*) des zu untersuchenden Gebäudes. Dabei werden separat Maxima und arithmetische Mittel über jeweils alle bewohnten und unbewohnten Räume erhoben (sogenannte *batch means* zur Untersuchung von einzelnen Teilen der Simulation [42, S. 186f]).



Funktion 4.1: Beispiel einer Messkampagne über sieben Tage in sechs bewohnten Räumen und einem Kellerraum (Tag 4) unter Verwendung fiktiver Messdaten.

Dabei ist zu beachten, dass – egal für wieviele Räume Messwerte vorliegen – immer insgesamt an sechs Tagen in bewohnten Räumen und ein Tag im Keller gemessen werden muss. Liegen Messdaten für weniger als 6 bewohnte Räume vor, so müssen einzelne Räume gegebenenfalls doppelt gemessen werden. Dabei gilt, dass Messungen über zwei Tage im gleichen Raum aber nicht gestückelt werden dürfen sondern immer zusammenhängend bleiben müssen.

Die Abbildung 4.1 veranschaulicht die zwei Komplexitätsdimensionen der Simulationen. So kann einerseits die Reihenfolge der Raummessungen variiert werden (4.1a)



(a) Variation der bewohnte Räume (b) Variation der Keller-Messungen

Abbildung 4.1: Komplexität möglicher Variationen von Messungskampagnen

und andererseits auch die Position der Kellermessung (4.1b). Nähere Betrachtungen zur Komplexität der Variationen und zum Umfang der Simulationen werden im Abschnitt 5.3 *Größenordnung der Simulationen* erörtert.

4.3 Statistische Auswertung

Kernstück ist – neben der Simulation von Messkampagnen – die Berechnung und Ausgabe von akkurate statistischen Kenngrößen. In den Anforderungen wurde bereits definiert, dass verscheidene Statistiken geführt werden müssen. Dies soll nun nochmal zusammengefasst werden.

- Für jede einzelne virtuelle Messkampagne werden folgende Größen ermittelt:
 - Arithmetischer Mittelwert (**AM**)
 - Geometrischer Mittelwert (**GM**)
 - Median (**Q50**)
 - Maximum (**MAX**)
- Über die gesamte Simulation hinweg werden über die vier zuvor genannten Kennzahlen jeweils eine zusammenfassende Statistik mit den folgenden Ergebnissen berechnet:
 - Arithmetischer Mittelwert
 - Arithmetische Standardabweichung (**SD**)
 - Variationskoeffizient (**CV**)
 - Geometrischer Mittelwert
 - Geometrische Standardabweichung (**GSD**)

- Perzentil 5 (Q5)
- Median
- Perzentil 95 (Q95)
- Perzentilsabweichung (QD)
- relative Perzentilsabweichung (RQD)

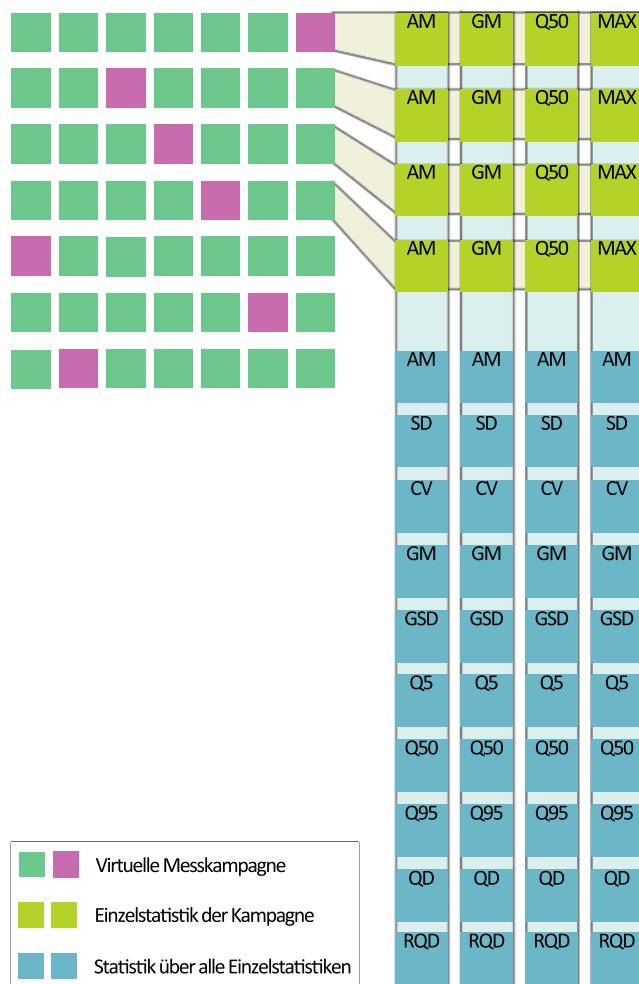


Abbildung 4.2: Übersicht der zu berechnenden statistischen Größen

Dies ergibt für jede einzelne Messkampagne jeweils vier statistische Größen und im Falle einer erfolgreichen Simulation insgesamt 40 Ergebnisse ($4 \cdot 10 = 40$), die dann fachgerecht ausgewertet werden können (Abbildung 4.2).

Die Vorgehensweise und weitere Grundlagen zur Berechnung der einzelnen Werte für die Auswertung werden in Abschnitt 5.4 *Statistische Grundlagen* näher untersucht.

Kapitel 5

Spezifikation

Um die Anforderungen für die Software umzusetzen bedarf es einer Spezifikationsphase, die besonderes Augenmerk auf alle denkbaren Details legt, da viele der folgenden Vorbetrachtungen im einzelnen schon die logischen Gedankenschritte der zu implementierenden Algorithmen und Programmteile aufweist.

5.1 Modellierung

Zunächst müssen die Anforderungen aus Kapitel 4 in ein grobes Modell umgesetzt werden, das die künftige Funktionalität des Simulationstools beschreibt [21, S. 42f]. Der Übersichtlichkeit halber wurde hierfür ein einfaches Ablaufdiagramm (Abbildung 5.1) erstellt, dass sechs grundlegende Funktionalitäten beschreibt:

1. Die Daten müssen zunächst vorbereitet werden, um automatisiert von der Software verarbeitet werden zu können (vgl. Abschnitt 5.2). Dieser Schritt muss manuell durchgeführt werden, da die Daten nicht immer konsistent im gleichen Ausgangsformat vorliegen.
2. Die aufbereiteten Daten werden dann von der Software eingelesen und geparsst, d.h. die Daten werden von der Software sofort in brauchbare Teildaten zerlegt.
3. Das Tool legt nun Objekte für die einzelnen Räume an, die später für die Simulationen verwendet werden sollen.
4. Die Raum-Objekte werden nun in allen möglichen Variationen kombiniert und somit virtuelle Messkampagnen generiert (vgl. Abschnitt 5.3).
5. Um den wissenschaftlichen Anspruch der Software zu wahren, muss jeder Schritt nachvollziehbar protokolliert werden. Dazu werden während der Simulation Log-Dateien angelegt.
6. Für die Auswertung der Simulationen müssen statistische Kenngrößen in geeigneten Format ausgegeben werden (vgl. Abschnitt 4.3).

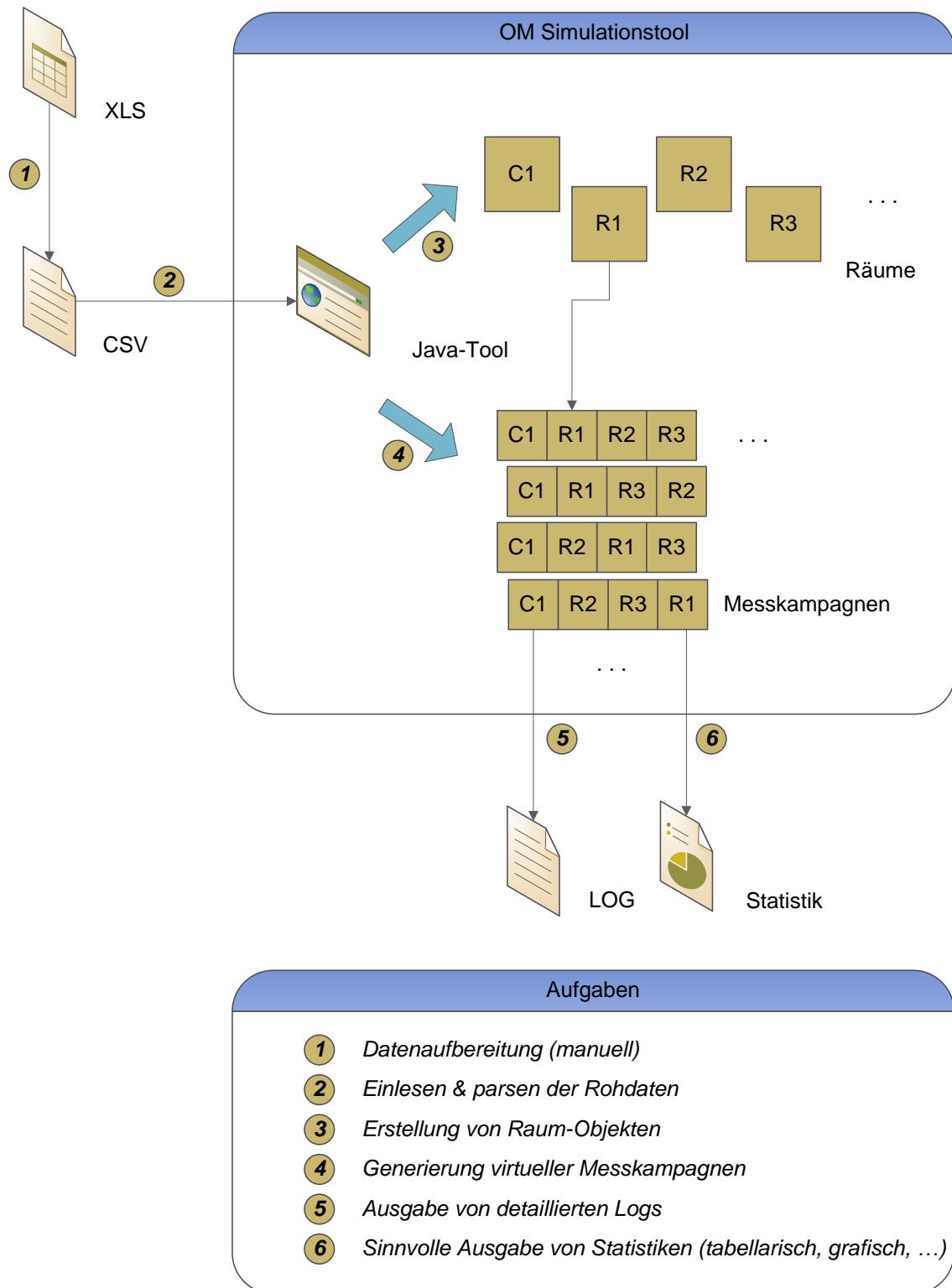


Abbildung 5.1: Ablaufdiagramm für das Simulationstool

5.2 Datenaufbereitung im CSV-Format

Für das zu entwickelnde Programm wird CSV als Format für die zu importierenden Daten der gemessenden Gebäude gewählt. Das CSV-Datenformat wird schon seit geraumer Zeit für die Speicherung und den einfachen Austausch tabellarischer Daten zwischen verschiedensten Statistik- und Tabellenkalkulationsprogrammen genutzt [46, S. 1]. Im allgemeinen bietet es daher drei grundlegende Vorteile gegenüber anderen Formaten wie zum Beispiel XML:

- CSV-Dateien können mit jedem gängigen Tabellenkalkulationsprogramm (zum Beispiel MICROSOFT Excel, LibreOffice Spreadsheet oder GOOGLE Docs Spreadsheet) erstellt werden.
- Der Import von CSV-Daten ist aufgrund seiner weiten Verbreitung in nahezu jedem Statistikprogramm möglich und bietet hier die Chance, die Daten auch in anderen, unabhängigen Programmen zu überprüfen.
- Die einfache, durch Kommata oder Semikola getrennte, Dateistruktur vereinfacht sowohl das manuelle Überprüfen der Textdaten durch Menschen als auch das einlesen und parsen der Daten durch speziell entwickelte Softwareprogramme.

5.2.1 Vorgehensweise

Für das Simulationstool werden die Daten wie folgt aufbereitet.

```
ID;C1;C2;R1;R2;R3;R4;R5;R6;R7;M1;M2
0;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn
1;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn
2;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn;Rn
...
...
```

Die Zeilen der Datei sind die Datenreihen der Tabelle und die Datenspalten werden durch Semikola getrennt, um Konflikte mit Komma-Werten vorzubeugen und somit das parsen zu erleichtern.

- Die erste Zeile (1) des Datenblattes stellt den Tabellenkopf dar.
 - Die erste Spalte (A) beinhaltet die laufende Nummer der Stunde der Messung ab Beginn, namentlich definiert als ID, beginnend mit 0.
 - Die Spalten ab B beinhalten die eindeutigen Bezeichner für die Räume des Gebäudes. Diese sehen wie folgend aus:

- * C1 bis Cn stehen für Kellerräume bzw. unbewohnte Räume mit annehmender höchster Radon-Belastung, z.B. Waschküchen, Heizkeller, Vorratskammern, etc.
- * R1 bis Rn stehen für bewohnte bzw. für die Gesundheit der Bewohner besonders relevante Räume, z.B. Schlafzimmer, Wohnzimmer, Kinderzimmer, etc.
- * M1 bis Mn stehen für sonstige, weniger für die Gesundheit relevante Räume, in denen sich die Bewohner seltener aufhalten, zum Beispiel Flure, Dachböden, etc.
- Ab der zweiten Zeile (2 bis n) werden die einzelnen parallelen Messreihen der Radonkonzentration eingetragen. Zu beachten ist hierbei, dass bei Messreihen, deren Granularität größer ist, als ein Wert pro Stunde, ein Mittelwert über eben eine Stunde gebildet werden muss, bevor die Werte in die neue Datei übernommen werden (siehe auch Abschnitt 5.2.2 Hinweise).
- Es sollten im Idealfall mindestens 336 Zeilen (14 Tage) in der Tabelle enthalten sein, um genügend Variationen simulieren zu können. Wichtig ist hierbei, dass darauf geachtet wird, dass alle Messreihen zum gleichen Zeitpunkt beginnen. Oft werden Messgeräte nicht auf die Stunde genau gleichzeitig aufgestellt und aktiviert. In diesem Fall sollten alle Messwerte zu Beginn entfernt werden, bis zu dem Zeitpunkt, an dem für alle (relevanten) Räume Messdaten vorliegen.

Beispiel einer CSV-Datei aus Objekt 8 vom Abschlussbericht zum Vorhaben *StSch-4534* [3], darin 11 Räume (2 C, 7 R, 2 M) zeitaufwöndig gemessen im August und September 2008:

```
ID;C1;C2;R1;R2;R3;R4;R5;R6;R7;M1;M2
0;2310;1032;1456;946;466;1496;192;805;2135;1337;490
1;4701;1488;2037;1529;661;1692;85;1782;2647;2268;492
2;6024;360;1928;1713;670;985;85;2041;1190;2893;456
...
399;10553;4439;3978;221;3225;3988;6837;682;5871;7099;1520
400;10996;4993;3895;368;3696;3985;7576;1987;5866;7355;1666
401;10317;5124;4227;645;3759;5187;6610;2684;6369;7291;1838
```

Die Raumbezeichner entsprechen folgenden Räumen:

- C1 - Felsenkeller, Erdgeschoss
- C2 - Heizungsraum, Erdgeschoss *
- R1 - Esszimmer, Erdgeschoss
- R2 - Schlafraum, 1. Obergeschoss

- R3 - Wohnzimmer, 1. Obergeschoss
- R4 - Kinderzimmer klein, 1. Obergeschoss *
- R5 - Gästebad, 1. Obergeschoss
- R6 - Kinderzimmer groß, 1. Obergeschoss
- R7 - Küche, Erdgeschoss *
- M1 - Flur, Erdgeschoss *
- M2 - Dachboden, 2. Obergeschoss *

*) Mittelwert über 1 Stunde gebildet.

Die Messwertreihe mit der Nummer ID=0 bezieht sich auf die erste Messung am 25. August 2008 um ca. 16 Uhr. Die Messreihe um 15 Uhr musste entfernt werden, weil zu dieser Zeit noch nicht alle Messgeräte aufgestellt und aktiviert waren. Die Messwertreihe mit der Nummer ID=401 bezieht sich auf die letzte Messung am 11. September 2008 um 10 Uhr. Die Messreihen um 11 und 12 Uhr wurden entfernt, weil die ersten Messgeräte schon deaktiviert oder abgebaut worden waren. 402 Messreihen entsprechen 402 Stunden, was 16 Tagen und 18 Stunden entspricht.

Die CSV-Datei wäre nun fertig für die Weiterverarbeitung mit geeigneter Software.

5.2.2 Hinweise

Bei der Erstellung von CSV-Dateien sind neben den zuvor genannten Spezifikationen folgende Hinweise zu beachten.

- Die Granularität der vorhandenen Messwertreihen muss immer dahingehend überprüft werden, so dass diese nur aus einem Messwert pro Stunde bestehen. Sollte die Granularität größer sein, müssen Mittelwerte über jeweils eine Stunde gebildet werden bevor die Daten exportiert werden können.
- Die Daten müssen in der Einheit $\frac{Bq}{m^3}$ (*Becquerel pro Kubikmeter*) vorliegen. Kommastellen können gerundet werden, da die Genauigkeit von $1\frac{Bq}{m^3}$ ausreichend ist und viele Radonmessgeräte eine Nachweisgrenze von mehr als $1\frac{Bq}{m^3}$ aufweisen. (Beispielsweise weist der ATMOS 12 DPX eine kleinste nachweisbare Aktivität (*en. minimal detectable activity - MDA*) von $2\frac{Bq}{m^3}$ bei Messungen über 60 Minuten auf [65, S. 2].)
- Abschließend sollten die CSV-Daten nochmals in einem Texteditor überprüft werden, ob die Datei korrekt formatiert wurde. Häufige Fehler sind Semikola am Zeilenende, leere Datensätze am Ende der Datei, die lediglich aus Semikola bestehen, sowie in Anführungszeichen gesetzte Datenfelder.

5.3 Größenordnung der Simulationen

Nun gilt es zunächst zu ermitteln, ob es überhaupt technisch möglich ist, komplexe systematische Simulationen aller verfügbarer Variationen durchzuführen oder ob gegebenenfalls probabilistische Auswahlverfahren eine begrenzte Anzahl zufälliger Variationen generieren. Dazu müssen Überlegungen zu den Algorithmen und Berechnungen der möglichen Gesamtvariationen aufgestellt werden. Für diese Vor betrachtung wird ein optimaler Messzeitraum von 14 Tagen gewählt, um die Werte der verschiedenen, betrachteten Gebäudetypen vergleichen zu können.

Für Gebäude mit sechs oder mehr Räumen ($R \geq 6$) berechnet sich die Anzahl der höchstmöglichen Variationen (N) mit einem Keller ($C = 1$), sieben möglichen Positionen der Kellermessung ($P = 7$, vergleiche Abbildung 5.2) wie folgt. Für einen Erhebungszeitraum von 14 Tagen (336 Stunden) ergeben sich 169 Startzeitpunkte der Schnellmessungen, da in den letzten 167 Stunden der Messungen die simulierten 7-Tages-Kampagnen über das Ende der Messreihen hinaus laufen würde ($T = 14 \cdot 24 - 167 = 169$).

$$N_{R \geq 6} = \frac{R!}{(R-6)!} \cdot C \cdot P \cdot T \quad (5.1)$$

$$N_{R \geq 6} = \frac{R!}{(R-6)!} \cdot 1 \cdot 7 \cdot 169 \quad (5.2)$$

$$N_{R \geq 6} = \frac{R!}{(R-6)!} \cdot 1183 \quad (5.3)$$

Daraus ergeben sich für gebäude mit mehr als sechs Räumen folgende Variations-



Abbildung 5.2: Positionen der Kellermessungen $P = 7$ bei Gebäuden mit $R \geq 6$ zahlen (Tabelle 5.1).

