Institut für Softwaretechnologie
Abteilung Software Engineering

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Fachstudie Nr. 84

Eignung von Java für mobile Rechner mit Windows CE/Mobile

Tobias Arnold Ivan Bogicevic Nico Gau

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Jochen Ludewig

Betreuer: Markus Knauß

begonnen am: 22.10.2007

beendet am: 21.04.2008

CR-Klassifikation: D.2.7

Versionshistorie

Version	Datum	Bearbeiter	Kommentar
0.1	09.02.2008	Tobias Arnold	Erstes Inhaltsverzeichnis
0.2	09.02.2008	Nico Gau, Tobias Arnold	Ergänzungen im Dialog
0.3	10.02.2008	Ivan Bogicevic	weitere Ergänzungen, Plattformenübersichtsbild erstellt, Formatierung überarbeitet
0.4	13.02.2008	Tobias Arnold	Tabellen für GuiBewertung eingefügt Swing mit CreMe ausgefüllt
0.5	14.02.2008	Ivan Bogicevic	Inhalte ausformuliert, erstes Kapitel fertiggestellt
0.6	16.02.2008	Tobias Arnold	GUI-Bewertung für J9/eRCP eingefügt
0.7	17.02.2008	Ivan Bogicevic	Analyse-Teil ausgearbeitet
0.8	18.02.2008	Ivan Bogicevic	Sehr viel ausformuliert, Szenario und rundenbasierte Untersuchung hinzugefügt
0.9	28.02.2008	Ivan Bogicevic	Umstrukturierung der Kapitel, weitere inhaltliche Ergänzungen
1.0	08.03.2008	Ivan Bogicevic	verschiedene Kürzungen und Ergänzungen
1.1	09.03.2008	Tobias Arnold	Gui
1.2	09.03.2008	Nico Gau	Ergebnisse Microbenchmarks
1.3	10.03.2008	Ivan Bogicevic	Titelblatt, Kapitel Portierung, Planungsdiagramme, Ausformulierungen
1.4	10.03.2008	Nico Gau, Ivan Bogicevic	Endergebnis ausformuliert, Design angepasst
1.5	10.03.2008	Nico Gau, Ivan Bogicevic	Änderungen nach Treffen mit Betreuer
1.6	10.03.2008	Tobias Arnold	GUI Lücken gefüllt, GUI Fazit, Inbetriebnahme Fazit
1.7	11.03.2008	Nico Gau	Microbenchmarks überarbeitet
1.8	12.03.2008	Ivan Bogicevic	Komplette Überarbeitung
1.9	14.03.2008	Ivan Bogicevic	Änderungen aus Review mit Tobias Arnold, Nico Gau, Ivan Bogicevic übenommen
2.0	14.03.2008	Nico Gau	Testrahmeninformationen korrigiert
2.1	18.03.2008	Ivan Bogicevic	Änderungen nach Review mit Betreuer
2.2	25.03.2008	Ivan Bogicevic, Tobias Arnold, Nico Gau	Änderungen nach Treffen mit dem Industriepartner

Zusammenfassung

In dieser Fachstudie werden mehrere Möglichkeiten untersucht, Anwendungen auf mobilen Rechnern mit Windows CE/Mobile zu betreiben. Insbesondere findet dabei ein Vergleich mehrerer Konfigurationen mit der .NET-Plattform von Microsoft und mit der Java ME-Plattform statt. Bestandteile dieser Untersuchung sind insbesondere Microbenchmark-Tests, GUI-Untersuchungen, sowie Aspekte der Portierung eines vorhandenen Programmes von .NET nach Java.