R	N
10	178.869.600
9	71.547.840
8	23.849.280
7	5.962.320
6	851.760

Tabelle 5.1: Anzahl möglicher Variationen bei $R \geq 6$

Bei Gebäuden mit weniger als sechs gemessenen bewohnten Räumen müssen einzelne Räume doppelt gemessen werden, so dass insgesamt an sechs Tagen in bewohnten Räumen Daten erhoben werden. Die Berechnung der höchstmöglichen Variationen N in Abhängigkeit der Anzahl der Räume R , mit $R < 6$, ist nicht trivial. Da nicht

beliebig in verschiedenen Räumen doppelt gemessen werden soll, sondern möglichst zusammenhängend, kommt als Variable die Anzahl der möglichen Positionen von Mehrfachmessungen (M) hinzu.

$$N_{R=6} = R! \cdot M \cdot C \cdot P \cdot T \quad (5.4)$$

Für $R = 5$ gibt es fünf mögliche Positionen der Doppelmessungen ($M = 5$, vergleiche Abbildung 5.3) und sechs mögliche Positionen für die Kellermessungen ($P = 6$).

C	R	R	C	R	C	R	C	R	C	R	C
C	R	C	R	R	C	R	C	R	C	R	C
C	R	C	R	C	R	R	C	R	C	R	C
C	R	C	R	C	R	C	R	R	C	R	C
C	R	C	R	C	R	C	R	C	R	R	C

Abbildung 5.3: Mögliche Kellermessungen $P = 6$ und Mehrfachmessungen $M = 5$ bei Gebäuden mit $R = 5$

$$N_{R=5} = 5! \cdot 5 \cdot 1 \cdot 6 \cdot 169 = 608400 \quad (5.5)$$

Für $R = 4$ gibt es sechs mögliche Kombinationen, um die Doppelmessungen zu platzieren ($M = 6$, vergleiche Abbildung 5.4), aber nur fünf mögliche Positionen für die Kellermessung ($P = 5$).

C	R	R	C	R	R	C	R	C	R	C
C	R	R	C	R	C	R	R	C	R	C
C	R	R	C	R	C	R	C	R	R	C
C	R	C	R	R	C	R	R	C	R	C
C	R	C	R	R	C	R	C	R	R	C
C	R	C	R	C	R	R	C	R	R	C

Abbildung 5.4: Kellermessungen $P = 5$ und Mehrfachmessungen $M = 6$ mit $R = 4$

$$N_{R=4} = 4! \cdot 6 \cdot 1 \cdot 5 \cdot 169 = 121680 \quad (5.6)$$

Für $R = 3$, $R = 2$ und $R = 1$ gibt es nur eine mögliche „Kombination“, um Mehrfachmessungen zu platzieren ($M = 1$) und die Anzahl der möglichen Positionen für die Kellermessungen ist lediglich $P = R + 1$ (Abbildung 5.5).

$$N_{R=3} = 3! \cdot 1 \cdot 1 \cdot 4 \cdot 169 = 4056 \quad (5.7)$$

$$N_{R=2} = 2! \cdot 1 \cdot 1 \cdot 3 \cdot 169 = 1014 \quad (5.8)$$

$$N_{R=1} = 1! \cdot 1 \cdot 1 \cdot 2 \cdot 169 = 338 \quad (5.9)$$

Abschließend nochmals zur Übersicht die Anzahl möglicher Variationen N in Ab-

C | R | R | C | R | R | C | R | R | C

(a) Kellermessungen $P = 4$ mit
 $R = 3$

C | R | R | R | C | R | R | R | C

(b) Kellermessungen $P = 3$
mit $R = 2$

C | R | R | R | R | R | R | C

(c) Kellermessungen $P = 2$
mit $R = 1$

Abbildung 5.5: Positionen der Kellermessungen bei $R = 3$, $R = 2$ und $R = 1$

hängigkeit der Anzahl der Räume R berechnet auf Grundlage eines 14-tägigen Datensatzes mit einem unbewohntem Raum C (Tabelle 5.2).

R	N
10	178.869.600
9	71.547.840
8	23.849.280
7	5.962.320
6	851.760
5	608.400
4	121.680
3	4.056
2	1.014
1	338

Tabelle 5.2: Anzahl möglicher Variationen in Abhängigkeit von der Anzahl der bewohnten Räume

5.4 Statistische Grundlagen

Um eine sinnvolle Auswertung der generierten, virtuellen Messkampagnen vornehmen zu können, sollten an dieser Stelle Bedeutungen und Formeln zur Berechnung verschiedener relevanter statistischer Kenngrößen zusammengefasst werden. Da die Formeln später hilfreich bei der Implementierung der verschiedenen Algorithmen zur Berechnung der statistischen Auswertung sind, werden einige komplexere Rechenwege vorab schon einmal als Pseudocode notiert [20].

- Die Anzahl n gibt die Gesamtheit der vorliegenden Messwerte einer Messreihe an und entspricht somit der Anzahl der gemessenen Stunden in einem

Gebäude.

$$n \in \mathbb{N} \quad (5.10)$$

- Die Summe der Messwerte SUM gibt das Ergebnis der Addition aller Werte an und dient nur als Zwischenschritt zur Berechnung der Mittelwerte.

$$SUM = x_1 + x_2 + \dots + x_n = \sum_{i=1}^n x_i \quad (5.11)$$

- Die Spannweite R gibt die Breite der Werteverteilung zwischen Minimum und Maximum an. Da diese Größe jedoch sehr anfällig für Ausreißer-Werte ist, wird im weiteren Verlauf noch der Perzentilsabstand betrachtet.

$$R = x_{max} - x_{min} \quad (5.12)$$

- Der Mittelwert AM gibt den einfachen arithmetischen Durchschnittswert an. Auch diese Kenngröße wird durch Ausreißer stark beeinflusst, weswegen später noch der geometrische Mittelwert betrachtet wird.

$$AM = \frac{SUM}{n} = \frac{1}{n} \cdot \sum_{i=1}^n x_i \quad (5.13)$$

```

FOR i = 0 TO n DO
    SET sum = sum + x[i]
ENDFOR
SET am = sum / n

```

Pseudocode 5.1: Berechnung des arithmetischen Mittels

- Die Varianz VAR gibt die mittlere quadratische Abweichung aller Messwerte vom arithmetischen Mittel an. Sie ist selbst nicht interpretierbar und dient nur als Grundlage zur Berechnung der Standardabweichung [11, S. 47].

$$VAR = \frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - AM)^2 \quad (5.14)$$

- Die Standardabweichung SD gibt die durchschnittliche Abweichung der Messwerte vom arithmetischen Mittelwert an.

$$SD = \sqrt{VAR} = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - AM)^2} \quad (5.15)$$

```

FOR i = 0 TO n DO
    SET var = var + (x[i] - am) * (x[i] - am)
ENDFOR
SET var = var / (n - 1)
SET sd = SQRT(var)

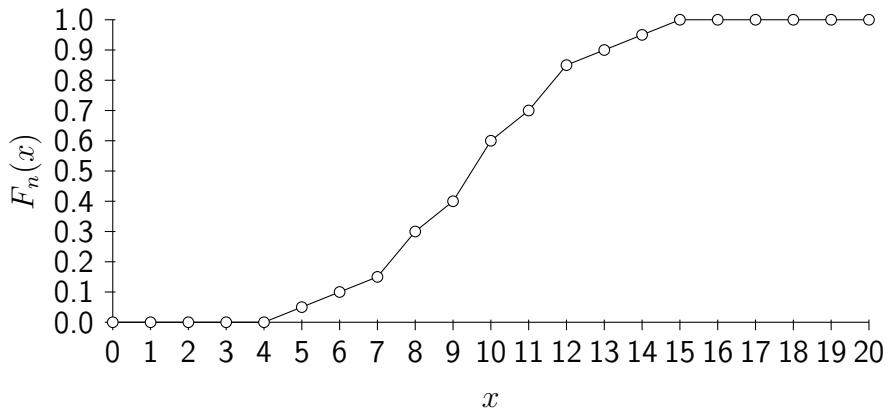
```

Pseudocode 5.2: Berechnung der Standardabweichung

- Der Variationskoeffizient CV setzt die Standardabweichung ins Verhältnis zum arithmetischen Mittel und dient als relatives Streuungsmaß [11, S. 49].

$$CV = \frac{\sqrt{VAR}}{AM} = \frac{SD}{AM} \quad (5.16)$$

- Die empirische Verteilungsfunktion $F_n(x)$ bildet die kumulierte relative Häufigkeitsverteilung ab [11, S. 27] (Funktion 5.1).

Funktion 5.1: Empirische Verteilungsfunktion $F_n(x)$

- Die Perzentile Q_p geben Messwerte anhand ihrer Position p innerhalb der empirischen Verteilungsfunktion $F_n(x)$ an [42, S. 183].

$$F_n(p) = Q_p; \text{ mit } 0\% < p < 100\% \quad (5.17)$$

- Das Perzentil an der Position $p = 50\%$ wird als Median Q_{50} bezeichnet.
- Die Perzentile an den Positionen $p = 25\%$ und $p = 75\%$ werden als unteres Quartil Q_{25} und oberes Quartil Q_{75} bezeichnet.

- Der zentrale Perzentsabstand PA_p betrachtet die Spannweite zwischen oberen und unterem Perzentil und eliminiert Ausreißer in den Messwertreihen [11, S. 45].

$$PA_p = Q_{\frac{1+p}{2}} - Q_{\frac{1-p}{2}}; \text{ mit } 0\% < p < 100\% \quad (5.18)$$

- Der Quartilsabstand QA ist ein Spezialfall des Perzentilabstands PA_p mit $p = 50\%$.

$$QA = Q_{75} - Q_{25} \quad (5.19)$$

- Die Perzentilsabweichung QD und die relative Perzentilsabweichung RQD dienen als Streuungsmaße für die Verteilung der Messwerte, eliminieren aber im Gegensatz zur Spannweite und zum Variationskoeffizienten die Ausreißer.

$$QD = \frac{PA_p}{2} \quad (5.20)$$

$$RQD = \frac{PA_p}{2 \cdot Q_{50}} \quad (5.21)$$

- Der Mittelwert GM gibt den geometrischen Durchschnittswert an und interpretiert im Gegensatz zum arithmetischen Mittel das Produkt \prod anstatt der Summe \sum [11, S. 42].

$$GM = \sqrt[n]{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \sqrt[n]{\prod_{i=1}^n x_i}; \text{ mit } x_i > 0 \quad (5.22)$$

$$\text{Äquivalent dazu gilt: } \ln GM = \frac{1}{n} \cdot \sum_{i=1}^n \ln x_i \quad (5.23)$$

```

FOR i = 0 TO n DO
    SET gsum = gsum + LOG(x[i])
ENDFOR
SET gm = gsum / n
SET gm = EXP(gm)

```

Pseudocode 5.3: Berechnung des geometrischen Mittels

- Die geometrische Standardabweichung GSD ist ein gewichtetes Streuungsmaß der Abweichung vom geometrischen Mittelwert GM .

$$GSD = e^{\sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n \ln (x_i - GM)^2}} \quad (5.24)$$

$$\text{Beziehungsweise: } \ln GSD = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n \ln (x_i - GM)^2} \quad (5.25)$$

Nachdem nun die Formeln aufgestellt und definiert wurden, sollten sie problemlos in der Software als Algorithmen implementiert werden können. Einige der hierbei aufgestellten Ausdrücke stellen lediglich Teilschritte zur Berechnung endgültig benötigter Kenngrößen dar. Für eine Übersicht der tatsächlich zu berechneten Größen siehe Abschnitt 4.3 *Statistische Auswertung*.

```
FOR i = 0 TO n DO
    SET gvar = gvar + (LOG(x[i]) - LOG(gm)) * (LOG(x[i]) - LOG(gm))
ENDFOR
SET gsd = gvar / (n - 1)
SET gsd = SQRT(gsd)
SET gsd = EXP(gsd)
```

Pseudocode 5.4: Berechnung der geometrischen Standardabweichung

Kapitel 6

Konzept

Nach einer ausführlichen Analyse der Anforderungen und Spezifikationen des Simulationstools in den vorangegangenen Kapiteln müssen nun noch letzte grundlegende Entscheidungen getroffen werden, bevor die Software entworfen und implementiert werden kann. So ist immer noch die Frage der grundsätzlichen Systementscheidung offen und es sollten mögliche Vorgehensmodelle als Einzelentwickler ohne Team abgewogen werden.

6.1 Plattform und Entwicklungsumgebung

Die Wahl der Plattform und der Programmiersprache waren ein erster kritischer Diskussionspunkt in diesem Projekt.

Seitens der HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT BERLIN (HTW BERLIN) wurde eine Entwicklung im .NET-Framework von MICROSOFT in der Programmiersprache C# nahegelegt, da dies im Studiengang BETRIEBLICHE UMWELT-INFORMATIK (BUI) der zur Zeit gängige Standard ist. Darüber hinaus könnte das Framework EMPORER verwendet werden [83], welches eine pluginbasierte Architektur bietet und für die schnelle und einfache Implementierung von Betrieblichen Umweltinformationssystemen (BUIS) zur Stoffstromanalyse und zu Simulationszwecken entwickelt wurde [43, S. 49ff].

Seitens des BUNDESAMTES FÜR STRAHLENSCHUTZ (BfS) wurde aus verschiedenen Gründen eine plattformunabhängige Programmierung in Java empfohlen, um die Einsatzfähigkeit der Software nicht auf Windows-Rechner zu beschrenken.

6.1.1 Plattformunabhängige Entwicklung

Trotz der immernoch unangetasteten Führungsposition von MICROSOFT WINDOWS auf dem Betriebssystememarkt (Tabelle 6.1) wird sich an dieser Stelle für eine plattformunabhängige Lösung entscheiden, da davon ausgegangen werden muss, dass –

gerade im wissenschaftlichen Kontext der durchzuführenden Simulationen – durchaus nicht immer auf standardisierte Bürorechner zurückgegriffen werden kann.

Rang	Betriebssystem	Anfragen	Anteil
1	Windows	107.264.160.000	73,38%
2	Mac OS	12.297.000.000	8,41%
3	iOS (iPad, iPhone)	10.895.340.000	7,46%
4	Linux (inklusive Android)	7.388.850.000	5,05%
5	Blackberry	1.179.810.000	0,81%

Tabelle 6.1: Die populärsten Plattformen [82]

6.1.2 Programmiersprache JAVA

Aufgrund der Entscheidung für eine plattformunabhängige Lösung fiel die Wahl für die Programmiersprache Java nicht all zu schwer. So ist Java nicht nur die populärste objektorientierte Programmiersprache seit über 10 Jahren [79] (Tabelle 6.2) sondern stellt auch mit der Java Virtual Machine (JVM) einen für die meisten Plattformen (Windows, Mac OS, Linux, SunOS, et cetera.) verfügbaren Interpreter bereit [5, S. 23, 26f].

Rang	Sprache	Anteil 2012	Differenz 2011
1	C	17,555%	+1,39%
2	Java	17,026%	-2,02%
3	C++	8,896%	-0,33%
4	Objective-C	8,236%	+3,85%
5	C#	7,348%	+0,16%

Tabelle 6.2: Die populärsten Programmiersprachen [79]

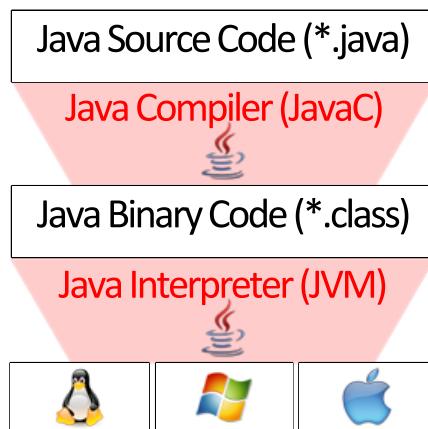


Abbildung 6.1: Trennung von Compiler und Interpreter in Java

Java ist eine Simulationssprache [39, S.144ff] und bietet mit der Objektorientierung ein *natürliches* Rahmenwerk zur Modellbildung der Welt [31, S.55]. Durch Objekte,

ihre Eigenschaften und Funktionen wird die konzeptionelle Kluft zwischen Modell und der realen Welt minimiert.

Das besondere an Java ist die Trennung von Compiler und Interpreter (Abbildung 6.1). Demnach wird erst durch den Java Compiler der Quell-Code in einen universellen Byte-Code kompiliert (sogenannter *JVM-Code*). Dieser Code ist eine Art Zwischensprache, welche sich an Java orientiert, aber eine einfache Struktur – ähnlich wie Maschinencode – aufweist [10, S. 39]. Dieser JVM-Code wird dann durch die jeweilige virtuelle Maschine auf der entsprechenden Plattform interpretiert.

6.1.3 Entwicklungsumgebung ECLIPSE

ECLIPSE ist eine Integrierte Entwicklungsumgebung (IDE) und dient als universelle Plattform zur Entwicklung von Software [29, S. 6]. ECLIPSE selbst ist ein Open-Source-Projekt und komplett in Java geschrieben.

Wesentliche Vorteile sind der eingebaute Debugger, automatische Syntaxhervorhebung (*en. syntax highlighting*) sowie automatischer Code-Einzug (*en. indentation*) [5, S. 288ff]. Darüber hinaus bietet die Plugin-Architektur nahezu unbegrenzte Erweiterungsmöglichkeiten.

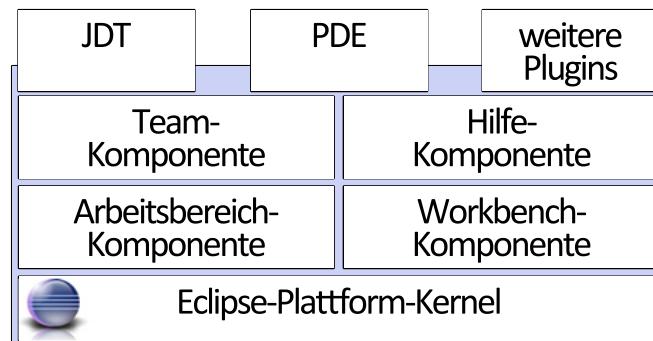


Abbildung 6.2: Architektur der ECLIPSE-IDE (mit Java-Plugins, [nach 29, S. 7])

So bieten die Plugins Java-IDE (*en. Java Development Toolkit*, JDT) und Plugin-Entwicklungsumgebung (*en. Plug-in Development Environment*, PDE) die wesentlichen Erweiterungen für Java-Entwickler [29, S. 6] (Abbildung 6.2).

6.2 Vorgehensmodell

Im folgenden werden Überlegungen zur Vorgehensweise der Softwareentwicklung diskutiert, die sich vor allem an der Größe des Entwickler-Teams und der Komplexität der zu entwickelnden Software orientieren.

6.2.1 Möglichkeiten als Einzelentwickler

Da das Simulationstool von nur einem Entwickler entworfen und implementiert wird, sind moderne Ansätze der agilen Softwareentwicklung wie zum Beispiel *Scrum* oder *Extreme Programming* (XP) zu vernachlässigen, da diese sich speziell mit der Optimierung der Arbeit und Kommunikation im Team aus Entwicklern, Designern und Testern beschäftigen [19, S. 182, 247] [18, S. 207ff].

Für die Entwicklung und Implementierung werden 500 Personenstunden veranschlagt, was bei einem Entwickler circa 3 Monate Entwicklungszeit bedeutet. Aufgrund dieser überschaubaren Komplexität des Projektes können verschiedene klassische Vorgehensmodelle angewendet werden. Dabei werden vor allem sequentielle Modelle wie das *Wasserfallmodell* oder das *V-Modell* und nichtsequentielle, zyklische wie das *Spiralmodell* oder das *Prototyping* unterschieden [21, S. 112].

Sequentielle Modelle zeichnen sich durch eine streng definierte Ablauffolge der einzelnen Entwicklungsphasen aus. Der Prozess der Entwicklung ist hierbei sehr aufwändig und zeichnet sich durch viele Probleme aus, unter anderem:

- Der *Stille-Post-Effekt* [22, S. 46] bewirkt, dass kleinere Unklarheiten in den ersten Phasen der Anforderungen zu großen, unüberschaubaren Abweichungen im Endprodukt führen können.
- Aufgrund des relativ unübersichtlichen Prozessmodells ist es schwer, zu späterem Zeitpunkt der Entwicklung, einen Einblick in untere Ebenen des Modells zu erhalten.

Bei zyklischen, iterativen Modellen wurden die Probleme der sequenziellen Herangehensweise adressiert und durch ein ständiges, wiederholtes durchlaufen aller Phasen flexibler auf Änderungen oder Abweichungen in Anforderungen reagiert [22, S. 48ff]. Derartige Vorgehensweisen werden auch als evolutionäre Modelle bezeichnet [21, S. 115 f].