Inhaltsverzeichnis

Versionshistorie **3**

Zusammenfassung **5**

1 Allgemeines **13**
1.1 Einleitung 13
1.2 Vorgehen 13
1.2.1 Übersicht 13
1.2.2 Kriterien 14
1.2.3 Testrahmen 14

2 Analyse **15**
2.1 Kundenanalyse 15
2.2 PDA-Modell 15
2.3 Software 15
2.4 Erfahrungen 15
2.5 Ist-Stand des Programms 15
2.5.1 Grundarchitektur 15
2.5.2 Programmablauf 16
2.5.3 GUI/Steuerelemente 16
2.6 Wichtige Aspekte 17
2.7 Java auf mobilen Geräten im Allgemeinen 17
2.8 Literaturrecherche 17
2.8.1 Zweck 17
2.8.2 Vorgehensweise 17
2.8.3 Bewertungskriterien 17
2.8.4 Literaturliste 18
2.8.5 Fazit 18

3 Plattformauswahl **19**
3.1 Szenario 19
3.2 Übersicht 19

3.3 Office-Lösung 19
3.4 Windows nativ 20
3.5 Alternative Plattformen 20
3.6 Papier-Lösung 20
3.7 Plattformen für .NET 20
3.7.1 .NET Framework 20
3.7.2 .NET Compact Framework 1.0 21
3.7.3 .NET Compact Framework 3.5 21
3.7.4 andere Versionen von .NET CF 21
3.8 Plattformen für Java 21
3.8.1 Konfigurationen von Java ME 21
3.8.2 Java ME via IBM J9 21
3.8.3 Java ME via CreMe 22
3.8.4 Java ME via PhoneME 22
3.8.5 Java SE via MySaifu 22
3.9 GUI-Bibliotheken für Java 22
3.9.1 AWT 22
3.9.2 Swing 22
3.9.3 eRCP 22

4 Plattformvergleich 25
4.1 Einleitung 25
4.2 Runde I – Art der Lösung 25
4.3 Runde II – Edition der Plattform 27
4.4 Runde III – Welche konkrete VM 29
4.5 Runde IV – Kombination mit GUI-Bibliotheken 31
4.6 GUI Bewertung 32
4.6.1 Bewertungsmethode 32
4.6.2 Allgemeines 33
4.6.3 Container 33
4.6.4 Layoutmanager 35
4.6.5 Standard Elemente 36
4.6.6 Spezielle Elemente 39
4.6.7 Fazit 40
4.7 Microbenchmark-Test 40
4.7.1 Testfälle 40

4.7.2 Ergebnisse ... 42
4.7.3 Fazit ... 42
4.8 Entwicklungsumgebung ... 45
4.8.1 J9 und eRCP ... 45
4.8.2 CreME und Swing ... 46
4.8.3 .NET Compact Framework 1.0 ... 47
4.8.4 .NET Compact Framework 3.5 ... 48
4.8.5 Fazit ... 49
4.9 Inbetriebnahme ... 49
4.9.1 J9 und eRCP ... 49
4.9.2 CreME und Swing ... 49
4.9.3 .NET Compact Framework 1.0 und 3.5 ... 50
4.9.4 Fazit ... 50

5 Portierung ... 51
5.1 Vorgehensweisen ... 51
5.2 Automatische Portierung ... 52
5.3 Fazit ... 53

6 Untersuchungsergebnisse ... 55
6.1 Empfohlene Plattform ... 55
6.2 Empfehlung für den Industriepartner ... 56

7 Rückblick ... 57

8 Rahmenbedingungen und Planung der Fachstudie ... 59
8.1 Beteiligte Personen ... 59
8.2 Vertragliche Anforderungen an die Projektdurchführung ... 59
8.3 Lieferumfang ... 59
8.4 Leistungen der Auftraggeber ... 59
8.4.1 Industriepartner ... 59
8.4.2 Universität ... 59
8.5 Richtlinien für die Fachstudie ... 60
8.5.1 Konfigurationsmanagement ... 60
8.5.2 Design-Richtlinien ... 60
8.6 Entwicklungsprozess ... 60

8.7 Externe Meilensteine 60
8.8 Phasen der Entwicklung 60
8.9 Dokumentationsplan 60
8.9.1 Produktdokumente 60
8.9.2 Nachführung 60
8.10 Prüfungen 61
8.11 Abnahmeprozedur 61
8.12 Berichtswesen 61
8.13 Risiken 61
8.13.1 Mangelnde Hardwareinfrastruktur 61
8.13.2 Probleme mit der Softwareinfrastruktur 61
8.13.3 Kostenpflichtige Software 61
8.13.4 Verzögerungen 61
8.13.5 Ausfall eines Teammitglieds 62
8.14 Gantt-Diagramm 62
8.15 Termindrift-Diagramm 63

Anhang A – Literaturliste 65
.Net Compact Framework 65
Java ME 66
Vergleiche zwischen .NET Compact und Java ME 67
Vergleiche zwischen .NET und Java SE & Java EE 74

Anhang B – Microbenchs 77
Testfälle 77
Ergebnisse Compact Framework 1.0 91
Ergebnisse Compact Framework 3.5 92
Ergebnisse J9 93

Anhang C – Phasen der Entwicklung 95
Planungsphase (22.10.2007 – 05.11.2007) 95
Voranalyse (06.11.2007 – 21.12.2007) 95
Analyse (06.01.2008 – 21.01.2008) 95
Evaluierung (22.01.2008 – 18.02.2008) 96
Projektabschluss (19.02.2008 – 21.04.2008) 96

1 Allgemeines

1.1 Einleitung

Für Schäden, die durch Hagel verursacht wurden, gibt es eine Spezialversicherung, bei der Kunden im Agrarbereich ihre Anbauflächen gegen solche Hagelschäden versichern können. Hierfür bietet die Versicherung ihren Kunden eine Webanwendung, mit der Kunden ihre Felder und darauf angebaute Fruchtgattungen verwalten und versichern können. Diese Anwendung wurde von der aformatik GmbH & Co. KG (http://www.aformatik.de) entwickelt und wird seit Jahren gepflegt.

Tritt ein Hagelschaden ein, wird dieser durch einen fachkundigen Schätzer vor Ort bewertet. Die Bewertung wird auf Formularen dokumentiert und orientiert sich an der Schadensart und der Fruchtgattung, wobei die Vielfalt der Formulare hoch ist. Eine elektronische Schadenserfassung ist ebenfalls verfügbar. Diese ist eine Client-Server-Anwendung, mit der die Schätzer vor Ort mit Hilfe mobiler Windows CE/Mobile basierter Geräte (PDAs) den Schaden dokumentieren und zur Schadensregulierung mit einer in Java realisierten Serveranwendung abgleichen. Der Abgleich erfolgt über Web-Services. Der Prototyp der Software für die mobile Datenerfassung wurde von einem externen Dienstleister in Microsoft VisualBasic.NET CF (Compact Framework) v1.0 implementiert und wird jetzt von der aformatik GmbH & Co. KG weiterentwickelt.

Es hat sich erwiesen, dass die Implementierung der Client-Software in VisualBasic.NET in einigen Punkten weiter optimiert werden kann. Die Gründe hierfür sind:
- Die **Performanz** ist teilweise nicht ausreichend.
- Einige Anforderung, wie etwa spezielle **GUI-Elemente**, sind in VisualBasic.NET CF v1.0 nur durch „workarounds" lösbar.
- Für Server und Client müssen **verschiedene Entwicklungs- und Testumgebungen** eingesetzt werden.
- Die aformatik GmbH & Co. KG weist hohe **Kompetenzen in der Java-Technologie** auf.

Deshalb bietet es sich an, falls sinnvoll, die Client-Software auf Basis einer Java Plattform neu zu implementieren. Hierfür kommt beispielweise Java ME (Micro Edition) in Betracht.

Darauf bezogen wird in dieser Fachstudie die Eignung von Java für mobile Rechner mit Windows CE/Mobile, auch im Vergleich zur .NET-Technologie, untersucht. Insbesondere wird festgestellt, welche Plattform in welcher Konfiguration am besten für die Anforderungen der mobilen Schadenserfassung geeignet ist.