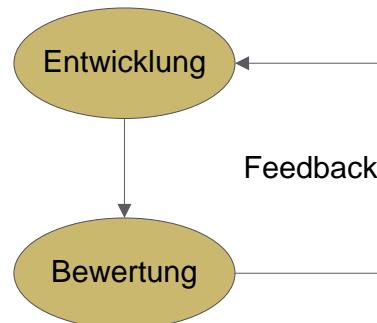


Abbildung 6.3: Grundprinzip der evolutionären Softwareentwicklung [nach 21, S. 115]

6.2.2 Das Prototyping

Das Prototyping ist ein evolutionäres Softwareentwicklungsparadigma, bei dem durch inkrementelle Entwicklungszyklen sukzessiv erste lauffähige Programmteile entworfen und implementiert werden. Dabei ist es von großer Bedeutung, dass diese Programmteile – sogenannte Prototypen – schnell operational sind, da diese die Grundlage für Diskussionen mit dem Anwender bilden. Somit ist es möglich, die Prototypen zu verwenden, um zunächst die Anforderungsdefinition inhaltlich zu vertiefen, um durch einzelnes Komponentendesign die Spezifikationen so detailliert wie möglich auszuarbeiten, oder um die Machbarkeit verschiedener Funktionalitäten frühzeitig zu ermitteln [6, S. 15ff]. Dadurch wird in erster Linie die Kommunikation zwischen Anwender und Entwickler optimiert [22, S. 46], da schnell erste Ergebnisse vorgestellt und evaluiert werden können.

Über die Verwendbarkeit der Prototypen im späteren Endsystem gibt es verschiedene Standpunkte in der Literatur. So schlagen einige Autoren vor, dass Prototypen schon den möglichen Kern eines künftigen Systems darstellen können [21, S. 116], andere hingegen warnen vor einer katastrophalen Softwarearchitektur und dadurch resultierende Sicherheitslücken sowie Performanzebußen [22, S. 46]. Letztendlich muss der Entwickler anhand der Komplexität und der Anforderungen des Projektes abwegen, ob Prototypen dem reinen Erkenntnisgewinn dienen sollen, oder sukzessive zum finalen Softwaresystem implementiert werden sollen. Oft besteht auch gar kein Unterschied zwischen Prototyp und Produkt [6, S. 17], da der Zyklus des Prototyping beliebig oft wiederholt werden kann, um flexibel stets auf neue Anforderungen reagieren zu können.

Das Prototyping soll in diesem Projekt Anwendung finden, da die Vorteile im relativ kleinen Team, bestehend aus drei Experten vom BUNDESAMT FÜR STRAHLENSCHUTZ und einem Entwickler von der HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT BERLIN, überwiegen (Tabelle 6.3).

6.2.3 Open-Source-Softwareentwicklung

Da die Entwicklung des Simulationstools nicht mit dem finalen Release beendet sein soll und um die Transparenz für den wissenschaftlichen Anspruch an die Software zu maximieren, wird das Programm unter einer Open-Source-Lizenz veröffentlicht und die Quellen ohne Einschränkungen offen gelegt. Somit ist es sowohl für Anwender als auch für potentielle Entwickler möglich, über den bisher veranschlagten Rahmen des Projektes hinaus, das Programm gebührenfrei und uneingeschränkt zu nutzen, zu testen und zu warten [33, S. 153]. Darüber hinaus wird durch die Veröffentlichung das Wissen und die Erfahrungen, die in der Software stecken, weiter verteilt und somit bewahrt, auch wenn die ursprünglichen Projektteilnehmer nicht mehr an dem Programm arbeiten oder gar nicht mehr auffindbar sind.

Als Lizenz wird die GNU GENERAL PUBLIC LICENCE 3.0 (GPLv3) der FREE SOFTWARE FOUNDATION (FSF) gewählt [64], welche als strenges *Copyleft* einge-

Vorteile	Nachteile
<ul style="list-style-type: none"> • Minimierung der Kommunikationsprobleme zwischen Anwender und Entwickler • frühzeitige Erkennung und Behebung von Fehlern, Ungenauigkeiten und Unvollständigkeiten • Steigerung der Produktivität durch Minimierung des Test- und Wartungsaufwands • Vereinfachung der Abschätzung des Aufwands durch detaillierte Anforderungsdefinition • Minimierung des Risikos eines Fehlschlags des gesamten Projektes aufgrund der experimentellen Vorgehensweise 	<ul style="list-style-type: none"> • Gefahr einer endlosen Entwicklungsphase durch immer neu definierte Anforderungen durch den Anwender • Gefahr des Produktiveinsatzes von Prototypen beim Anwender, da dieser schon ein <i>funktionierendes Produkt</i> sieht

Tabelle 6.3: Vor- & Nachteile des Prototyping [nach 6, S. 23]

stuft wird (Abbildung 6.4). Copyleft ist eine Anspielung auf das Wort *Copyright* (deutsch: Urheberrecht) und räumt jedem die Rechte ein, das vorliegende Werk zu vervielfältigen und bewahrt dennoch den Lizenzierungs-Anspruch des Urhebers. Dies bedeutet, dass jeder die Software verwenden, verbreiten und modifizieren kann, solange dies auch weiterhin unter Bedingungen von GPLv3- oder dazu kompatiblen Lizenzen geschieht. Aufgrund dieser Lizenz kann das Simulationstool nach der Open-Source-Definition der OPEN SOURCE INITIATIVE (OSI) als *Freie Software* bezeichnet werden [74].

Im Bereich der behördlichen Nutzung von Software wird oft Open-Source-Software (OSS) empfohlen. So hat zum Beispiel das BUNDESVERWALTUNGSAKT (BVA) ein OSS-Kompetenzzentrum eingerichtet [58], welches den Einsatz von Open-Source-Software in Bundesbehörden fördert [57]. Wie bereits in Abschnitt 6.1 erwähnt, wurde auch am BUNDESAMT FÜR STRAHLENSCHUTZ eine quellenoffene und plattformunabhängige Entwicklung in der Programmiersprache Java bevorzugt. Im folgenden sei eine Gegeüberstellung wesentlicher Vor- und Nachteile des Einsatzes quellenoffener Softwareprogramme an Behörden aufgestellt (Tabelle 6.4).

Vorteile	Nachteile
<ul style="list-style-type: none"> • keine Lizenzkosten • umfassende Nutzungsrechte ohne Einschränkungen der Nutzungsdauer und Nutzerzahl • hohe Qualität und geringe Fehlerrate aufgrund der Möglichkeit, die Quellen unabhängig zu überprüfen (<i>Peer Review</i>) • schnelle Behebung von Fehlern und Sicherheitslücken durch unabhängige Entwickler (<i>Community</i>) • geringe Anfälligkeit für Schadcode aufgrund der Transparenz • stets leicht erweiterbar durch eigene Plugins oder Schnittstellen • Veränderbarkeit des Codes, da jeder weiterentwickeln darf • Plattformunabhängigkeit, da bei Bedarf das Programm portiert werden kann • Software bleibt langfristig verwertbar, Code langfristig lesbar • Herstellerunabhängigkeit, da unterschiedliche Dienstleister Wartung, Support und Entwicklung anbieten können 	<ul style="list-style-type: none"> • geringe Akzeptanz durch Anwender aufgrund veränderter Arbeitsabläufe • erhöhter Schulungs- und Umschulungsaufwand • teilweise schlechter oder gar kein Support • hohe Barrieren für den Einsatz oder Umstieg, da viele fachspezifische Anwendungen meist proprietär sind • keine Haftung oder Gewährleistung seitens der Lizenzgeber

Tabelle 6.4: Wesentliche Vor- & Nachteile von Open-Source-Software für die behördliche Nutzung [nach 60]

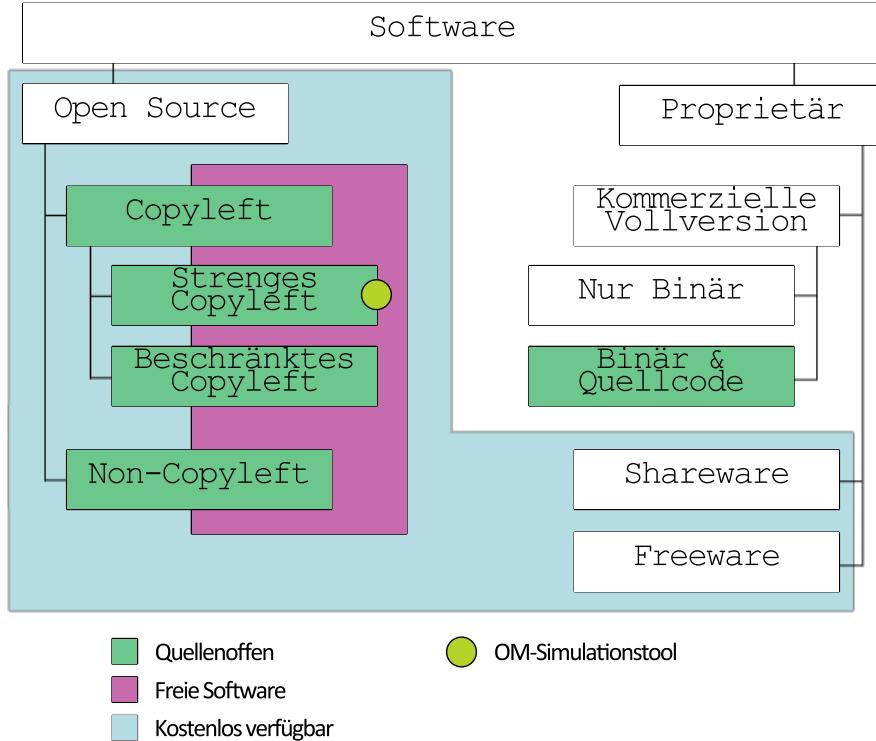


Abbildung 6.4: Unterschiedliche Softwarelizenzierungen [nach 59]

6.3 Versionierung

Die Versionsverwaltung ist in der Softwareentwicklung ein Faktor, der sowohl für die Sicherheit der Daten gegenüber Verlust als auch für die Nachvollziehbarkeit der Änderungen an eben diesen Daten wichtig ist. Darüber hinaus besteht der Vorteil, dass durch die zentrale Speicherung der Daten im Internet (oder gegebenenfalls im Intranet) eine Arbeit an dem Projekt von jeden beliebigen Arbeitsplatz mit Zugang zum Netzwerk möglich ist. (Von der *Offline-Versionierung* wird hier einmal abgesehen.)

In diesem Projekt wird während der frühen Entwicklungsphase das Versionierungssystem Subversion (SVN) [54] auf einem Server der HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT BERLIN verwendet. Für die spätere Vorbereitung der Releases wird nach GITHUB [70] mit der Versionierungssoftware Git [61] migriert (vgl. Abschnitt 7.4.2 *Projektmanagement mit Github*).

6.3.1 Release-Verwaltung

Die Softwareentwicklung mit Versionierungstools unterliegt meist dem Edit-Compile-Test-Zyklus [5, S. 155] (Abbildung 6.5). Dabei wird zunächst der aktuelle Entwicklungsstatus aus dem Versionierungsarchiv (*en. repository*) herunter geladen (mit `update` (SVN) beziehungsweise `pull` (Git)). Daraufhin wird der Zyklus solan-

ge durchlaufen, bis der Entwickler seine Teilschritte fertiggestellt hat und mit den Ergebnissen zufrieden ist.

1. *Edit*: Zunächst wird der Quellcode bearbeitet, um Funktionalität erweitert oder gegebenenfalls korrigiert.
2. *Compile*: Der Quellcode wird dann kompiliert und in Maschinencode beziehungsweise maschinennahen Bytecode übersetzt.
3. *Test*: Die Software wird auf die neuen, überarbeiteten oder korrigierten Funktionen hin getestet. Dieser Prozess ist im Allgemeinen als *debuggen* bekannt.

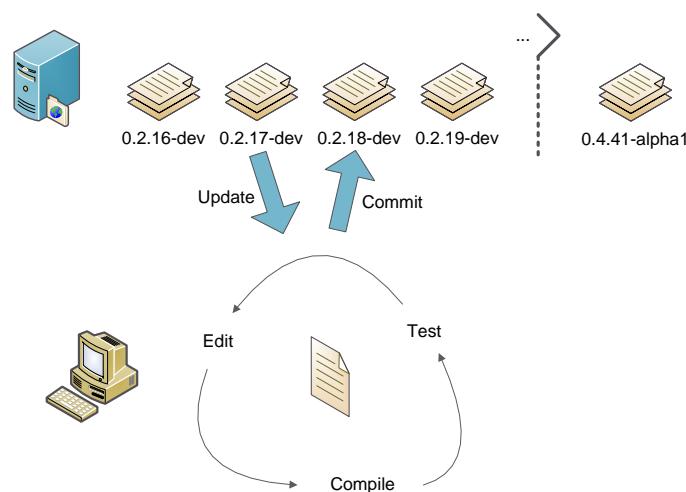


Abbildung 6.5: Der Edit-Compile-Test-Zyklus [nach 5, S. 155]

6.3.2 Etikettierung

Die Etikettierung (auch *en. version tags*) der verschiedenen Softwareversionen wird an die des Linux-Kernels von Linus Torvalds angelehnt ([Vgl. Linux Makefile, 80]). Die Tags werden in vier wesentliche, hierarchische Bestandteile gegliedert (Abbildung 6.6).

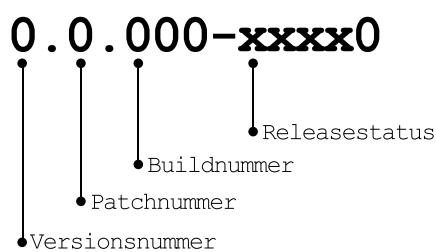


Abbildung 6.6: Etikettierungsschema des Simulationstools

- Die Versionsnummer ist die höchste hierarchische Schicht der Etikettierung. Sie stellt die Hauptversion des Simulationstools dar. Während der Entwicklungsphase ist diese bei 0, die erste stabile Version bekommt die Nummer 1 und so weiter.
- Die Patchnummer ist eine Sub- oder Unterversionsnummer, die zum Beispiel während der Entwicklung die einzelnen Prototypen voneinander abgrenzt.
- Die Buildnummer entspricht intern der SVN- beziehungsweise Git-Commits, das heißt, diese wird immer erhöht, wenn eine funktionierende Version in die Repositories eingecheckt wird (auch bei minimalen Änderungen). Stabile Releases erhalten extern keine Buildnummer (vgl. auch Abbildung 6.5).
- Der Releasestatus dient der schnellen Identifizierung einer Version hinsichtlich ihrer Stabilität und wird in fünf verschiedene Status unterschieden:
 - Der **dev**-Status (*en. development*) kennzeichnet alle ungetesteten, unstabilen Entwicklerversionen.
 - Der **alpha**-Status kennzeichnet alle ersten Testversionen. Dieser Status setzt voraus, dass die größten Teile der Funktionalität vollständig implementiert sind.
 - Der **beta**-Status tritt ein, wenn erste interne Tests erfolgreich verliefen und keine weiteren Probleme identifiziert werden können. Diese Phase wird auch als öffentliche Testphase bezeichnet, in der Anwender das Produkt selbstständig verwenden und überprüfen können.
 - Der **rc**-Status (*en. release candidate*) bezeichnet alle Versionen, die bereit sind zur Veröffentlichung, aber noch nicht als endgültige stabile Versionen gelten.
 - Endgültig stabile Releases erhalten keinen Statustag, zum Beispiel „1.0“.

Die Etikettierung hat den Zweck einerseits einen bestimmten Status der Software einzufrieren, wenn ein gewisser Stand der Entwicklung erreicht wurde, und bietet andererseits die Möglichkeit der Identifizierung und Lokalisierung von Problemen, die Anwender unter Verwendung der Versionsnummer melden können [5, S. 182].

Kapitel 7

Implementierung

Wie bereits im Abschnitt 6.2.2 *Das Prototyping* erwähnt, soll die Implementierung prototypisch erfolgen. Zunächst sind drei Prototypen geplant, die sich mit dem Entwurf und der Implementierung von verschiedenen Aspekten der Anwendung beschäftigen:

1. Eine Konsolenanwendung, die die Kernfunktionalität der Simulationen fokussiert.
2. Eine Datenbankanwendung, die die Anforderungen an die Persistenz umsetzt.
3. Sowie eine Desktopanwendung, die hauptsächlich die Oberfläche als Schwerpunkt setzt.

Dieses Kapitel befasst sich zyklisch während der Entwicklung der Prototypen mit der Entwurfs-, Implementierungs- und Testphase, aber es können auch vereinzelt neue Anforderungen oder Spezifikationen anfallen, die dann nochmals behandelt werden müssen. Dies ist nötig, da das Anforderungsmodell nie fertig ist, da während der Entwicklung so zum Beispiel die Stakeholder neue Ideen haben können oder die Entwickler während der Entwurfsphase neue Erkenntnisse gewinnen [30, S. 61]. So waren zum Beispiel zunächst nur zwei Prototypen geplant, aber durch den Erkenntnisgewinn aus der Konsolenanwendung wurde schnell klar, dass ein weiterer Zwischenschritt nötig ist.

Für die ersten Schritte der Implementierung wird Objekt 8 aus dem Abschlussbericht zum Vorhaben *StSch-4534* [3] als Beispielobjekt verwendet und die aufbereiteten Daten aus dem Abschnitt 5.2.1 *Vorgehensweise* zunächst statisch in den Prototypen eingebunden. Bei zwei Kellerräumen $C = 2$ mit sieben möglichen Messpositionen $P = 7$ und sieben bewohnten Räumen $R = 7$ über einen Zeitraum von 402 Stunden, abzüglich 167 Stunden, in denen die virtuelle Messkampagne über das Ende der reellen Messungen hinaus gehen würde $H = 235$, ergeben sich nach Formel (5.1) aus Abschnitt 5.3 *Größenordnung der Simulationen* circa 16,5 Millionen mögliche Messvariationen, die es zu simulieren gilt.

$$N = \frac{R!}{(R-6)!} \cdot C \cdot P \cdot T \quad (7.1)$$

$$N = \frac{7!}{(7-6)!} \cdot 2 \cdot 7 \cdot 235 = 16.581.600 \quad (7.2)$$

7.1 Prototyp 0.1 - Konsolenanwendung

Zunächst wird ein Prototyp als einfache Konsolenanwendung in Java implementiert. Dies dient zunächst der Überprüfung der gewünschten Funktionalität ohne viel Zeit für die Erstellung von Oberflächen aufzuwenden. Sollte diese erreicht werden, können die Klassen direkt in den folgenden Prototypen wiederverwendet werden.

Ziel ist es hierbei, das Domänenmodell auszuarbeiten (Klassen, Attribute, et cetera), die CSV-Dateien erfolgreich einzulesen und zu parsen, sowie die Ausgabe ausführlicher Logs, die wiedergeben, was die Software zu welchem Zeitpunkt macht, um die zukünftigen Simulationen bis in das letzte Detail rückverfolgen zu können.

Es wurden maximal 80 Personenstunden veranschlagt für die Implementierung dieses relativ vereinfachten Prototypen.

7.1.1 Domänenmodell

Mit der Software- und Systemdomäne plant der Entwickler die grobe Struktur des Softwaretools [30, S. 63]. Dazu wird zunächst ein Paket- oder Klassendiagramm entworfen (Abbildung 7.1). Das Ziel ist es, die Komponenten so zu konzeptionieren, dass sie später in weiteren Prototypen als Teilkomponenten wiederverwendet werden können.

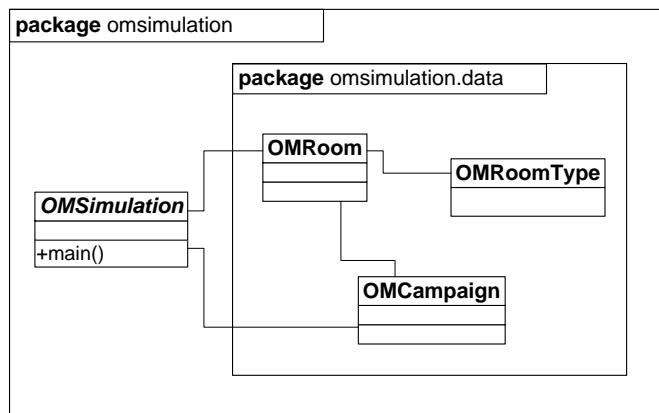


Abbildung 7.1: Paketdiagramm des ersten Prototypen

Der Prototyp soll aus drei Klassen und einer Aufzählung (*Enum*) bestehen. Die abstrakte Hauptklasse **OMSimulation** enthält die `main()`-Methode und stellt somit

den Einstiegspunkt für das Programm dar. Die Klassen `OMRoom` und `OMCampaign` stellen das Grundgerüst dar für die Objekte der einzelnen Räume und Messkampagnen, die später generiert werden sollen. Das Enum `OMRoomType` definiert die drei verfügbaren Raumtypen `R`, `C` und `M`.