1.2 Vorgehen

1.2.1 Übersicht

Zunächst wird in diesem Projektbericht neben allgemeinen Aussagen das Vorgehen in dieser Fachstudie kurz beschrieben.
Es folgen die Ergebnisse aus der **Analyse** inklusive einer Literaturrecherche.
Der Hauptteil besteht aus der Untersuchung der verschiedenen Möglichkeiten, die oben beschriebene Anwendung zur mobilen Schadenserfassung zu betreiben und zu entwickeln. Diese Untersuchung erfolgt in zwei Schritten:
Im ersten Schritt, der **Plattformauswahl**, entscheidet sich, welche Plattformen wie etwa Java ME und .NET in welchen Konfigurationen, also auch mit welchen Bibliotheken und Einstellungen, für die geforderte Anwendung überhaupt in Betracht kommen.
Im zweiten Schritt, beim **Plattformvergleich**, werden diese Konfigurationen aufgegriffen und einzeln untersucht. In mehreren Runden werden sie dabei immer näher eingegrenzt, bis zuletzt die besten Konfigurationen übrig bleiben. Diese werden dann in den letzten Runden detaillierteren **GUI**- und **Microbenchmark**-Tests unterzogen.

Es folgt eine Betrachtung der verschiedenen **Entwicklungsumgebungen** der einzelnen Technologien. Zuletzt wird der jeweilige Aufwand dargestellt, die **Inbetriebnahme** einer Anwendung durchzuführen. Das bedeutet, es werden die nötigen Schritte dargestellt, ein Programm „zum Laufen“ zu bekommen.
Danach werden im Teil **Portierung** Möglichkeiten und Strategien beschrieben, das bestehende Programm von .NET CF in ein Java ME-Programm umzuwandeln. Vor- und Nachteile und der Aufwand für eine automatische (maschinelle) und für eine manuelle Portierung werden hierbei berücksichtigt.
Die **Untersuchungsergebnisse** beinhalten eine abschließenden Empfehlung bezüglich der Plattformen und der Portierung.
Es folgt ein **Rückblick**, in dem unser Vorgehen und unsere gesammelten Erfahrungen aus dem Fachstudien-Projekt kurz reflektiert werden.
Der Projektbericht endet mit einem Kapitel über die **Rahmenbedingungen und Planung der Fachstudie**, in dem unter anderem Richtlinien, vertragliche Anforderungen und weitere Details wie etwa Zeitplan und nachträgliche Risikobewertung zu finden sind.

1.2.2 Kriterien

Beim Vergleich wurden festgelegte Kriterien beachtet. So wurde bei der Bewertung der GUI berücksichtigt, ob und mit welchem Aufwand bestimmte GUI-Elemente realisierbar sind, wie gut sie benutzt werden können und welches Aussehen sie haben, wovon letztere Bewertung teilweise subjektiv ist.
Bei den Microbenchmark-Tests war es wichtig, realistische Operationen zu testen, die in Anwendungen üblich sind und auch speziell in der Anwendung zur mobilen Schadenserfassung häufig vorkommen.

1.2.3 Testrahmen

Getestet wurde mit einem mobilen Rechner (PDA) des Modells „MDA Pro“ mit einem 520MHz ARM-Prozessor und Windows Mobile 5.0. Dieser wurde per USB-Kabel an einen Windows-XP Rechner angeschlossen und per ActiveSync verbunden. So konnte die Testapplikationen entweder manuell hochgeladen werden, oder es konnte eine direkte Verbindung zu Visual Studio aufgebaut werden.

2 Analyse

2.1 Kundenanalyse

Am 28.11.2007 wurde bei einem gemeinsamen Treffen mit dem Industriepartner (aformatik GmbH & Co. KG) vor Ort eine Kundenanalyse durchgeführt. Die Ergebnisse daraus werden zusammen mit einigen Aspekte die sich später ergeben haben hier vorgestellt.

2.2 PDA-Modell

Auf folgendem PDA wird die oben beschriebene mobile Anwendung zur Schadenserfassung entwickelt:

- PDA-Modell „XDA Mini“
- hier als Telefonversion mit 300MHz
- ARM-Prozessor
- Gerät ist nicht Zielgerät, auf dem es eingesetzt wird

Bei schnelleren PDA-Modellen ist die Performanz jedoch ausreichend.

2.3 Software

Folgende Versionen an Software werden verwendet:

- Pocket PC 2003
- Windows Mobile 5, aber auch 2003 second edition
- Microsoft .NET CF (Compact Framework) v1.0 (Laut damaligen Informationen sei damals 2.0 zu langsam und der Aufwand für den Umstieg auf 3.0 sei zu hoch)
- Android wird (noch) nicht verwendet, könnte aber auch interessant sein
- Visual Studio 2003 (nicht 2005)

Es gibt einen Emulator, dieser ging aber teilweise nicht, es traten Treiberprobleme mit Windows auf.
Ein grafischer Code-Editor ist nicht erwünscht und braucht auch nicht weiter untersucht werden, da er schlechten Code erzeuge.

2.4 Erfahrungen

- Objektorientierte Programmierung erzeuge langsamere Programme.
- Die Präferenz des Industriepartners liegt bei Java.
- Bekannte Einschränkungen von .NET CF v1.0 seien:
  - Es gibt keine Scrollpane, die Container scrollt.
  - Es gibt keinen Layoutmanager.
- Es wurden bereits kleinere Tests mit Java durchgeführt, dann jedoch verworfen, weil:
  - Scrollpanes zu langsam sind.
  - Die Partnerfirma hatte .NET verwendet.

2.5 Ist-Stand des Programms

2.5.1 Grundarchitektur

Server: hostet WebService
PDA (Client): Es wird .NET CF benutzt, um auf Webservices zuzugreifen.

Folgende 3-Schichtenarchitektur wurde vollständig umgesetzt:

- Präsentation (GUI)
  - Modul Anfangsteil
  - Modul Regulierungsassistent
- Business-Applikation (Logik) als Singleton

    - Es gibt einen „Manager“ für jeden Bereich der Anwendung.
    - Für jeden Use-Case wird eine eigene Klasse benutzt.
    - Jede Seite der Eingabe entspricht einem „Baustein“.
- Datenbank (persistent)
    - Für jede DB-Tabelle gibt es eine eigene Klasse (zur Kapselung).

2.5.2 Programmablauf

- DB-Struktur wird angelegt (über create-statements)
    - nur falls noch keine da ist
    - zwei Teile: eine für die queue, eine für Applikations-Daten
- Login-Seite wird aufgerufen
    - Login-Daten werden über den Server überprüft und lokal gespeichert
    - Falls bereits lokale Login-Daten vorhanden sind, ist auch ein Offline-Login möglich (ab zweitem Einloggen)
- Menü „Partnerauswahl“ ist folgendermaßen aufgebaut:
    - Schätzung
    - Positionen: Eine Position wird ausgewählt und dann mit dem Regulierungsassistent „reguliert“
    - Vorbesichtigung
    - Zwischenbesichtigung
    - Endregulierung
    - Unterschrift
- Aussagen zum Regulierungsassistent
    - Er zeigt die Bausteine an (1 Seite = 1 Baustein).
    - Pro Baustein wird lokal die Eingabe gespeichert, erst am Ende der Phase (aller Bausteine) werden die Daten mit dem Server abgeglichen.
- Sonstiges:
    - Die Performance der DB-Zugriffe ist möglicherweise noch besser optimierbar.
    - Zu Webservices: Ein Paket wird zwischengespeichert, wenn der Server nicht verfügbar ist (wie in einer queue).
    - Zur Datenübertragung: Keine Verwendung von XSD-Typen oder verschachtelten Ausdrücken (aus Performance-Gründen), sondern Austausch durch CSV-Daten.

2.5.3 GUI/Steuerelemente

- Im Programm sind sehr viele
    - Radiobuttons
    - Labels
    - Textfelder
    - Buttons
- weniger vorhanden, aber durchaus verwendet werden
    - Checkboxen
    - Spinner („numeric up-down“)
    - Skizze (PictureBox)
    - DatePicker (von OpenNet CF)
    - GroupBoxen (von OpenNet CF)
    - ListView (Tabelle)
- besondere Eigenschaften
    - Scrollpanes im Programm sind sehr langsam.
    - Listen und Scrollpanes sind teilweise selbst programmiert (über direkte Grafikprogrammierung), um eine bessere Performanz zu erreichen.
    - Beim Start des Assistenten werden die Labels der Elemente geladen.