Für die statistischen Größen, die in den jeweiligen Objekten berechnet werden, ist zu beachten, dass als Datentyp keine *Integer*-Werte verwendet werden. Die Genauigkeit eines Integers genügt zwar für das Endergebnis, aber beim Rechnen ohne Kommastellen entstehen teils gravierende Fehler, die vermieden werden können, indem als Datentyp sinnvoll für alle Werte *Double* gewählt wird.

7.1.2 Simulationen

Die größte Komplexität – und damit verbunden den größten Aufwand – bei der Implementierung weist die Erstellung aller möglicher Variationen gemäß des Protokolls „6+1“ auf. Allein knapp 1800 Zeilen Code werden für die Methode `generateVariations()` benötigt, was weit über die Hälfte des gesamten Codes abdeckt. Zunächst werden in sechs verschachtelten `for`-Schleifen alle Variationen der bewohnten Räume generiert und in ein Array aus `OMRoom`-Objekten abgelegt (Quellcode 7.1).

Daraufhin werden zu den Raumvariationen an allen möglichen Positionen – wie in Abschnitt 5.3 *Größenordnung der Simulationen* definiert – die Kellerräume in verschachtelten `for`- und `while`-Schleifen hinzugefügt (Quellcode 7.2).

Die beiden vorangegangenen Codebeispiele sind nur Ausschnitte und beziehen sich auch nur auf Gebäude die sechs oder mehr bewohnte Räume besitzen. Bei Gebäuden mit weniger als sechs Räumen kommt eine weitere Komplexitätsdimension der Doppelmessungen hinzu, wie hier am Beispiel eines Gebäudes mit nur vier bewohnten Räumen gezeigt werden soll (Quellcode 7.3).

Sind alle Variationen generiert, kann die eigentliche Simulation beginnen. Dazu werden mit Hilfe der Methode `generateCampaigns()` die `OMCampaign`-Objekte anhand der Variationen erstellt, welche die einzelnen virtuellen Messkampagnen abbilden (Quellcode 7.4).

Die `OMCampaign`-Objekte berechnen automatisch ihre Attribute, welche die wesentlichen statistischen Größen aus Abschnitt 4.3 *Statistische Auswertung* bereit stellen (Quellcode 7.5).

Damit ist die Kernanforderung an das Simulationstool bereits erfüllt, da alle möglichen Messkampagnen und ihre Wertreihen generiert wurden. Auch wenn Daten wie diese hier oft tabellarisch oder grafisch dargestellt werden sollten [42, S. 148], wird hier zunächst der Fokus nur auf Logs gelegt, da die sinnvolle Ausgabe der Ergebnisse erst zu späteren Zeitpunkten der Entwicklung relevant wird.


```

// get all cellars
OMRoom[] cellars = getCellars();
// calculate final number of variations
int finalArraySize = tmpArraySize * cellarCount * 7;
OMRoom[] finalScheme[] = new OMRoom[finalArraySize][7];
int x = 0;
int y = 0;
for (int g = 0; g < cellarCount; g++) {
    while (y < tmpArraySize) {
        finalScheme[x][0] = cellars[g]; // 1st cellar position
        finalScheme[x][1] = tmpScheme[y][0];
        finalScheme[x][2] = tmpScheme[y][1];
        finalScheme[x][3] = tmpScheme[y][2];
        finalScheme[x][4] = tmpScheme[y][3];
        finalScheme[x][5] = tmpScheme[y][4];
        finalScheme[x][6] = tmpScheme[y][5];
        x++;
        y++;
    }
    y = 0;
    while (y < tmpArraySize) {
        finalScheme[x][0] = tmpScheme[y][0];
        finalScheme[x][1] = cellars[g]; // 2nd cellar position
        finalScheme[x][2] = tmpScheme[y][1];
        finalScheme[x][3] = tmpScheme[y][2];
        finalScheme[x][4] = tmpScheme[y][3];
        finalScheme[x][5] = tmpScheme[y][4];
        finalScheme[x][6] = tmpScheme[y][5];
        x++;
        y++;
    }
} // et cetera ...
}

```

Quellcode 7.2: Positionen der Kellermessungen

Tatsächlich stellte sich nach den ersten Druchläufen, die in etwa 20-30 Minuten auf dem Bürorechner dauerten, heraus, dass die Log-Dateien sogar noch wesentlich größer wurden und über 2 GB Speicherplatz benötigten. Für Text- oder Log-Dateien dieser Größenordnung wird spezielle Software benötigt, diese zu öffnen, da konventionelle Texteditoren für gewöhnlich die komplette Datei in den Arbeitsspeicher laden wollen. Das hier verwendete NOTEPAD++ kann nur Dateien bis knapp 2 GB öffnen und ist daher nicht geeignet, da es auch diese in den Arbeitsspeicher lädt. Der Bürorechner hat nur 2 GB Arbeitsspeicher. Alternativen wären nun:

- GVIM
- EMACS
- SLICKEDIT
- LARGE TEXT FILE VIEWER (LTFVIEWER)

Allerdings haben auch diese Programme Probleme, die sich auf die Gesamtperformance des Rechners auswirken kann, bei derartig großen Dateien. Weitere Möglichkeiten wären zum Beispiel die Logs bei einer Größe von über 1GB in Ein-Gigabyte-Parts zu splitten. Auf Linux-Systemen ist die Nutzung des Tools LOGROTATE zu

```

while (y < tmpArraySize) {
    finalScheme[x][0] = cellars[g]; // 1st cellar position
    finalScheme[x][1] = tmpScheme[y][0];
    finalScheme[x][2] = tmpScheme[y][0]; // double measurement
    finalScheme[x][3] = tmpScheme[y][1];
    finalScheme[x][4] = tmpScheme[y][1]; // double measurement
    finalScheme[x][5] = tmpScheme[y][2];
    finalScheme[x][6] = tmpScheme[y][3];
    x++;
    y++;
}
y = 0;
while (y < tmpArraySize) {
    finalScheme[x][0] = tmpScheme[y][0];
    finalScheme[x][1] = tmpScheme[y][0]; // double measurement
    finalScheme[x][2] = cellars[g]; // 2nd cellar position
    finalScheme[x][3] = tmpScheme[y][1];
    finalScheme[x][4] = tmpScheme[y][1]; // double measurement
    finalScheme[x][5] = tmpScheme[y][2];
    finalScheme[x][6] = tmpScheme[y][3];
    x++;
    y++;
} // and so on ...

```

Quellcode 7.3: Positionen der Doppelmessungen für $R = 4$

empfehlen, aber ein solches System steht zur Zeit nicht zur Verfügung, da unter WINDOWS XP entwickelt wird.

Nun muss aber beachtet werden, dass die Logs nicht *bearbeitet* werden sollen. Daher wird keinerlei Text-*Editor* benötigt. Es reichen lediglich schmale Tools wie LESS, um sich die Dateien teilweise, schrittweise oder gar komplett auf die Konsole ausgeben zu lassen und dort die Schritte der Software nachlesen zu können.

Sollte dennoch ein Editor gewünscht werden, ist der LTFVIEWER zu empfehlen, da dieser noch recht performant und einfach zu bedienen ist.

7.1.4 Ergebnisse

Die CSV-Dateien einzulesen, zu parsen und daraus Objekte für die jeweiligen Räume zu erstellen funktioniert wie gewünscht. Auch die Simulation aller möglicher Variationen lies sich relativ schnell implementieren und lief problemlos.

Es traten aber dann aber, wie erwartet, Probleme aus Sicht der Informationstechnologie bei der finalen Erzeugung der kompletten Reihen virtueller Messkampagnen auf.

1. Die erste Überlegung war es, Messkampagnen lokal zu generieren, in eine Logfile zu schreiben (oder alternativ in der Konsole auszugeben) und dann das Objekt mit der nächsten folgenden Messkampagne zu überschreiben. Dies schont zwar (Arbeits-)Speicherkapazität des Rechners, erhöht aber die Rechenzeit für die komplette Simulation rund um den Faktor 70 (Zeit mit Ausgabe auf die

```
// get number of total measurements
int valueCount = getValueCount();
// calculates possible start times
int total = valueCount - 7 * 24 + 1;
// gets the previously generated variations
OMRoom[] campaignRooms[] = getVariationScheme();
int campaignLength = campaignRooms.length;
int x = 0;
// iterates through all available variations
for (int a = 0; a < campaignLength; a++) {
    // iterates through all available start times
    for (int start = 0; start < total; start++) {
        // generates the virtual campaigns
        OMCampaign campaign = new OMCampaign(start, campaignRooms[a]);
        // write generated campaign to log- or csv-file
        // ...
        x++;
    }
}
```

Quellcode 7.4: Generierung der Messkampagnen

```
// constructor of the OMCampaign class
public OMCampaign(int start, OMRom[] rooms) {
    this.Start = start;
    this.Rooms = new OMRom[7];
    this.Rooms[0] = rooms[0];
    this.Rooms[1] = rooms[1];
    this.Rooms[2] = rooms[2];
    this.Rooms[3] = rooms[3];
    this.Rooms[4] = rooms[4];
    this.Rooms[5] = rooms[5];
    this.Rooms[6] = rooms[6];
    // triggers calculation of all statistical parameters
    calculateAttributes();
}
```

Quellcode 7.5: Konstruktor der Klasse OMCampaign

Konsole: 44,9 Minuten, Zeit ohne Ausgabe: 0,663 Minuten. Büro-Rechner: INTEL CORE 2 Duo 2x 1.80 GHz, 2 GB RAM, WINDOWS XP, SP 3). Nachteil hierbei war klar, neben der hohen Rechenzeit, dass die Daten nicht direkt vom Tool weiter verarbeitet werden können, ohne die Logs nochmals zu parsen.

2. Die zweite Überlegung war dann ein Versuch zur Optimierung. Die Objekte für die Messkampagnen wurden nicht nach jeder Ausgabe überschrieben, sondern in ein großes Array gespeichert. Hierbei ergab sich dann ein Arbeitsspeicherproblem. Nach etwa 2,97 Millionen Iterationen (circa 18% der Gesamtanzahl, vgl. Formel (7.2)) hat Eclipse einen JAVA.lang.OutOfMemoryError ausgegeben und das Programm abgebrochen. Bei wiederholten Tests unter den gleichen Bedingungen konnte anhand des Task-Managers festgestellt werden, dass die Arbeitsspeicherlast für diese Methode enorm war. Nach der Simulation von ca. 2,97 Mio. Messkampagnen wurden ca. 287 MB RAM verbraucht (RAM zu Beginn des Tests: 1.018.648 kB, RAM zum Höhe- beziehungsweise Abbruchpunkt 1.312.960 kB). Dies entspricht circa 100 kB pro Messkampagne und wür-

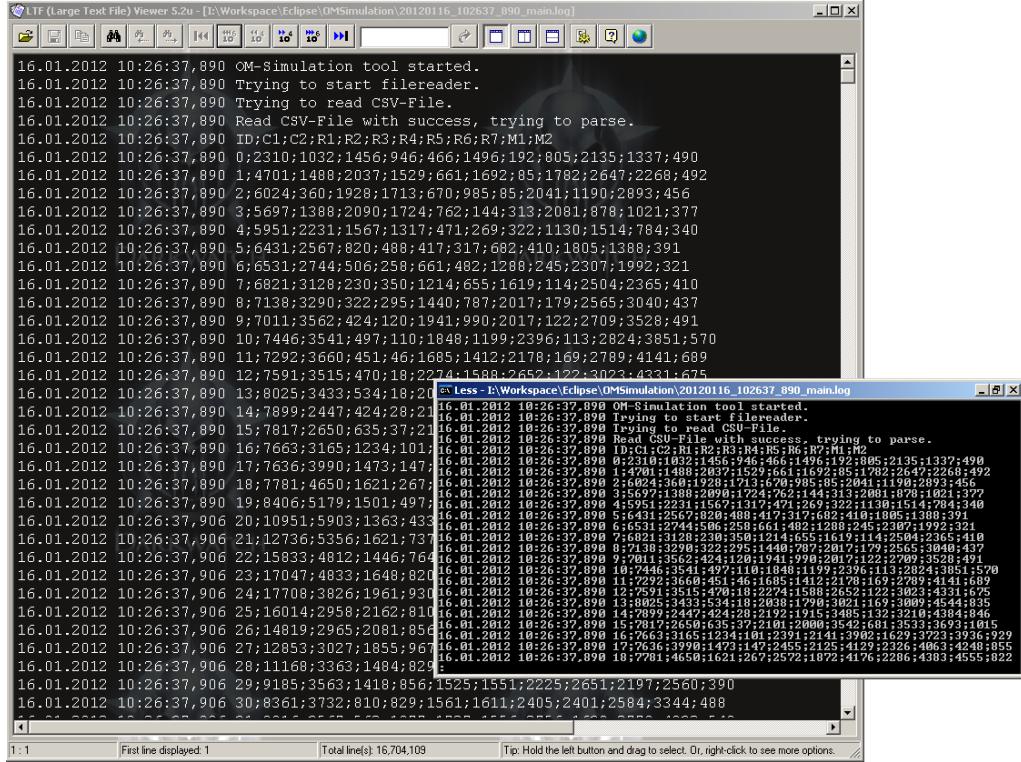


Abbildung 7.2: Logviewer im Vergleich: LTFVIEWER und LESS

de bedeuten, dass, einmal abgesehen vom `JAVA.lang.OutOfMemoryError`, diese Simulation von 16,5 Millionen Messkampagnen rund 1,56 GB Arbeitsspeicher benötigen würde. Vorteil bei dieser Variante wäre allerdings, dass die generierten Messkampagnen sofort in der Software weiter verwendet werden können.

Obwohl die knappen Rechnerkapazitäten in der Vorbetrachtung bedacht wurden, ist die klare Schlussfolgerung nach Implementierung des Prototypen 0.1, dass dieser noch nicht in dem gewünschten Maße dem Funktionsumfang entspricht, wie es in den Anforderungen festgelegt wurde. Daher ist es ratsam an diesem Punkt über einen weiteren Prototypen nachzudenken, welcher speziell die Persistenz der Messkampagnen fokussiert und performant implementiert (vgl. Abschnitt 7.2).

Inhaltliche Ergebnisse zu den generierten Werten können zu diesem Zeitpunkt noch nicht gemacht werden, da die 16,5 Millionen Datensätze zwar schon vorhanden sind, aber dennoch nicht aufbereitet werden können.

7.1.5 Tests

Um gute Softwaretests durchzuführen, muss viel Zeit eingeplant werden [48, S. 18]. Es ist wichtig schon frühzeitig während der Entwicklung Testphasen einzuplanen, da Fehler, die sich zu Beginn einschleichen, um so aufwändiger zu beheben sind, um so später diese entdeckt werden.

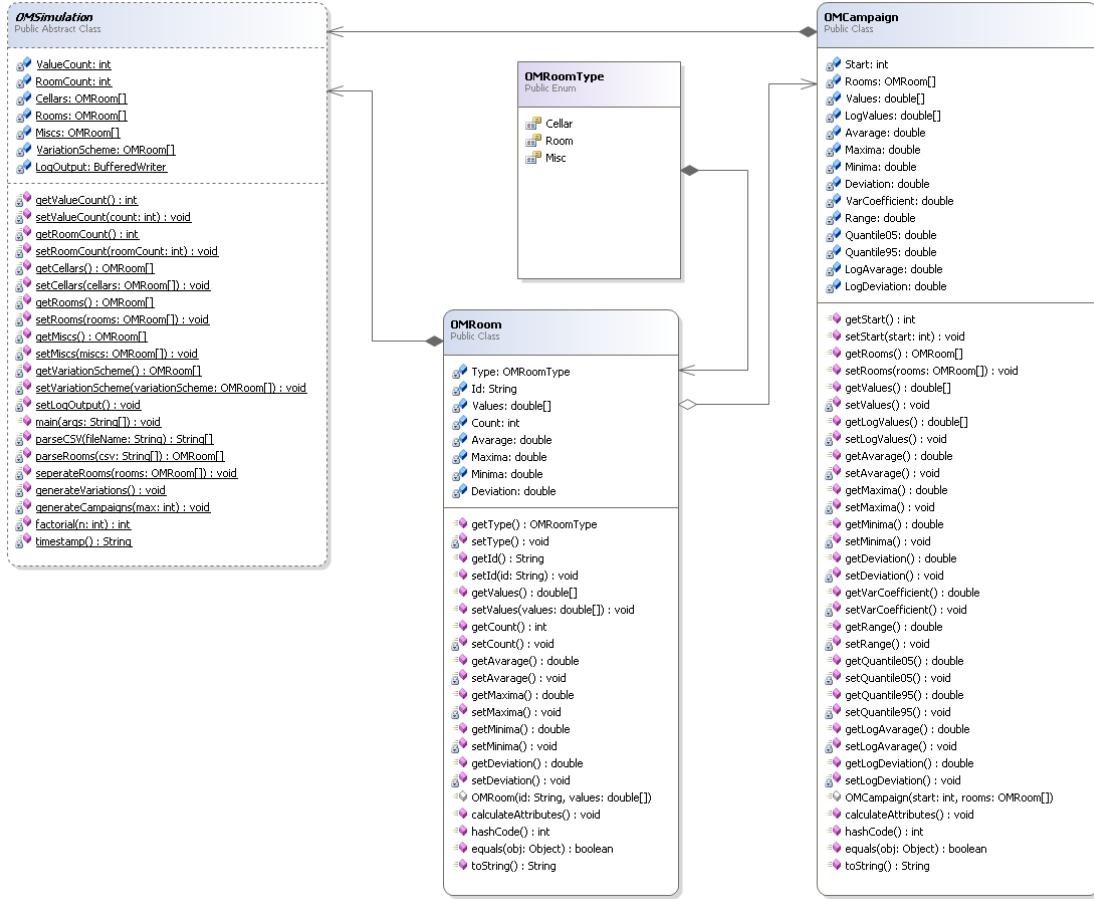


Abbildung 7.3: Klassendiagramm des ersten Prototypen

Um Tests für den vorhandenen Prototypen zu fahren und die Funktionalität auf Ausnahmefehler zu überprüfen werden weitere Objekte mit verschiedenen Datensätzen und Räumen in die Software eingelesen. Dazu ist der Prototyp ähnlich wie ein Testtreiber implementiert. Die Kernfunktionalität wurde in ein Konsolenprogramm eingebettet, welches Testdaten bereit stellt, die beliebig angepasst werden können. Diese Art der Vorgehensweise beim Software-Test wird auch *White-Box-Testing* genannt [48, S. 59ff], da der Tester auch gleichzeitig der Entwickler ist, und somit der Code bekannt ist. Diese Art der Tests soll zu diesem Zeitpunkt zunächst ausreichen, um mindestens einmal jeden Algorithmus im Code zu durchlaufen und um eine einfache Verifikation der Funktionalität zu erreichen.

Dazu wird Objekt 9 aus dem Abschlussbericht zum Vorhaben *StSch-4534* [3] aufbereitet. Dieses Objekt hat nur einen kellerartigen Raum, fünf bewohnte Räume und einen nicht weiter relevanten Raum (1C, 5R, 1M). Die wesentlichen Unterschiede zu Objekt 8, mit dem die Software bis hier entwickelt wurde, sind:

- nur fünf bewohnte Räume anstatt von sieben, was bei diesem Objekt einen anderen Algorithmus im Code des Prototypen 0.1 anspricht
- nur ein Kellerraum anstatt von zweien

- wesentlich geringere Radon-Werte

Tatsächlich war der erste Testlauf ein Erfolg, da ein Ausnahmefehler in eben dem bis dato unzureichend getesteten Algorithmus auftrat. Dieser Algorithmus generiert alle möglichen Variationen mit nur 5 Räumen. Dieser Fehler konnte aber debugt, lokalisiert und ausgebessert werden. Darüber hinaus fielen in den Logs weitere kleinere Flüchtigkeitsfehler auf, unter anderem vergessene Zeitstempel, zu groß angesetzte Arrays oder `null`-Werte, die alle Problemlos behoben werden konnten.