  - Pro Seite gibt es nur so viele GUI-Elemente wie auf einmal auf den Schirm passen (oder etwas mehr).
  - Das Aussehen der zukünftigen GUI muss nicht exakt dem jetzigen Stand entsprechen.

2.6 Wichtige Aspekte

Folgende zu untersuchenden Aspekte und Ergebnisse der Fachstudie sind für den Industriepartner wichtig

- **Unterschiede** zwischen .NET CF und Java
- **Performance**
- Die zu verwendende Entwicklungsart sollte
  - **einfach** und
  - **objektorientiert** sein
- Das Ergebnis einer Entwicklung sollte so
  - **wartbar** und
  - **modular** wie möglich sein

2.7 Java auf mobilen Geräten im Allgemeinen

Folgende allgemeine Fragen zur Plattformauswahl wurden als wichtig erachtet. Sie werden im Abschnitt über allgemeine Fragen zur Plattformauswahl beantwortet:

- Kann die .NET und die Java-Technologie prinzipiell überhaupt verwendet werden, oder gibt es grundsätzliche Argumente dagegen?
- Welche konkreten Programmiersprachen bieten sich für .NET an?
- Ist speziell VisualBasic.NET geeignet, oder gibt es bessere Alternativen?
- Welche Vor- und Nachteile gibt es zwischen Java SE und Java ME?
- Welche weiteren Möglichkeiten gibt es, Java-Anwendungen zu betreiben?

2.8 Literaturrecherche

2.8.1 Zweck

Hier werden die Ergebnisse der Literaturrecherche dargestellt. Es werden die untersuchten Quellen aufgelistet und bewertet. Zum Einen wurde recherchiert, ob es schon vergleichende Literatur zum Thema Java ME vs. .NET CF gibt, zum Anderen werden Quellen für Beschreibungen und Anleitungen zu den einzelnen Themen zusammengefasst.

2.8.2 Vorgehensweise

Für die Literaturrecherche wurde vor allem die normale google-Suche verwendet. Die besten Ergebnisse lieferten Suchworte wie „Java ME vs. .NET compact“, „Java mobile“, „.NET mobile“. In der zweiten Stufe wurde eine Recherche mit google scholar durchgeführt, um wissenschaftliche Arbeiten zum Thema zu finden. Zuletzt wurden noch einige Personen, die sich aktuell und in der Vergangenheit mit dem Thema beschäftigt haben, angeschrieben. Die Bewertungen sind teilweise subjektiv.

2.8.3 Bewertungskriterien

Für die Bewertung der Quellen wird folgendes Schema verwendet, wobei die Bewertung teilweise subjektiv ist:

Link:	Der Link zu der beschriebenen Quelle (Stand Anfang 2008)
Quelle:	Einrichtung, die die Informationen bereitstellt, Datum
Typ:	Art der Quelle (z.B. Diplomarbeit)
Zusammenfassung:	Eine kurze Zusammenfassung des Inhalts der Quelle
Wissenschaftlicher Anspruch:	Eine Bewertung des wissenschaftlichen Anspruchs in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Beispielsweise hat ein Paper einer Fachkonferenz einen sehr hohen wissenschaftlichen Anspruch. Dem gegenüber steht beispielsweise der Blog eines Privatmanns.
Relevanz:	Eine Bewertung der Relevanz für das in der Fachstudie behandelte Thema in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Eine Quelle ist relevant, wenn das darin behandelte Thema in großen Teilen mit dem Thema dieser Fachstudie übereinstimmt.
Verlässlichkeit:	Eine Bewertung der Verlässlichkeit der Quelle in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Eine Quelle erscheint dann als verlässlich, wenn sie einem professionellen Umfeld entstammt.
Unvoreingenommenheit:	Eine Bewertung der Unvoreingenommenheit der Quelle in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Eine Quelle ist unvoreingenommen, wenn sie dem Thema objektiv gegenübersteht. Beispielsweise wären Sun und Microsoft voreingenommen.
Aktualität:	Eine Bewertung der Aktualität in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Da sich der technologische Rahmen der Fachstudie ständig ändert, ist eine Quelle nur dann aktuell, wenn sie möglich zeitnah an dem Fachstudiendatum veröffentlicht wurde (Stand 2007/2008).
Bewertung:	Eine allgemeine Bewertung des Nutzwertes der Quelle für die Fachstudie in Schulnoten: 1 (sehr gut) bis 6 (schlecht). Subjektive Einschätzung der Quelle unter Berücksichtigung der vorhergehenden Einschätzungen.