Um das Verhalten des Prototypen abschließend nochmals zu verifizieren, wurden aus Objekt 9 eine neue CSV-Datei erstellt, indem der Header modifiziert wurde, um ein Gebäude mit drei Kellern und vier bewohnten Räumen zu simulieren. Darüber hinaus wurden alle Datensätze ab $ID = 336$ entfernt, um die gewünschte optimale Simulationsstrecke von 14 Tagen zu erreichen. Paralell wurden die berechneten Werte einzelner Messkampagnen stichprobenartig in EXCEL nachgerechnet und somit validiert.

Die Simulationsläufe funktionieren einwandfrei und die Werte stimmen mit den manuell berechneten überein. Somit können die Tests an dieser Stelle vorerst abgeschlossen werden und mit der Entwicklung des zweiten Prototypen begonnen werden.

7.2 Prototyp 0.2 - Datenbankanwendung

Da der zweite Prototyp vorwiegend die Persistenz der Daten, die bei der Simulation anfallen, fokussiert, sind hier zunächst ein paar neue Vorbetrachtungen notwendig. Auch wenn die Überschrift Datenbanken suggeriert, sind die Möglichkeiten der Persistenz wesentlich vielseitiger. Zu diesem Zeitpunkt der Entwicklung sind theoretisch alle Möglichkeiten in Erwägung zu ziehen. Zum einen können die Daten weiterhin in Text-Dateien abgelegt werden, hierfür würden sich vor allem CSV oder XML anbieten, aber es gibt auch Methoden, XLS-Dateien (EXCEL) als Datenablage sinnvoll zu verwenden. Zum anderen ist in Betracht zu ziehen, dass es nicht nur die klassischen, relationalen Datenbanken gibt, sondern auch neuere Ansätze der Objektdatenbanken und objektrelationalen Datenbanken.

7.2.1 Überlegungen zur Persistenz

Die XSQL-JDBC-TREIBER [78] bieten die Möglichkeit Daten aus Java-programmen im XLS-, CSV- oder XML-Format abzulegen. Was EXCEL angeht, sind die Limits für die Radon-Simulationen mit mehreren Millionen Datensätzen zu eng gesetzt. XLS-Dateien bis EXCEL 2003 erlauben lediglich 65.536 Zeilen und 256 Spalten, XLSX ab 2007 immerhin schon 1.048.576 Zeilen und 16.384 Spalten, was aber auch hier bei weitem nicht genügt [50]. Somit ist die Persistenz mit EXCEL von vornherein auszuschließen.

Was XML-Dateien angeht, ist hier mit großen Performanz- und Speicherplatz-Problemen zu rechnen. Die Struktur von XML ist so angelegt, dass für jeden Datensatz gemäß der Anforderungen sieben Zeilen benötigt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<SIMULATION>
<CAMPAIGN id="0">
<VAR>C1R1R2R3R4R5R6</VAR>
<AM>1234</AM>
<GM>1170</GM>
<Q50>1189</Q50>
<MAX>1934</MAX>
</CAMPAIGN>
<CAMPAIGN id="1">
    ...
</CAMPAIGN>
...
</SIMULATION>
```

Quellcode 7.6: Messkampagne abgebildet in XML

Eine XML-Datei mit 16,5 Millionen Datensätzen hätte dann schon über 100 Millionen Zeilen. Der unverhältnismäßige Speicherplatzbedarf und der Mehraufwand, diese Datensätze wieder in die Software einzulesen, lassen auch XML als Persistenzformat ausschließen.

CSV ist weiterhin als sinnvoller Datentyp für die aktuelle Problemstellung zu betrachten, da jeder Datensatz mit minimalstem Speicherplatz-Aufwand abgelegt wird und sehr einfach sowohl von Mensch als auch vom Java-Programm gelesen werden kann.

```
0;C1R1R2R3R4R5R6;1234;1170;1189;1934
1; ...
...
```

Quellcode 7.7: Messkampagne abgebildet in CSV

In Java bieten die JDBC-TREIBER [75] eine gute Möglichkeit mit vielen verschiedenen relationalen Datenbanken zu arbeiten. Dabei übernehmen die Treiber die Kommunikation mit der Datenbank. Somit können SQL-Statements einfach in den Java-Code eingebettet werden und Daten aus der Datenbank leicht abgerufen werden. Die für das Projekt passensten relationalen Datenbanken wären HYBERSQL (HSQLDB) [69], FIREBIRD [63], SQLITE [77] oder H2 [68]. Alle vier sind frei erhältlich, Open-Source, sowie plattformunabhängig und laufen im sogenannten *embedded mode* (eingetragenen Modus, vgl. Abbildung 7.4), was bedeutet, dass kein extra Datenbank-Server und damit einhergehende Administration nötig ist.

Die H2-Datenbank sticht bei diesem Vergleich heraus, da sie nicht nur in Java implementiert ist und als einfache JAR-Bibliothek eingebunden werden kann, sondern auch aufgrund ihrer schmalen Umsetzung die beste Performanz vorweisen kann im Vergleich zu anderen Datenbanken [67]. Das ablegen eines relationalen Datensatzes in Java würde wie folgt aussehen (Quellcode 7.8, vereinfacht).

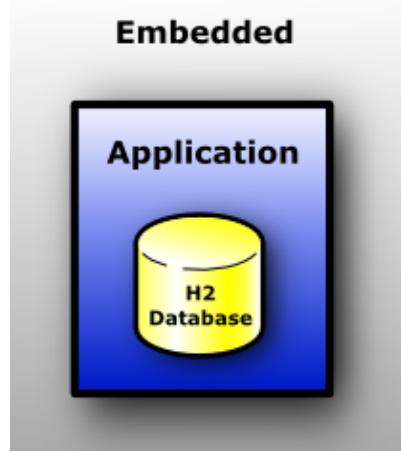


Abbildung 7.4: H2-Datenbank im eingebetteten Modus [66]

```

Class.forName("org.h2.Driver");
Connection db = DriverManager.getConnection("jdbc:h2:rel.db", "user", "pass");
Statement stmt = db.createStatement();
stmt.executeUpdate("INSERT INTO CAMPAIGN VALUES ('C1R1R2R3R4R5R6', 1234, 1170, ←
    1189, 1934)");
stmt.executeUpdate( /* ... another statement */ );
// ... add more statements
stmt.close();
db.close();

```

Quellcode 7.8: Abfrage mit der relationalen Datenbank H2

Nach diesem Ansatz kann aber nur unter erheblichen Aufwand wieder ein Objekt aus dem Datensatz gewonnen werden.

Da das Tool bisher komplett objektorientiert programmiert wurde und alle Räume sowie Messkampagnen als Objekte vorliegen, ist es an diesem Punkt auch durchaus sinnvoll, über eine Objektdatenbank nachzudenken, welche die Daten nicht in Tabellen ablegt, sondern direkt die Objekte speichert [45, S. 24]. Mögliche Objektdatenbanken für Java-Programme, die plattformunabhängig und unter freien Lizzenzen angeboten werden sind unter anderem DB4O [81], EYEDB [62] und PERST.LITE [72], wobei nur DB4O und PERST im eingebetteten Modus angeboten werden. Wie bereits angedeutet, ist dieser moderne Ansatz der Persistenz in Java sehr einfach zu implementieren. Quellcode 7.9 veranschaulicht, wie zum Beispiel eine Datenbank-Abfrage für das Anlegen einer neuen Messkampagne in DB4O aussieht (vereinfacht).

```

ObjectContainer db = Db4o.openFile("obj.db");
db.store(new OMCampaign(23, getRooms()));
db.store( /* ... another object */ );
// ... add more objects
db.commit();
db.close();

```

Quellcode 7.9: Abfrage mit der Objektdatenbank DB4O

Wie am Beispiel zu erkennen ist, wird hier kein SQL benötigt. Objektdatenbanken gehören daher auch zur Familie der NoSQL-Datenbanken (*Not only SQL*). Probleme könnten aber auch bei diesem Verfahren bezüglich großer Datenmengen auftreten. So müssen beim beenden des Programms die Objekte serialisiert (*pickle*) und beim erneuten Start des Programms wieder deserialisiert (*unpickle*) werden, was zu großen Performanz-Problemen führen kann und viel Hauptspeicher benötigt [27].

Abschließend ist zu dieser längeren Vorbetrachtung zu sagen, dass theoretisch keine Schlussfolgerungen getroffen werden können, welcher Ansatz nun der Beste ist, ohne eigene Tests im aktuellen Anwendungsfall durchzuführen. Da solche Benchmarks an dieser Stelle das Projekt *ad absurdum* führen würden, wird zunächst einfach der modernere, objektorientierte Ansatz implementiert. Sollte die Software an Grenzen stoßen, kann immernoch auf klassische Methoden zurückgegriffen werden.

Objektrelationale Datenbanken (ORDBMS) verhalten sich wie relationale Datenbanken, bieten aber vereinfachte Bedienung im Umgang mit Objekten, sogenannte OR-Mapper, an [1, S. 91f]. Dies wäre auch eine weitere Alternative, falls die Persistenz mit der Objektdatenbank fehlschlagen sollte.

7.2.2 Statistik

Ein bisher vernachlässigtes Problem ist, dass einige statistische Größen bei sehr großen Simulationsläufen nicht berechnet werden können. So können zum Beispiel keine Perzentile oder der Median bei Wertereihen von mehreren Millionen Datensätzen ohne Weiteres ermittelt werden, da dazu alle Werte sortiert und aufgelistet werden müssen. Denn für die Berechnung von vier statistischen Größen von sowohl unbewohnten als auch bewohnten Räumen und das sowohl für die arithmetischen als auch die geometrischen Mittelwerte müssen während der Simulation 16 Double-Arrays gefüllt werden (acht für die reinen und acht für die logarithmierten Werten).

$$n = 4 \cdot 2 \cdot 2 = 16 \quad (7.4)$$

Die Berechnung der einzelnen Größen ist weder mathematisch noch algorithmisch problematisch und war schnell implementiert. Doch die ersten Simulationen sorgten sofort für „Java.lang.OutOfMemoryError: Java heap space“-Fehler (Abbildung 7.5). Entsprechend der gesetzten Argumente für die Virtuelle Maschine (sogenannte VMargs), stehen mit dem Parameter `-Xmx512m` 512 MB Arbeitsspeicher der VM zur Verfügung [5, S. 94].

Ein Double benötigt 8 Byte. Ein Double-Array mit 16,5 Millionen Werten benötigt somit circa 127 MB. Damit überschreitet allein die Initialisierung der 16 Arrays mit insgesamt 1,977 GB benötigtem Arbeitsspeicher alle Grenzen sowohl der VM als auch des benutzten Bürokechners (Formel (7.5)). Eine Erhöhung des zugesicherten Heap-Space der VM ist daher an dieser Stelle nicht sinnvoll.

$$V = 8B \cdot 16.581.600 \cdot 16 = 2.122.444.800B = 1,977GB \quad (7.5)$$

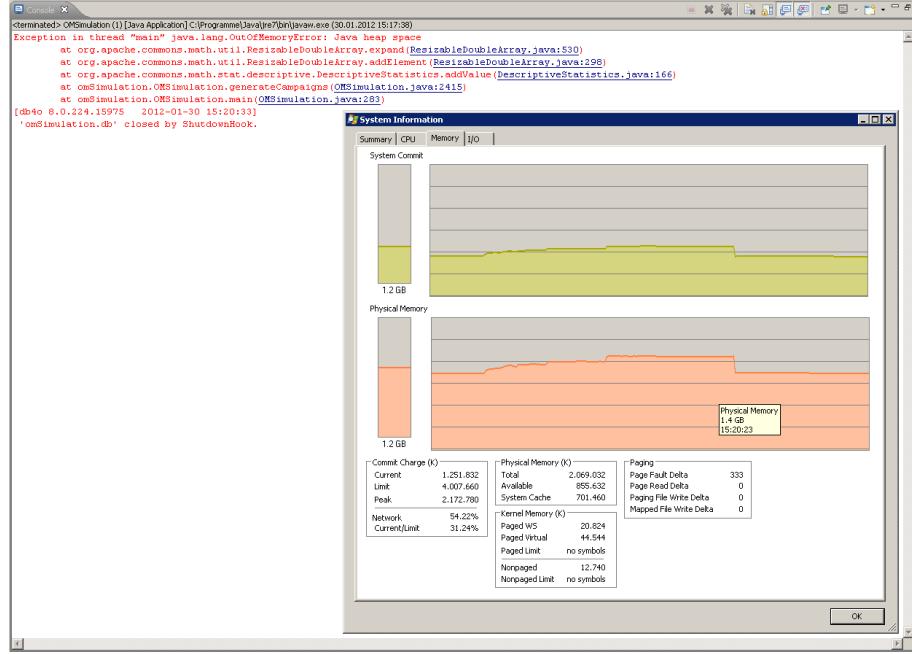


Abbildung 7.5: Die JVM überschreitet den zugesicherten Arbeitsspeicher und bricht ab

Anders herum betrachtet sind pro Variation 128 B benötigt (Formel (7.6)), was ein theoretisches Maximum von gut 4 Millionen Variationen bei 512 MB Arbeitsspeicher erlaubt (Formel (7.7)).

$$V = 8B \cdot 16 = 128B \quad (7.6)$$

$$N = \frac{512MB}{128B} = \frac{536.870.912B}{128B} = 4.194.304 \quad (7.7)$$

Versuche ergaben aber, dass alle Simulationen mit mehr als einer Million Variationen nicht ohne Fehler liefen. Daher gilt es nun nach alternativen Lösungen Ausschau zu halten und die Herangehensweise bei der Berechnung der Statistik zu überdenken. Denn neben der *deskriptiven Statistik*, die jeden einzelnen Wert speichert und zum Ende der Simulation daraus die Statistik bildet, gibt es für größere Problemstellungen auch die *zusammenfassende Statistik* [39, S. 147f], die während der Simulation statistische Größen ermitteln kann, ohne dabei alle Werte einzeln zu speichern. Dies hat den großen Vorteil, dass nicht alle Daten in den Arbeitsspeicher geschrieben werden müssen, sondern lediglich die nach jedem Durchlauf der Simulationsschleife berechneten statistischen Werte. Der Nachteil hierbei ist allerdings, dass bei dieser Variante – wie bereits eingangs betrachtet – Perzentile und der Median nicht berechnet werden können, da dafür stehts alle Werte benötigt werden.

Zur Berechnung der deskriptiven und zusammenfassenden Statistik werden in der Software nun die Bibliotheken der APACHE COMMONS MATH [53] verwendet, welche fast alle benötigten Methoden bereits implementiert haben.

Darüber hinaus wird nun in die Software ein Switch eingebaut. Simulationen mit

bis zu einer Million Durchläufen werden weiterhin deskriptiv durchgeführt, alles was darüber hinaus läuft, wird nur zusammenfassend mit Verzicht auf die Perzentile simuliert.

7.2.3 Ergebnisse

Der Prototyp 0.2 wurde nach drei Wochen im Februar 2012 fertig gestellt. Die Objektdatenbank DB4O konnte erfolgreich eingebunden werden und arbeitete zunächst wie gewünscht. Aufgrund der enormen Datenmengen der Simulationen wurde allerdings davon abgesehen, die eigentlichen Simulationen und simulierten Messkampagnen in der Datenbank abzulegen. Stattdessen wurde mit einer neuen Klasse `OMBuilding` die Möglichkeit geschaffen einzelne Gebäude als Objekte anzulegen.

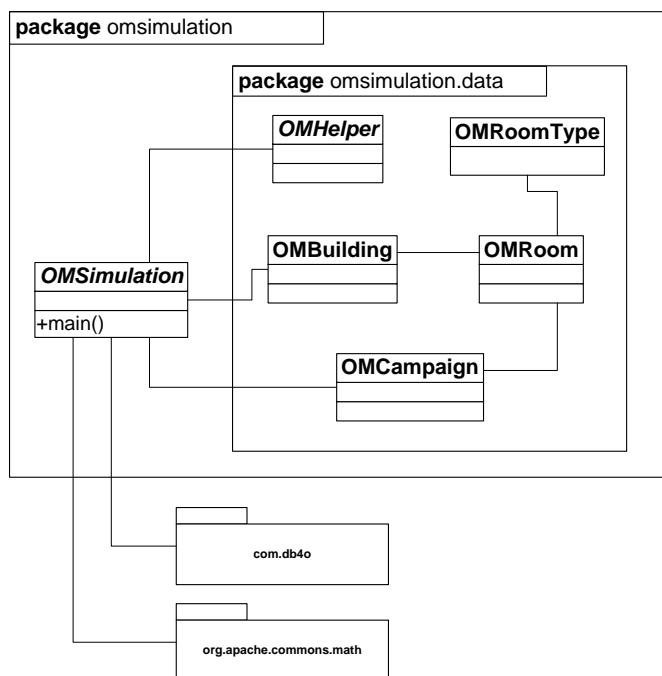


Abbildung 7.6: Paketdiagramm des zweiten Prototypen

Jedes Gebäude kann eine beliebige Anzahl von Räumen und Kellern haben und jedem Raum beziehungsweise Keller sind die jeweiligen Messreihen zugewiesen. So mit wurde die Handhabung der Objekte im Programm vereinfacht, da man nun die Messkampagnen über ein einziges Objekt (Gebäude) simulieren kann. Dieses Objekt wird mit all seinen Abhängigkeiten und Datenreihen in der Objektdatenbank abgelegt und kann nach Programmstart immer wieder abgerufen werden, ohne die Werte erneut einlesen zu müssen.

Aufgrund der schon zuvor diskutierten Limits in den Statistiken, wurden verschiedene neue Funktionalitäten für die Simulationsläufe hinzugefügt, die im folgenden kurz erläutert werden sollen.

- Die Funktionalität aus dem ersten Prototypen war, dass Simulationen mit drei, vier, fünf oder sechs Räumen systematisch durchlaufen werden können. Dazu werden entsprechend der Raumzahl des verwendeten Gebäudes alle Variationen einmal simuliert. Auch war es in Version 0.1 möglich schon ein Limit der maximalen Simulationsläufe zu setzen und somit probabilistisch eine begrenzte Anzahl an Messkampagnen zu generieren.
- Neu ist nun in Version 0.2, dass zufällige Simulationen einen Verteilungsschlüssel übergeben bekommen. Das heißt, dass der Anwender der Software später die Möglichkeit hat, zu definieren, im welchen Verhältnis wieviele Räume in Messkampagnen simuliert werden sollen. Die Angabe des Verhältnisses 73:20:5:2 zum Beispiel bedeutet, dass 73% der virtuellen Kampagnen mit sechs bewohnten Räumen simuliert werden, dass 20% mit fünf Räumen simuliert werden, 5% mit vier, sowie 2% mit drei Räumen.

Dies dient später der Möglichkeit zur Evaluation, ob die Anzahl der bewohnten Räume maßgeblich Einfluss auf die Simulationsergebnisse hat.

7.2.4 Tests

Um den zweiten Prototypen zu testen wird auch hier zunächst die Vorgehensweise des *White Box Testing* angewandt, da es zu diesem Zeitpunkt noch nicht möglich ist, Externe mit dem Testen der Software zu beauftragen. Dieser soll aber diesmal etwas ausführlicher gestaltet werden, als die Tests zum ersten Prototypen. Neben einen Funktionstest sind auch Last- und Ausnahmetests geplant.

- Ausnahmetests sind absichtliche Fehlerangriffe auf die Software, um bestimmte Fehlereinwirkungen zu überprüfen und korrekt abzufangen [2, S. 110]. Dadurch wird sichergestellt, dass absichtliche oder versehentliche Fehleingaben im Produktivbetrieb nicht zu Systemfehlern führen.
- Funktionstests sind einfache Verfahrenstests, die auf verschiedene Art und Weise die angestrebte Funktionsweise mit korrekten Eingabedaten überprüfen [48, S. 122]. Dies dient der Erkenntnisgewinnung, ob *die Software tut was sie soll* – ganz im Sinne des einfachen White-Box-Tests.
- Die Last- und Stresstests dienen der Ermittlung von möglichen Grenzen der Software und zur Beurteilung in wie weit das verwendete System auf große Datenmengen und langwierige Rechenzeiten reagiert [2, S. 172ff].

Vorweg ist zu erwähnen, dass die Algorithmen im aktuellen Prototypen folgende Minimalanforderungen an die Eingabedaten im CSV-Format stellen:

- Das Gebäude muss mindestens drei verwendbare bewohnte Räume aufweisen ($R \geq 3$).

- Das Gebäude muss mindestens einen unbewohnten Kellerraum besitzen ($C \geq 1$).
- Die Datenreihen müssen mindestens 168 Datensätze enthalten, um eine Woche Messzeitraum zu simulieren ($T \geq 168$).

Die Testdaten werden mit EXCEL generiert, um den Prozess zu beschleunigen. Es werden 20 verschiedene CSV-Dateien erstellt, die verschiedene Testszenarien ansprechen sollen (siehe Tabelle 7.1).

R	C	T	Szenario	Test
0	0	341	zu wenig Räume und Keller	Ausnahmetest
0	6	992	zu wenig Räume	Ausnahmetest
1	1	895	zu wenig Räume	Ausnahmetest
1	5	87	zu wenig Räume und Datensätze	Ausnahmetest
2	2	108	zu wenig Räume und Datensätze	Ausnahmetest
2	4	662	zu wenig Räume	Ausnahmetest
3	3	344	-	Funktionstest
4	2	475	-	Funktionstest
4	4	358	-	Funktionstest
5	1	653	-	Funktionstest
5	5	325	viele Keller	Funktions-/Lasttest
6	0	98	zu wenig Keller	Ausnahmetest
6	6	175	viele Keller	Funktions-/Lasttest
7	1	694	-	Funktionstest
7	7	949	viele Keller	Stresstest
8	2	528	viele Räume	Funktions-/Lasttest
8	6	64	zu wenig Datensätze	Ausnahmetest
9	3	575	viele Räume	Stresstest
9	5	763	viele Räume und Keller	Stresstest
10	4	616	viele Räume	Stresstest

Tabelle 7.1: 20 CSV-Testdateien

Ausnahmetests

Im folgenden seien zunächst für die Testszenarien der Ausnahmetests die Ergebnisse aufgezählt. Ziel ist es hierbei, das Programm auf alle erdenkbaren Fehleingaben vorzubereiten und diese ggf. durch Fehlermeldungen abzufangen. Würden solche Fehler im Produktivbetrieb auftreten und das Programm Ausnahmefehler werfen, könnten wichtige Daten verloren gehen.

- **0 Räume, 0 Keller, 341 Datensätze:** Bei diesem Test wurde zwar vom Programm erkannt, dass nicht genügend Räume und Keller vorhanden sind und eine entsprechende Fehlermeldung ausgegeben, aber dennoch wurde ein

Gebäude als Objekt in der Datenbank angelegt, was beim erneuten Programmstart einen Fehler lieferte. Um dies zu beheben, werden nun Gebäude, die nicht genug Räume und Keller aufweisen, abgefangen, bevor sie in der Objektdatenbank abgelegt werden.

- **0 Räume, 6 Keller, 992 Datensätze:** erfolgreich abgefangen.
- **1 Raum, 1 Keller, 895 Datensätze:** erfolgreich abgefangen.
- **1 Raum, 5 Keller, 87 Datensätze:** erfolgreich abgefangen.
- **2 Räume, 2 Keller, 108 Datensätze:** erfolgreich abgefangen.
- **2 Räume, 4 Keller, 622 Datensätze:** erfolgreich abgefangen.
- **6 Räume, 0 Keller, 98 Datensätze:** erfolgreich abgefangen. Eine Fehlermeldung in der Log-Datei wurde umformuliert, um das Verständnis des Problems zu garantieren, dass nicht nur mindestens drei Räume, sondern auch mindestens ein Keller vorhanden sein muss.
- **8 Räume, 6 Keller, 64 Datensätze:** erfolgreich abgefangen.

Zusammenfassend ist zu sagen, dass die Ausnahmetests zwar erfolgreich waren, aber vorwiegend Fehler in den CSV-Dateien abgefangen haben, die im Prototypen 0.1 schon hätten getestet werden können.

Funktionstests

Im folgenden werden erweiterte Funktionstests durchgeführt, die die normale Funktionalität des Tools ansprechen werden. Dazu werden zu jedem Datensatz drei Simulationsläufe durchgeführt: ein systematischer Testlauf, ein kurzer zufälliger Testlauf, der die deskriptive Statistik anspricht, sowie ein längerer zufälliger Testlauf, der die zusammenfassende Statistik anspricht. Zusätzlich werden Zeiten der Durchläufe sowie Speicherplatzbedarf der Datenbanken und Log-Dateien ermittelt, um gegebenenfalls Optimierungspotenziale aufzudecken. Die Ergebnisse sind in Tabelle 7.2 zusammengefasst.

R	C	T	PRS	DB	ZDS	ZZS	SYS	Resultat
3	3	344	1,06s/231kB	22,4kB	9,50s	162s	2,14s/1,74MB (D)	22 W
4	2	475	2,09s/321kB	72,5kB	9,44s	163s	38,5s/60,8MB (D)	25 W, P
4	4	538	1,48s/554kB	130kB	9,66s	164s	88,1s/146MB (Z)	23 W
5	1	651	1,45s/568kB	244kB	9,67s	163s	141s/239MB (Z)	10 W
5	5	325	3,91s/1,71MB	1,05MB	9,84s	163s	232s/388MB (Z)	7 W
6	6	175	7,30s/8,90MB	6,28MB	11,2s	164s	22,1s/29,8MB (D)	5 W, 6 E
7	1	694	7,25s/5,89MB	3,81MB	11,7s	174s	24,9m/2,49GB (Z)	6 W
8	2	528	23,2s/31,4MB	21,7MB	15,4s	168s	135m/13,6GB (Z)	2 W, P

Tabelle 7.2: Ergebnisse der Funktionstests

Erläuterungen zu Tabelle 7.2:

- PRS: Zeit zum Einlesen der Daten und Größe der Logs
- DB: Größe der Objektdatenbank, in der ein Objekt des jeweiligen Gebäudes abgelegt ist
- ZDS: Zeit der zufälligen, deskriptiven Simulation bei 100.000 Durchläufen im Verhältnis 73:20:5:2, die Größe der Logs betrug immer 13,7MB
- ZZS: Zeit der zufälligen, zusammenfassenden Simulation bei 2.000.000 Durchläufen im Verhältnis 73:20:5:2, die Größe der Logs betrug immer 275MB
- SYS: Zeit der systematischen Simulation und Größe der Logs, die Gesamtzahl der Simulationen hängt von der Komplexität der Daten ab, daher kann immer nur entweder deskriptive (D) oder zusammenfassende (Z) Simulation durchgeführt werden.
- Resultat: W = Warnung (unkritisch), E = Fehler/Exception (kritisch), P = Problem identifiziert. Probleme und Fehler werden nachfolgend erläutert.
 - Bei der zufälligen Simulation für das Gebäude mit 4 Räumen und 2 Kellern entsprechend dem o.g. Verhältnis trat der Fall ein, dass für die Berechnung der absoluten Simulationszahl die Summe nicht glatt war. Für dieses Fall sieht das Tool zwar eine Lösung vor, die aber nur unzureichend durchdacht war. So wurde für ein Gebäude mit nur vier bewohnten Räumen plötzlich versucht eine Messkampagne mit sechs Räumen zu generieren. Dies wurde korrigiert, indem zuvor überprüft wird, welche Variationen überhaupt zur Verfügung stehen.

```

10:10:57,015 Simulation will start with 100000 random variations:
10:10:57,015 28571 Simulations using 3 different rooms. (Ratio: ca. 28% (2))
10:10:57,015 71428 Simulations using 4 different rooms. (Ratio: ca. 71% (5))
10:10:57,015 0 Simulations using 5 different rooms. (Ratio: ca. 0% (0))
10:10:57,015 1 Simulations using 6 different rooms. (Ratio: ca. 0% (0))
10:10:57,015 Warning: Simulating 1 campaigns for 6 rooms, but only 0 variations ←
are existing for 6 rooms.
10:10:57,015 Warning: Consider to reduce the ratio or number of total ←
simulations.

```

Quellcode 7.10: Logausgabe

- Bei versehentlicher Falscheingabe des Dateinamens reagierte das Programm korrekt und hat kein Objekt angelegt.
- Abschließen trat noch ein kleinerer Fehler bei der Berechnung des Status auf. Das Problem war hierbei, dass der Integer, der zur Berechnung verwendet wird, zunächst mit 100 multipliziert wird, was den Maximalwert eines 32-Bit-Integers (circa 2,15 Milliarden) schon ab 21,5 Millionen Simulationen überschreitet. (Tatsächlich wurde im Log in Zeile 21.534.739 der erste negative Status -21% gefunden, womit das Problem bewiesen ist.) Die Lösung ist hier trivial, indem einfach die Multiplikation mit 100

```

14:53:30,000 Trying to read CSV-File '2s\testdata_6r6c175.csv'.
14:53:30,000 Error: 2s\testdata_6r6c175.csv (Das System kann die angegebene ←
Datei nicht finden)
14:53:30,000 Error: Failed to read CSV-File2s\testdata_6r6c175.csv.
14:53:30,015 Error: Separation test failed. Not enough rooms or cellars.
14:53:30,015 Error: Not enough rooms. No building stored to database.

```

Quellcode 7.11: Logausgabe

an das Ende der Gleichung gestellt wird. Für x oder max einen Long-Datentyp zu verwenden scheint nicht nötig, da Simulationen mit über 2 Mrd. Durchläufen selbst nicht mal in den Belastungstests auftraten.

```

perc = x * 100 / max; //wrong
perc = x / max * 100; //new

```

Quellcode 7.12: Status

Aus diesen Ergebnissen lassen sich folgende Benchmarkwerte für das Simulations-Tool auf dem Büreorechner ermitteln (Tabelle 7.3). Als Benchmarks werden in der Testbranche Überprüfungen der Leistungsfähigkeit von Programmen oder Systemen bezeichnet, die durch verschiedene Messungen und durch Bildung von Kennzahlen definiert werden können [48, S. 131].

R	C	T	Variationen	Zeit	Geschw.	Speicher	Sp./Simulation
4	4	538	1.065.600	88s	12.109/s	149.504kB	143,67B
5	1	651	1.738.800	141s	12.331/s	244.736kB	144,13B
5	5	325	2.826.000	232s	12.181/s	397.312kB	143,97B
7	1	694	18.557.280	1.493s	12.429/s	2.610.954kB	144,07B
8	2	528	101.606.400	8.088s	12.562/s	14.339.777kB	144,52B
Durchschnitt				12.323/s			144,07B

Tabelle 7.3: Benchmarks der systematischen Simulationen

Last- und Stresstests

Anhand der Kennzahlen aus den Benchmarks des vorangegangenen Abschnitts lassen sich zunächst theoretische Laufzeiten und Speicherbedarf der Last- und Stresstests berechnen (Tabelle 7.4).

Aufgrund des großen Zeit- und Speicheraufwands für die Belastungstests wurde auf einen privaten Rechner mit einer anderen Plattform zurückgegriffen. Daher können diese Tests auch gleichzeitig als Configuration Tests unter anderen Hard- und Softwarebedingungen betrachtet werden [48, S. 135]. Folgende Eckdaten weist der Computer auf:

R	C	T	N	Zeit	Speicher
7	7	949	192.875.760	4,35h	25,9GB
9	3	574	515.652.480	11,6h	69,2GB
9	5	763	1.259.496.000	28,4h	169GB
10	4	616	1.896.652.800	42,8h	254GB

Tabelle 7.4: Vorraussichtliche Systemlast

- Plattform: LINUX MINT DEBIAN GNU/Linux 3.0
- Hardware: AMD ATHLON X2 5200 @ 2.60 GHz, 4 GB RAM
- Java VM: JDK 7, VMargs -Xss2048k -Xmn256m -Xms1024m -Xmx2048m

Anzumerken ist hierbei, dass der größere Arbeitsspeicher von 4GB auch größeren Heap für die Java Virtuelle Maschine bieten kann. Daher wurde dieser für die Lasttests auf 2GB verdoppelt. Folgende Simulationen wurden zunächst durchgeführt (Tabelle 7.5).

#	R	C	T	PRS	DB	Simulation	Probleme
1	10	4	616	-	-	-	DB4O-Exception (write)
2	9	5	763	-	-	-	DB4O-Exception (write)
3	9	3	574	-	-	-	DB4O-Exception (write)
4	8	6	640	88,7s/89,6MB	65,2MB	-	DB4O-Exception (read)
5	8	5	640	73,0s/74,9MB	54,4MB	-	DB4O-Exception (read)
6	8	4	640	60,7s/60,2MB	43,5MB	8,34h/35,6GB	-
7	7	7	949	66,9s/36,8MB	26,5MB	5,47h/25,8GB	-

Tabelle 7.5: Ergebnisse der ersten Lasttests

Die Simulationen #1, #2 und #3 konnten zwar von der Software erfolgreich ge-parst werden, allerdings gelang es nicht, das Objekt `OMBuilding` in der Datenbank abzulegen, da die DB4O-Bibliotheken an dieser Stelle eine Exception warfen. Durch Reduzierung der Anzahl der Räume konnten dann Simulationen #4 und #5 er-folgreich eingelesen und in der Datenbank gespeichert werden. Allerdings versagte hier auch die DB4O-Bibliothek beim erneuten Öffnen der Datenbank. Lediglich die Simulationen #6 und #7 verliefen problemlos. Daraus lässt sich auf ein maxima-les Raumkontingent von 8 bewohnten und 4 unbewohnten Räumen ermitteln. Um dies auf Konsistenz zu überprüfen wurden drei weitere Simulationen durchgeführt (Tabelle 7.6).

#	R	C	T	PRS	DB	Simulation	Probleme
8	8	4	4444	69,4s/63,6MB	43,8MB	-	A.C.MATH-Exception
9	8	4	1024	66,1s/60,6MB	43,5MB	15,3h/64,6GB	-
10	8	4	672	63,5s/60,3MB	43,5MB	9,07h/38,0GB	-

Tabelle 7.6: Ergebnisse weiterer Lasttests

Bei der 8. Simulation stellte sich dann heraus, dass sehr lange Datensätze zwar problemlos von der Datenbank verarbeitet werden können, hierbei aber dann die APACHE.COMMONS.MATH-Bibliotheken Exceptions warfen. Die Simulationen #9 und #10 verliefen wiederum problemlos. Aufgrund der zuvor aufgetretenen Ausnahmefehler werden folgende Begrenzungen für die Software für sinnvoll erachtet und in den Code eingebaut:

- minimal bewohnte Räume: 3
- maximal bewohnte Räume: 8
- minimal unbewohnte Kellerräume: 1
- maximal unbewohnte Kellerräume: 4
- minimal Datensätze: 168 (entspricht einer Woche)
- maximale Datensätze: 1008 (entspricht sechs Wochen)

Sollten Objekte diese großzügig gesetzten Grenzen überschreiten, ist vom Anwender gegebenenfalls darüber nachzudenken, ob es sinnvoll ist die Datensätze aufzuteilen.

Ein weiterer nicht-trivialer Fehler ist bei diesen Tests aufgefallen. So wurden gleich in mehreren Berechnungen die mathematische (1,2,3,...) sowie die informationstechnische Zählweise (0,1,2,...) verwechselt, was dazu führte, dass:

- der letzte Datensatz der CSV-Datei nicht eingelesen wurde.
- die Anzahl der Datensätze nicht stimmte und der letzte eingelesene Datensatz nicht berücksichtigt wurde (es fehlten somit schon die letzten beiden Datensätze).
- die Zufallszahlen in den probabilistischen Simulationen falsch berechnet wurden und somit immer eine Variation weniger generiert wurde als möglich.

Um diese festgelegten Grenzen anschließend nochmals zu verifizieren, wird folgende Simulation auf dem etwas leistungsschwächeren Bürorechner durchgeführt (VMargs `-Xss1024k -Xmn128m -Xms512m -Xmx1024m`, Tabelle 7.7).

#	R	C	T	PRS	DB	Simulation	Probleme
11	3	1	168	1,02s/51,0kB	9,90kB	0,875s/7,39kB	-
10	8	4	1008	37,4s/67,2MB	43,5MB	11,5h/66,6GB	-

Tabelle 7.7: Ergebnisse weiterer Lasttests

Nach Behebung aller aufgetretenen Fehler sind die Tests für den Prototypen 0.2 erfolgreich abgeschlossen und es kann zur Entwicklung des dritten Prototypen übergegangen werden.

7.3 Prototyp 0.3 - Desktopanwendung

Für die Desktopanwendung sind zwei wesentliche Grundanforderungen von Bedeutung.

1. Zunächst steht der Entwurf und die Umsetzung einer Bedienoberfläche für die bereits vorhandenen Funktionalitäten an. Dies betrifft:
 - a. den Import von CSV-Dateien und Ablage in der Objektdatenbank.
 - b. die Durchführung von systematischen und zufälligen Simulationen mit denen im Prototypen 0.2 zuvor definierten Parametern.
2. Anschließend muss die Oberfläche mit entsprechenden Elementen zur Auswertung ausgestattet werden. Dies umfasst:
 - a. die Untersuchung der Datensätze und grafische Ausgabe der Messreihen einzelner Räume.
 - b. die Auswertung der Ergebnisse der zufälligen Simulationen.

Es wurde sich darauf verständigt, dass die systematischen Simulationen nur der Orientierung dienen und aufgrund ihrer großen anfallenden Ergebnismengen (vgl. Tabelle 7.7) eine einzelne Auswertung der Datensätze sowohl statistisch als auch informationstechnisch betrachtet nicht zielführend sei. Für die Auswertung der systematischen Simulationen genügt das Log-File und die Ausgabe der Gesamt-Ergebnisse als CSV. Die zufälligen Simulationen werden auf 100.000 Durchläufe begrenzt und sollen in der Datenbank abgelegt werden, um sie im Anschluss einer ausführlichen grafischen Auswertung zuführen zu können.

7.3.1 Entwurf der Oberfläche

Die Oberflächen sollten einfach gehalten werden und intuitiv bedienbar sein. Auf Menüs wird verzichtet, da der Umfang der Funktionalität eher übersichtlich ist. Die Navigationsfolge erfolgt durch eine Tab-Leiste am oberen Rand der Software. Dabei soll die Reihenfolge der Tabs auch dem logischen Ablauf einer Simulation entsprechen: Import der CSV-Dateien, Möglichkeit zur Überprüfung der Datensätze, Durchführung der Simulation, Auswertung der Simulation, Möglichkeit zu manuellen Tests, sowie Export der Ergebnisse.

Die **Import-Ansicht** ermöglicht das Einlesen von CSV-Dateien. Es wird ein Name für das Projekt oder Objekt definiert, ein Datum des Beginns der Messungen sowie der Dateiname und der Pfad der CSV-Datei. Um 0-Werte und leere Werte auszutilgen beim Einlesen, wird die untere Nachweisgrenze angegeben, damit das Programm solche Werte entsprechend verarbeiten kann.

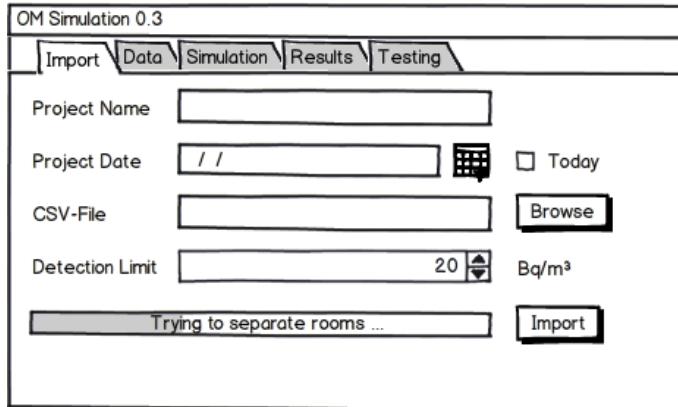


Abbildung 7.7: Import-Ansicht

Die **Datensatz-Ansicht** dient der manuellen Überprüfung der Datensätze. So können alle Konzentrationsverläufe aller Räume einzeln aufgerufen und auf Unregelmäßigkeiten untersucht werden: Angezeigt sollen neben dem Verlauf der Radonkonzentration über die Zeit auch das Maximum, das arithmetische Mittel sowie die Standardabweichung.

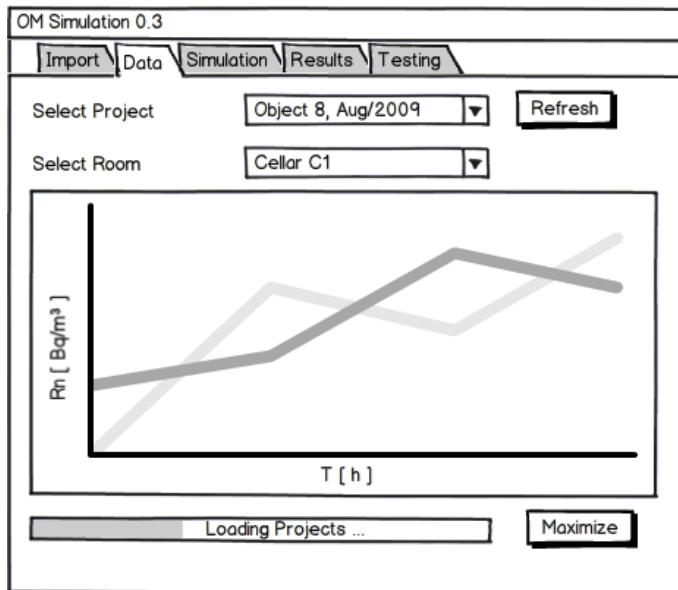


Abbildung 7.8: Datensatz-Ansicht

Die **Simulations-Ansicht** ist der eigentliche Kernpunkt der Software. Hier können Simulationen anhand der eingelesenen Datensätze durchgeführt werden. Das entsprechende Objekt muss zunächst ausgewählt werden. Werden keine Objekte angezeigt, können diese mit dem Aktualisierungs-Button aus der Datenbank geladen werden. Nun kann zwischen systematischen und zufälligen Simulationen gewählt werden. Bei zufälligen Simulationen müssen weitere Parameter angegeben werden: Die maximale Anzahl der zu simulierenden Messkampagnen sowie das Verhältnis der Simulation von 3+1, 4+1, 5+1 und 6+1 Räumen. Durch das Hinzufügen von Hintergrundrauschen (*random noise*) können in gewissen Maße Messfehler mitsimuliert

werden.

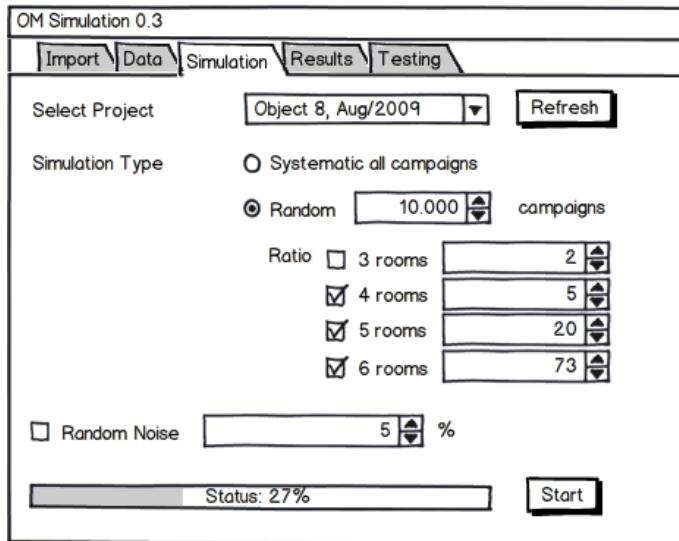


Abbildung 7.9: Simulations-Ansicht

Die **Auswertungs-Ansicht** stellt grafisch die Verteilungsfunktion der statistischen Ergebnisse einer Simulation dar. Dazu muss eine Simulation aus der Datenbank geladen werden. Durch das Anklicken eines Wertes in der Grafik soll die entsprechend hinterlegte Messkampagne in der Testansicht geöffnet werden können, damit eine möglichst fundierte Auswertung der Ergebnisse möglich ist.

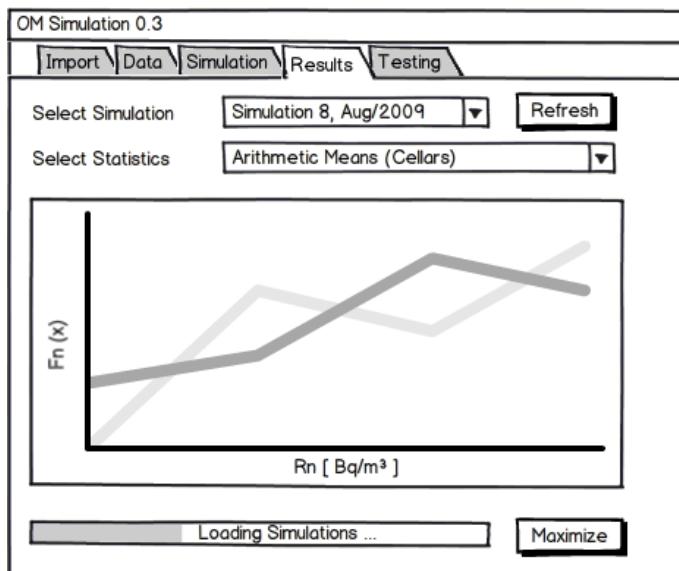


Abbildung 7.10: Auswertungs-Ansicht

Die **Testansicht** ermöglicht es, manuelle Messkampagnen zu erstellen. Dazu muss ein zuvor eingelesenes Objekt geladen werden. Nun können sechs beliebige Räume, ein Keller, sowie ein Startzeitpunkt ausgewählt werden und eine Messkampagne simuliert werden. Angezeigt werden detaillierte statistische Ergebnisse in sowohl tabellarischer als auch grafischer Form.

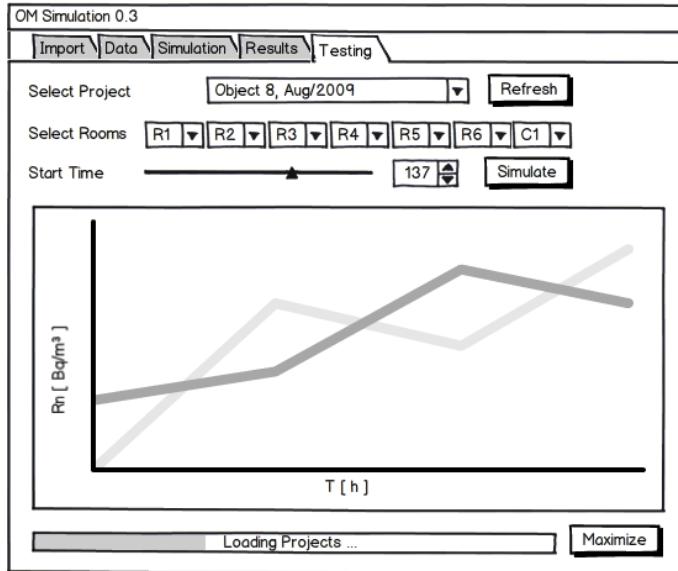


Abbildung 7.11: Testansicht

Die Entwürfe wurden mit BALSAMIQ MOCKUPS erstellt [55]. Die Implementierung der Oberfläche für das Simulationstool erfolgt mit dem von Java bereitgestellten *Standard-Widget-Toolkit* (SWT), dass eine einheitliche, systemunabhängige Schnittstelle für Grafik- und andere GUI-Elemente bereit stellt [5, S. 294f]. Dazu nutzt SWT die jeweiligen Elemente der Oberfläche automatisch vom verwendeten Betriebssystem, ohne dass der Entwickler die verschiedenen Elementtypen kennen und extra implementieren muss.

7.3.2 Nebenläufigkeit und Parallelisierung

Parallelisierung wurde bei diesem Simulationstool bisher vernachlässigt, da die Implementierung eher simpel gehalten wurde. Aber nun sollen einige Schritte zur Optimierung vorgenommen werden. Da Simulationen oft sehr hohe Anforderungen an die Hardware stellen, sollten auch bei diesem Simulationstool die Prozesse in verschiedenen Threads parallelisiert werden. Die Problemstellung ist, dass einzelne Prozesse des Programms standardmäßig zum Beispiel nicht auf verschiedene Kerne des Prozessors aufgeteilt werden, wodurch große Kapazitäten der Hardware ungenutzt bleiben [23, S. 373f, 377f].

Ganz akut wird dieses Problem im aktuellen Anwendungsfall, wenn eine Simulation gestartet wird. Dabei friert die Oberfläche ein, weil sowohl Oberfläche als auch Simulation in einem einzigen Thread ausgeführt werden. Damit das Programm aber auch während langwieriger Prozesse responsiv bleibt, wird nun der Simulationsprozess in einen neuen Thread ausgelagert.

Dazu wird eine neue, innere Klasse angelegt, die das Interface `SwingWorker` implementiert (Quellcode 7.13). Diese innere Klasse enthält zwei vorgegebene Methoden: Die Methode `doInBackground()` arbeitet hier ähnlich wie die `main()`-Methode

des Prototypen 0.2 und startet mit allen nötigen Parametern die Simulation. Nach Abschluss der Simulation wird die Methode `done()` aufgerufen, die den Prozess abschließt.

```
// inner class simulation implements the swing-worker-interface
class Simulation extends SwingWorker<Void, Void> {
    public Void doInBackground() {
        // run the simulation
    }
    public void done() {
        // do stuff after simulation finished
    }
}
```

Quellcode 7.13: Aufbau eines Swing-Worker-Threads für Simulationen

Um den Thread mit der Simulation zu starten, wird ein Objekt der inneren Klasse `Simulation` instanziert und durch die Methode des Interfaces `execute()` gestartet (Quellcode 7.14).

```
// create a new thread
Simulation simulationTask = new Simulation();
// execute the task in background
simulationTask.execute();
```

Quellcode 7.14: Ausführung eines Threads im Hintergrund

Das Resultat ist, dass nun Simulationen im Hintergrund laufen können und der Anwender mit der Software weiterhin arbeiten kann, zum Beispiel um vorangegangene Simulationen auszuwerten oder ähnliches.

7.3.3 Grafische Ausgabe der Ergebnisse

Um die Daten, die durch den Import und die Simulationen generiert werden, sinnvoll darstellen zu können, sollen Grafen ausgegeben werden. Es stellte sich schnell heraus, dass eine eigenständig implementierte Lösung für die grafische Ausgabe der Ergebnisse schnell den Rahmen dieses Projektes sprengen würde. Daher wurde nach Bibliotheken ausschau gehalten, die diese Aufgabe übernehmen können.

Dazu wurden die JFREECHART-Bibliotheken entdeckt [71]. Sie stellen ein komplexes und umfassendes Werk dar, um Funktionsgrafen und Diagramme jeglicher Art in Java-Programmen zu erzeugen und darzustellen. JFREECHART selbst wurde komplett in Java implementiert und ist unter freien Lizenzen erhältlich.

Als Beispiel sei hier eine Verteilungsfunktion der arithmetischen Mittel der bewohnten Räume einer Simulation mit 10.000 generierten Messkampagnen aufgeführt (Abbildung 7.12). Um diese Funktion zu erstellen muss lediglich ein `JFreeChart`-Objekt instanziert werden (Quellcode 7.15). Diesem Objekt können dann komplett

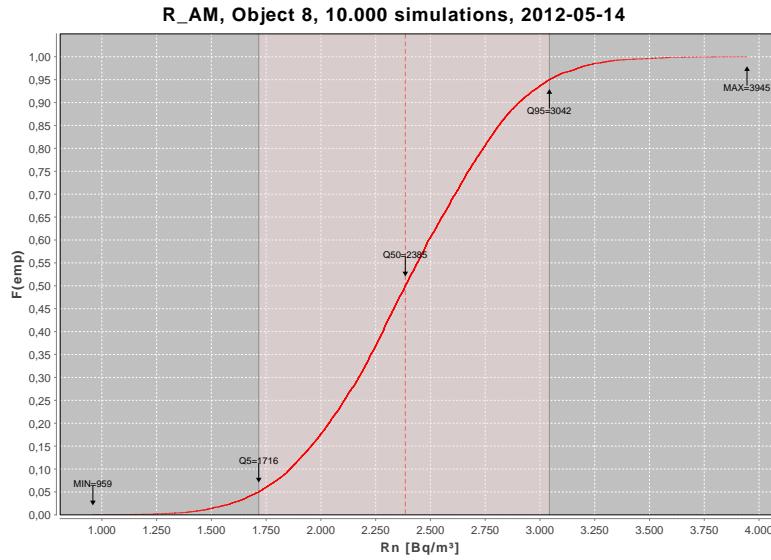


Abbildung 7.12: Verteilungsfunktion der arithmetischen Mittel der bewohnten Räume einer Simulation mit 10.000 generierten Messkampagnen

Datensätze hinzugefügt werden und nach belieben eine Vielzahl von Optionen angepasst werden.

```
// method to create a distribution chart
public static JFreeChart createDistributionChart(String title, ←
    DescriptiveStatistics statistics, boolean preview) {
    // get values and create dataset
    double[] distValues = statistics.getSortedValues();
    XYSeriesCollection dataSet = new XYSeriesCollection();
    XYSeries distSeries = new XYSeries("Distribution");
    for (int i = 0; i < distValues.length; i++) {
        distSeries.add(distValues[i], (0.5 + (double) i) / (double) distValues.←
            length);
    }
    dataSet.addSeries(distSeries);
    // create chart with chartfactory
    JFreeChart chart = ChartFactory.createXYLineChart(title, "Rn [Bq/m³]", "F(emp)←
        ", dataSet, PlotOrientation.VERTICAL, false, true, false);
    // modify plot settings
    XYPlot plot = (XYPlot) chart.getPlot();
    // ...
    return chart;
}
```

Quellcode 7.15: Methode um eine Verteilungsfunktion zu erstellen

7.3.4 Tests

Da mit Abschluss des dritten Prototypen im Großen und Ganzen alle Anforderungen erfüllt sind, wird nun zu einer Art *Black-Box-Testing*-Phase über gegangen. Im Gegensatz zu den Tests vom ersten und zweiten Prototypen wird nun das Tool öffentlich bereit gestellt, so dass Externe das Programm testen können, die den Quellcode

nicht kennen [48, S. 81ff]. Der wesentliche Unterschied ist dabei, dass die Funktionalität nicht darauf untersucht wird, ob die Software tut was sie soll, sondern ob sie *nicht tut was sie nicht soll* [48, S. 122].

Für diese öffentliche Testphase wird ein neuer Prototyp in `alpha`-Version zur Verfügung gestellt und die Resultate werden im nächsten Abschnitt behandelt.

7.4 Prototyp 0.4 - öffentliche Testphase

Der vierte Prototyp kann in dieser Phase der Entwicklung schon als Vorabversion betrachtet werden, da prinzipiell alle Anforderungen bereits umgesetzt wurden und soweit funktionieren. Im folgenden werden aber noch Optimierungen vorgenommen. Im Vordergrund steht hierbei vor allem, dass die Software nun so vorbereitet werden soll, dass sie der Endanwender ohne Weiteres nutzen kann. Der Prototyp 0.3 liegt nur als Quellcode und JVM-Code vor, welcher nicht direkt ausführbar ist. Daher wird nun ein ANT-Build-Skript entwickelt, die Software in ein öffentliches Repository bei GITHUB hochgeladen und Start-Routinen für verschiedene Plattformen geschrieben, so dass der Anwender das Programm durch einen Klick nur noch zu starten braucht.

Diese Phase wird auch als Systemtest bezeichnet. Dabei steht die Gesamtvalidierung und ein Akzeptanztest beim Kunden im Vordergrund, welcher nun überprüfen kann, ob Anforderungen erfüllt sind oder nicht [48, S. 162].

7.4.1 ANT-Build-Skripte

ANT [52] ist ein Build-Tool, dass ähnlich wie früher MAKE die Verwaltung von Softwareentwicklungsprojekten und aller zugehöriger Dateien übernimmt [49, S. 19]. Das Prinzip beruht auf einer Build-Datei (ähnlich der *Makefile* unter MAKE), in welcher der Entwickler alle Regeln definiert, die zum Erstellen, Kompillieren und Ausführen der Software notwendig sind. Die Build-Skripte von ANT sind im XML-Format aufgebaut und erfüllen damit generelle Anforderungen an die hierarchische Struktur des Build-Prozesses [36, S. 13]. Der Trend führte in der Java-Entwicklung weg von MAKE hin zu ANT, da ANT selbst in Java implementiert und damit plattformunabhängig ist, während hingegen MAKE Kompatibilitätsprobleme zwischen verschiedenen Betriebssystemen aufweist [5, S. 323f].

Für das Simulationstool wird nun ANT verwendet, da bis einschließlich zum dritten Prototypen das Projekt in ECLIPSE entwickelt wurde. Um das Tool nun von der IDE zu lösen und auf die Veröffentlichung vorzubereiten, wird das Projekt auf verschiedenste strukturelle Anforderungen hin überarbeitet. So wird zunächst die Verzeichnisstruktur an die der ANT-Richtlinien angepasst [vgl. 5, S. 354]:

- Das `/bin`-Verzeichnis (von *en. binary*) enthält alle kompilierten (`.class`) und gepackten (`.jar`) Binärdateien, die zur Ausführung des Programmes benötigt werden.

- Das */doc*-Verzeichnis (von *en. documentation*) enthält die wesentliche Quellcode-Dokumentation.
- Das */lib*-Verzeichnis (von *en. libraries*) beinhaltet alle Bibliotheken dritter Anbieter, von denen das Simulationstool abhängt.
- Im */src*-Verzeichnis (von *en. source*) finden sich alle Quellcode-Dateien (*.java*), die vom Anwender eingesehen und gegebenenfalls bearbeitet und kompiliert werden können.
- Im Wurzel-Verzeichnis der Software werden Start-Skripte und eine Bedienungsanleitung abgelegt.

```

<?xml version="1.0"?>
<project name="OMSimulation" default="jar" basedir="..">
  <property name="name" value="omsimulation" />
  <property name="version" value="0.4.54-rc2" />
  <property name="bin.dir" value="bin" />
  <property name="lib.dir" value="lib" />
  <property name="src.dir" value="src" />
  <property name="packages" value="de.bfs.radon.omsimulation.* , de.bfs.radon.omsimulation.data.* , de.bfs.radon.omsimulation.gui.* , de.bfs.radon.omsimulation.gui.data.*" />
  <property name="main.class" value="de.bfs.radon.omsimulation.OMMainFrame" />
  <path id="classpath">
    <fileset dir="${lib.dir}" includes="**/*.jar"/>
  </path>
  <target name="jar" depends="compile">
    <jar jarfile="${bin.dir}/${name}-${version}.jar" compress="no" basedir="${bin.dir}" includes="de/**" />
  </target>
  <target name="compile">
    <javac srcdir="${src.dir}" destdir="${bin.dir}" classpathref="classpath" debug="off" deprecation="on" optimize="on" source="1.7" includeantruntime="false" />
  </target>
</project> $ %todo remove $

```

Quellcode 7.16: ANT-Skript für den zweiten Release Candidate (vereinfacht)

Ein Beispiel eines frühen ANT-Skriptes für den vierten Prototypen ist im Quellcode 7.16 zu sehen. Neben grundlegenden Eigenschaften (*en. property*) und dem Klassenpfad (*en. path*) sind die verschiedenen Build-Ziele (*en. target*) erkennbar, die nacheinander ausgeführt werden. Dieses Skript – von unten nach oben gelesen – kompiliert zunächst alle *.java*-Dateien aus dem */src*-Verzeichnis in das */bin*-Verzeichnis und verpackt diese danach in ein Java-Archiv (*.jar*).

Die Verwendung von ANT lohnt erst ab einem gewissen Projektumfang [36, S. 3]. Das Simulationstool ist zwar nicht sehr umfangreich hinsichtlich der Anzahl von Klassen und Paketen, stellt aber hohe Anforderungen an die virtuelle Maschine von Java hinsichtlich der Speicherauslastung (vgl. Abschnitt 7.2.4). Das Build-Skript ermöglicht es, Parameter an die JVM zu übergeben (sogenannte *VMargs*), die den zugesicherten Arbeitsspeicher erhöhen.

7.4.2 Projektmanagement mit GITHUB

Für die öffentliche Testphase und die spätere Veröffentlichung des Simulationstools wird der Quellcode von der intern zur Entwicklung verwendeten Versionierungssoftware SVN in ein öffentliches Git-Repository auf GITHUB migriert. Git ist besonders leistungsstark und flexibel und stellt eine grundlegende Weiterentwicklung und Verbesserung bestehender Versionsverwaltungssysteme dar [34, S. 1, 5f], welches ursprünglich von Linus TORVALDS 2005 zur besseren Verwaltung des umfangreichen Linux-Kernels entwickelt wurde [28, S. 19].

Das bekannteste kostenfreie Git-Hosting ist seit 2008 GITHUB, welches nicht nur öffentliche Git-Repositories anbietet, sondern auch ein umfassendes Projektmanagement ermöglicht [28, S. 307ff]. So werden unter anderem Wikis zur Dokumentation, Issue-Tracker zur Nachverfolgung von Bugs sowie Hilfe-Seiten zur Unterstützung der Endanwender angeboten. Durch das Anlegen von Tags werden automatisch für jede Version der Software gepackte Downloads im `.zip`- und `.tar.gz`-Format erstellt. Ein weiterer wesentlicher Vorteil von GITHUB ist, dass jeder potenzielle Entwickler die Software forken (von *en. fork* - abzweigen) und nach Belieben modifizieren oder weiterentwickeln kann.

Das öffentliche Repository für das OM-Simulationstool ist unter github.com/donSchoe/omsimulation zu finden.

7.4.3 Aufgedeckte Probleme und Optimierungen

Während der Testphase wurden vor allem Optimierungen vorgenommen. So wurde unter anderem die Performanz des Simulationstools verbessert, indem die eingelesenen Objekte besser verarbeitet wurden, und vor allem die Oberfläche optimiert, sodass diese auch unter Linux- und Mac-OS-Systemen verständlich und bedienbar ist.

Durch das Verarbeiten des Codes durch ANT wurden auch Fehler im Quellcode entdeckt, die ECLIPSE scheinbar ignoriert hat. So wurden zum Beispiel unzulässige Unicode-Symbole direkt im Quellcode abgelegt, ohne die offiziellen Unicode-Character-Repräsentationen [] zu verwenden.

Als einziger gravierender Fehler fielen einige Probleme in den Algorithmen zu der Berechnung der arithmetischen und geometrischen Standardabweichung auf, welche darauf hin nochmals komplett überarbeitet und optimiert wurden.

Als nach einigen Wochen alle Fehler behoben waren und auch durch externe Tester keine Probleme mehr entdeckt werden konnten, wurde als erste stabile Version das fertige OM-Simulationstool 1.0 veröffentlicht. Somit ist an diesem Punkt die Entwicklung vorrest abgeschlossen.

Kapitel 8

Fazit

Mit dem Abschluss der Entwicklung des vorgestellten Simulationstools ist auch der technische Dokumentationsteil an dieser Stelle abgeschlossen. Die eigentliche wissenschaftliche Überprüfung der Robustheit orientierender Messungen durch das BUNDESAMT FÜR STRAHLENSCHUTZ steht allerdings noch aus.

8.1 Resultat

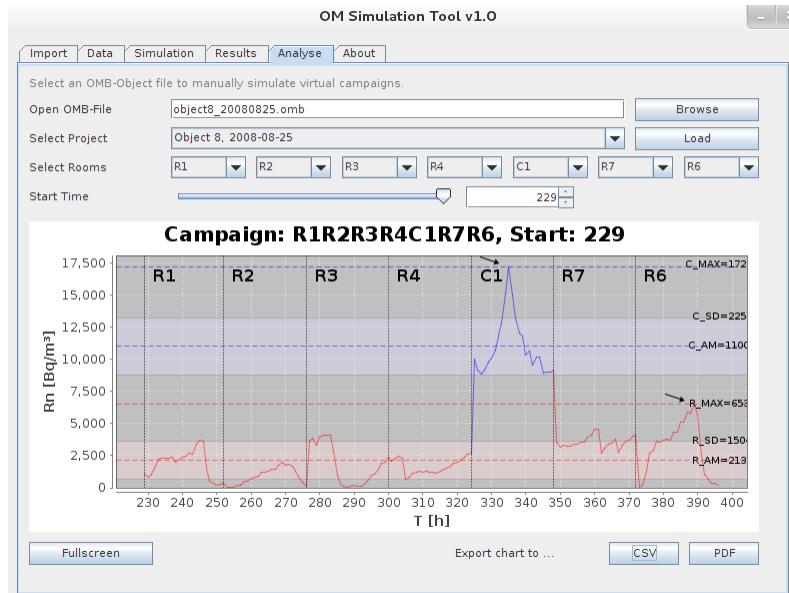


Abbildung 8.1: Screenshot des fertigen Simulationstools

Die am 14. Mai 2012 veröffentlichte Version 1.0 stellt zunächst die abschließende stabile Version dieses Projektes dar (Abbildung 8.1). Diese implementiert im Wesentlichen alle zuvor definierten Anforderungen und ermöglicht darüber hinaus durch die grafische Visualisierung der Simulationsergebnisse eine detaillierte Auswertung. Durch die Offenlegung des Quellcodes bei Github ist einerseits die stetige

Verfügbarkeit für Interessierte an der Thematik gewährleistet und andererseits die Möglichkeit geboten, das Programm bei Bedarf weiter zu entwickeln oder auf andere, ähnliche Simulationsproblemstellungen umzumodellieren.

8.2 Ausblick

Im Anschluss an diese Arbeit erfolgt durch ein Expertenteam am Bundesamt für Strahlenschutz die ausführliche Simulation orientierender Messungen unter verschiedenen Bedingungen sowie eine detaillierte Auswertung der Simulationsergebnisse. Dadurch wird eine Evaluation der orientierenden Messungen gemäß des Schnellverfahrens „6+1“ ermöglicht, welche beschreibt wie brauchbar die Ergebnisse derartiger Messkampagnen in Hinsicht auf die Radonbelastung in Gebäuden sind. Stellt sich dabei heraus, dass dieses Messprotokoll durchaus zuverlässig erste Aussagen über die Belastung durch Radon zulässt, könnte sich hier schnell ein Standard für orientierende Messungen entwickeln. Für bewertende Messungen werden aber weiterhin Langzeitmessverfahren herhalten müssen. Abschließend seien zukünftige Veröffentlichungen um Dr. Bernd HOFFMANN et al. durch das BUNDESAMT FÜR STRAHLENSCHUTZ oder das BUNDESMINISTERIUM FÜR UMWELT, NATURSCHUTZ UND REAKTORSICHERHEIT zu dieser Thematik empfohlen.

Quellenverzeichnis

A Literatur

- [1] Rolf Assfalg, Udo Goebels und Heinrich Welter. *Internet Datenbanken. Konzepte, Methoden, Werkzeuge.* Bonn: Addison Wesley Longman Verlag, 1998.
- [2] Graham Bath und Judy McKay. *Praxiswissen Softwaretest. Test Analyst und Technical Test Analyst.* Heidelberg: Dpunkt Verlag, 2011.
- [3] René Baumert u. a. *Erarbeitung fachlicher Grundlagen für die Entwicklung zeit- und kosteneffektiver Verfahren zur Bestimmung von Strahlenexpositionen durch Radon in Wohnungen.* Abschlussbericht zum Vorhaben St.Sch. 4534. Dresden: IAF - Radioökologie GmbH, 2010.
- [4] Bayrisches Landesamt für Umwelt. *Umweltwissen. Radon in Gebäuden.* Broschüre. München, 2008.
- [5] Oliver Böhm. *Java Software Engineering unter Linux.* Nürnberg: SuSE PRESS, 2002.
- [6] Walter R. Bischofberger und Gustav Pomberger. *Prototyping-Orientated Software Development. Concepts and Tools.* Hrsg. von David Gries. Berlin, Heidelberg, New York: Springer-Verlag, 1992.
- [7] Gert Blumenthal, Dietmar Linke und Siegfried Vieth. *Chemie. Grundwissen für Ingenieure.* Wiesbaden: B. G. Teubner Verlag, 2006.
- [8] Hartmut Bossel. *Systeme, Dynamik, Simulation. Modellbildung, Analyse und Simulation komplexer Systeme.* Norderstedt: Books on Demand, 2004.
- [9] Tim Bray u. a. *Extensible Markup Language (XML) 1.0 (Fifth Edition) - W3C Recommendation.* Aufgerufen am: XYZ. 1998. URL: <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [10] Deryck F. Brown und David Anthony Watt. *Programming Language Processors in Java. Compilers and Interpreters.* Edinburgh: Pearson Education Limited, 2000.
- [11] Rebecca Bulander, Wolfgang Gohout und Katja Specht. *Statistik für Wirtschaft und Technik.* München: Oldenbourg Verlag, 2012.
- [12] Bundesamt für Strahlenschutz. *Das Bundesamt für Strahlenschutz. Arbeitsschwerpunkte und Forschung.* Broschüre. Salzgitter, 2009.

- [13] Bundesamt für Strahlenschutz. *Ermittlung der Strahlenexposition durch Radon-222 und dessen kurzlebige Zerfallsprodukte 1*. Aufgerufen am: 21.05.2012. URL: http://www.bfs.de/de/ion/radon/messmethoden_radonmesstechnik.pdf.
- [14] Bundesamt für Strahlenschutz. *Jahresbericht 2010*. Jahresbericht. Salzgitter, 2011.
- [15] Bundesamt für Strahlenschutz. *Strahlenschutzforschung*. Programmreport 2009. URN: URN:NBN:DE:0221-201009153217. Salzgitter, 2010.
- [16] Bundesamt für Strahlenschutz. *Strahlung & Strahlenschutz. Eine Information des Bundesamtes für Strahlenschutz*. Broschüre. Salzgitter, 2008.
- [17] Deutscher Bundesrat. *Drucksache 639/11. Vorschlag für Richtlinie des Rates zur Festlegung grundlegender Sicherheitsnormen für den Schutz vor Gefahren einer Exposition gegenüber ionisierender Strahlung*. Brüssel: Europäische Kommission, 2011.
- [18] Mike Cohn. *Agile Software-Entwicklung. Mit Scrum zum Erfolg*. München: Addison-Wesley, 2010.
- [19] Mike Cohn. *User Stories für die agile Software-Entwicklung mit Scrum, XP u.a.* Heidelberg, München, Landsberg, Frechen, Hamburg: Mitp Verlag, 2010.
- [20] John Dalbey. *The Pseudocode Standard*. Aufgerufen am: XYZ. 2003. URL: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html.
- [21] Rainer Dumke. *Software Engineering. Eine Einführung für Informatiker und Ingenieure*. 4. Auflage. Wiesbaden: Vieweg Verlag, 2003.
- [22] Andreas Frick. *Der Software-Entwicklungs-Prozess. Ganzheitliche Sicht*. München, Wien: Carl-Hanser-Verlag, 1995.
- [23] Björn Gehlsen und Bernd Page. »Parallel and Distributed Simulation«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [24] Karl Gertis. *Radon in Gebäuden. Eine kritische Auswertung vorhandener Literatur*. Stuttgart: Frauenhofer IRB Verlag, 2008.
- [25] Joachim Gräser u. a. *Radon: Messung und Bewertung*. Aufgerufen am: XYZ. 2011. URL: http://www.ages.at/uploads/media/Radon_-_Messung_und_Bewertung_02.pdf.
- [26] Joachim Gräser u. a. »Radonsanierungen im Kanton Tessin (Schweiz)«. In: *5. Sächsischer Radontag - 7. Tagung Radonsicheres Bauen*. Hrsg. von Marcus Hoffmann. Centro di Competenza Radoon CCR, SUPSI, Canobbio-Lugano (CH). 2011.
- [27] Thomas Güttler. *Objektorientierte Datenbank ZODB - Serialisierung*. Aufgerufen am: 25.01.2012. 2005. URL: http://www.thomas-guettler.de/vortraege/zodb/einfuehrung.html#link_4.1.
- [28] Valentin Haenel und Julius Plenz. *Git. Verteilte Versionsverwaltung für Code und Dokumente*. München: Open Source Press, 2011.

- [29] Steve Holzner. *Eclipse*. Köln: O'Reilly Verlag, 2004.
- [30] Andreas Korff. *Modellierung von eingebetteten Systemen mit UML und SysML*. Heidelberg: Spektrum Akademischer Verlag, 2008.
- [31] Wolfgang Kreutzer. »Object-Orientated System Development and Simulation«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [32] Hans Peter Latscha und Helmut Alfons Klein. *Anorganische Chemie. Chemie-Basiswissen I*. Berlin, Heidelberg, New York: Springer, 2007.
- [33] Jens Lehmbach. *Vorgehensmodelle im Spannungsfeld traditioneller, agiler und Open-Source-Softwareentwicklung*. Stuttgart: Ibidem-Verlag, 2007.
- [34] Jon Loeliger. *Versionskontrolle mit Git. Projekte effizient und flexibel verwalten*. Köln: O'Reilly Verlag, 2009.
- [35] Peter Mandl. *Masterkurs Verteilte Betriebliche Informationssysteme*. Wiesbaden: Vieweg + Teubner, 2009.
- [36] Bernd Matzke. *Ant. Eine praktische Einführung in das Java-Build-Tool*. Heidelberg: dpunkt Verlag, 2005.
- [37] Carlo Ripa di Meana. *EU-Richtlinie 90/143/EURATOM. zum Schutz der Bevölkerung vor Radonexposition in Gebäuden*. Brüssel: Europäische Kommission, 1990.
- [38] Bernd Page. »Simulation in Practice«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [39] Bernd Page, Nicolas Knaack und Ruth Meyer. »First Steps in Simulation Programming«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [40] Bernd Page und Wolfgang Kreutzer. »Introduction and Basic Terms«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [41] Bernd Page, Wolfgang Kreutzer und Volker Wohlgemuth. »Simulation Software«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [42] Bernd Page, Julia Kuck und Johannes Göbel. »Simulation Statistics«. In: *Simulating Discrete Event Systems with UML and Java*. Hrsg. von Bernd Page und Wolfgang Kreutzer. Aachen: Shaker Verlag, 2005.
- [43] Dominik Panic, Tobias Schnackenbeck und Volker Wohlgemuth. »Eine offene Anwendungsarchitektur als Fundament eines Methodenbaukastens für betriebliche Umweltinformationssysteme«. In: *Simulation in den Umwelt und Geowissenschaften*. Hrsg. von Jochen Wittmann und Volker Wohlgemuth. Aachen: Shaker Verlag, 2007.
- [44] Claus Rautenstrauch. *Betriebliche Umweltinformationssysteme. Grundlagen, Konzepte und Systeme*. Berlin, Heidelberg: Springer Verlag, 1999.

- [45] Duri Schmidt. *Persistente Objekte und Objektorientierte Datenbanken*. München, Wien: Carl Hanser Verlag, 1991.
- [46] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files - RFC 4180*. Aufgerufen am: XYZ. 2005. URL: <https://tools.ietf.org/html/rfc4180>.
- [47] Frauke Strümpel. »Simulation zeitdiskreter Modelle mit Referenzen«. Diplomarbeit. Hamburg: Universität Hamburg, 2003.
- [48] Georg Erwin Thaller. *Software-Test. Verifikation und Validation*. Hannover: Verlag Heinz Heise, 2002.
- [49] C. L. Tondo, A. Nathanson und E. Yount. *Das Softwarewerkzeug Make*. München: Prentice Hall Verlag, 1995.
- [50] Philipp von Wartburg. *Limitationen in Microsoft Excel*. Aufgerufen am: 25.01.2012. 2010. URL: <http://www.xlam.ch/xlimits/excel-new.htm>.
- [51] World Health Organisation. *WHO Handbook on indoor Radon. A public health perspective*. Hrsg. von Hajo Zeeb und Ferid Shannoun. Genf: WHO Press, 2009.

B Internetverweise

- [52] Apache Software Foundation. *The Apache Ant Project*. Aufgerufen am: 29.05.2012. URL: <https://ant.apache.org/>.
- [53] Software Foundation Apache. *Apache Commons Math - Statistics*. Aufgerufen am: 02.02.2012. 2012. URL: <https://commons.apache.org/math/userguide/stat.html>.
- [54] Software Foundation Apache. *Apache Subversion - Enterprise-class centralized version control for the masses*. Aufgerufen am: XYZ. 2004. URL: <http://subversion.apache.org/>.
- [55] Balsamiq. *Balsamiq Mockups*. Aufgerufen am: 12.05.2012. 2012. URL: <http://www.balsamiq.com/products/mockups>.
- [56] Bundesamt für Strahlenschutz. *Einführung Radon*. Aufgerufen am: 19.05.2012. 2011. URL: <http://www.bfs.de/de/ion/radon/einfuehrung.html>.
- [57] BIT Bundesstelle Für Informationstechnik. *Kompetenzzentrum Open-Source-Software*. Aufgerufen am: XYZ. 2011. URL: http://www.bit.bund.de/BIT/DE/Beratung/CC_OSS/node.html.
- [58] BVA Bundesverwaltungsamt. *Das Kompetenzzentrum Open Source Software*. Aufgerufen am: XYZ. 2011. URL: http://oss.bund.de/ueber-uns#CC_OSS.
- [59] BVA Bundesverwaltungsamt. *Open-Source-Software Lizizenzen*. Aufgerufen am: XYZ. 2011. URL: <http://www.oss.bund.de/node/123>.
- [60] BVA Bundesverwaltungsamt. *Pro und Contra Open-Source-Software*. Aufgerufen am: XYZ. 2011. URL: <http://www.oss.bund.de/node/133>.

- [61] Scott Chacon und Linus Torvalds. *Git - The fast version control system.* Aufgerufen am: XYZ. 2005. URL: <http://git-scm.com/>.
- [62] EyeDB. *EYEDB - open source object database.* Aufgerufen am: 25.01.2012. 2012. URL: <http://db4o.com/>.
- [63] Firebird. *Firebird Database.* Aufgerufen am: XYZ. 2012. URL: <http://firebirdsql.org/>.
- [64] Inc Free Software Foundation. *GNU General Public License (GPL) Version 3.* Aufgerufen am: XYZ. 2007. URL: <https://www.gnu.org/licenses/gpl-3.0.html>.
- [65] A. B. Gammadata Instrument. *Radon Gas Monitor Atmos 12 DPX.* Aufgerufen am: XYZ. 2008. URL: http://www.gammadatainstrument.se/_resources/File/ATMOS_datasheet_ENG_v1.4.pdf.
- [66] H2-Database. *Database Connection Modes.* Aufgerufen am: 25.01.2012. 2010. URL: http://www.h2database.com/html/features.html#connection_modes.
- [67] H2-Database. *Database Performance Comparison.* Aufgerufen am: 25.01.2012. 2010. URL: http://h2database.com/html/performance.html#performance_comparison.
- [68] H2-Database. *H2 Database Engine.* Aufgerufen am: XYZ. 2012. URL: <http://h2database.com/html/main.html>.
- [69] HyperSQL. *HSQLDB - 100% Java Database.* Aufgerufen am: XYZ. 2012. URL: <http://hsqldb.org/>.
- [70] P.J. Hyett, Tom Preston-Werner und Chris Wanstrath. *Github.* Aufgerufen am: XYZ. 2008. URL: <http://github.com/>.
- [71] JFree. *JFreeChart.* Aufgerufen am: 14.05.2012. 2012. URL: <http://www.jfree.org/jfreechart/>.
- [72] McObject. *Perst - An open source, object-oriented embedded database.* Aufgerufen am: 25.01.2012. 2012. URL: <http://mcoobject.com/perst>.
- [73] Rutherford Online. *Lexikon der Elemente: Elementbeschreibung Radon.* Aufgerufen am: 21.05.2012. 2006. URL: <http://www.unterra.de/rutherford/ele086.htm>.
- [74] The Open Source Initiative. *The Open Source Definition.* Aufgerufen am: XYZ. 2012. URL: <http://opensource.org/docs/osd>.
- [75] Oracle. *Java SE Technologies - Database.* Aufgerufen am: XYZ. 2012. URL: <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>.
- [76] RadonEnvironmental. *Understanding How Radon Enters A House.* Aufgerufen am: 21.05.2012. URL: <http://www.radonenvironmental.co.uk/radon-gas/how-does-radon-enter-a-home.html>.
- [77] SQLite. *SQLite Database.* Aufgerufen am: XYZ. 2012. URL: <https://sqlite.org/>.

- [78] Sourceforge. *Excel-JDBC-Driver*. Aufgerufen am: XYZ. 2004. URL: <http://xsql.sourceforge.net/>.
- [79] BV TIOBE Software. *TIOBE Programming Community Index for April 2012*. Aufgerufen am: XYZ. 2012. URL: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.
- [80] Linus Torvalds. *Linux Makefile: Commit 693d92a*. Aufgerufen am: XYZ. 2011. URL: <http://github.com/torvalds/linux/commit/693d92a1bbc9e42681c42ed190bd42b636ca876f>.
- [81] Versant. DB4O - *Database for Objects*. Aufgerufen am: 25.01.2012. 2012. URL: <http://db4o.com/>.
- [82] Wikimedia. *Wikimedia Traffic Analysis Report - Operating Systems*. Aufgerufen am: XYZ. 2012. URL: http://stats.wikimedia.org/archive/squid_reports/2012-03/SquidReportOperatingSystems.htm.
- [83] Volker Wohlgemuth. *Emporer-Projekt*. Aufgerufen am: XYZ. 2006. URL: <http://www.emporer.net/emporer>.