2.8.4 Literaturliste

Die vollständige Literaturliste mit den Bewertungen befindet sich in Anhang A.

2.8.5 Fazit

Nach Auswertung der Quellen wird eine Tendenz Richtung .NET CF hinsichtlich Performance und WebService-Integration sichtbar. Jedoch müssen eigene Benchmarks entwickelt werden, um ein konkretes Ergebnis zu erzielen. Anscheinend ist die VM von Java nicht so performant wie die JIT von .NET. Auch kann man bei .NET per P/Invoke Teile, die hochperformant laufen müssen, in C/C++ auslagern, was bei Java ME nicht möglich ist, da das JNI bei Java ME nicht vorhanden ist. Weiterhin zeigt die durchgeführte Literaturrecherche auf, dass nur sehr wenig bis keine aktuellen Vergleiche zwischen .NET CF und Java ME stattgefunden haben.

3 Plattformauswahl

3.1 Szenario

Die grundsätzliche Frage ist, wie in Zukunft mit der Applikation verfahren werden soll. Zunächst einmal gibt es folgende Möglichkeiten:

- .NET CF v1.0 wird beibehalten
- Wechsel zu einer anderen Version, man bleibt jedoch bei .NET CF
- Wechsel zu Java
- Wechsel zu einer komplett anderen Lösung

Da die Hauptkompetenz der aformatik im Java-Bereich liegt, ist zu bedenken, dass sich im Falle einer Entscheidung für Java positive Synergieeffekte ergeben. Die Firma könnte dadurch auf ihre langjährigen Erfahrungen in Java zurückgreifen. Im gegenteiligen Fall würde ein Aufwand für Fortbildungen und Einarbeitungen in der nicht-Java-Technologie entstehen. Dieser wäre bei Lösungen wie etwa der Android-Plattform besonders hoch, da sich diese von der bisherigen Lösung deutlich unterscheidet. Welche Alternativen es im Detail gibt, wird im Folgenden erläutert.

3.2 Übersicht

Folgende Abbildung gibt eine Übersicht über die vorgestellten und untersuchten Lösungen:

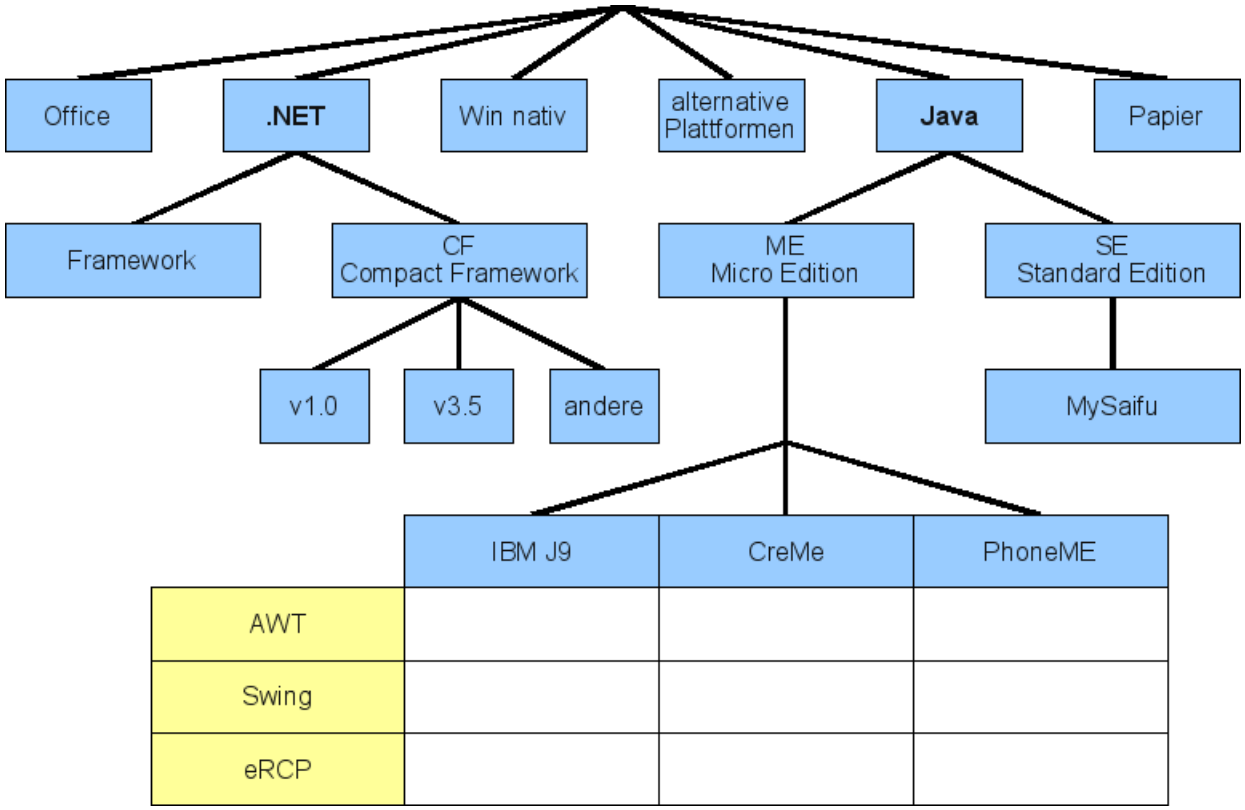


Abbildung 1: Übersicht der Lösungen

3.3 Office-Lösung

Eine sehr einfache Möglichkeit, den Anforderungen der mobilen Schadenserfassung zu einem gewissen Grad gerecht zu werden, wäre weg von einer separaten Applikation und hin zu einer Office-Lösung. So könnte z.B. der Schätzer den begutachteten Schaden in eine Excel-Tabelle oder in ein

Textdokument eintragen. Auch komplexere Formen, wie etwa editierbare Formulare im PDF-Format sind denkbar.

3.4 Windows nativ

Die Applikation könnte auch als direkte Windows Mobile 5.0 Anwendung geschrieben und verwendet werden, ohne virtuelle Maschine als Zwischenschicht. Hierbei müssten die von Windows angebotenen Bibliotheken direkt verwendet werden.

3.5 Alternative Plattformen

Außer Java und .NET gibt es noch weitere Plattformen, die für mobile Geräte geeignet sein könnten. Einige davon sind:

- **Symbian**
  Eine für Smartphones sehr weit verbreitete Alternative zu den verschiedenen Windows Versionen bietet Symbian OS. Symbian ermöglicht zum Einen das Ausführen von Programmen, die der JavaME Spezifikation entsprechen. Zum Anderen existieren speziell auf Symbian zugeschnittene Benutzeroberflächen, mit denen sich Anwendungen mit komplexen Oberflächen realisieren lassen.
  Ein Beispiel hierfür ist UIQ (http://www.uiq.com/). Zudem gibt es eine Unterstützung für verschiedene Skriptsprachen und einige relationale Datenbanksysteme.
- **Android**
  Android ist eine neue Open-Source Plattform für mobile Geräte. Das Betriebssystem soll es besonders einfach machen, Applikationen für den mobilen Bereich zu entwickeln. Eine Besonderheit ist, dass alle Funktionen des PDAs/Telefons angepasst werden können. So können zum Beispiel Programme geschrieben werden, die die Anruf-Funktion eines Mobiltelefons nutzen.
- **Linux**
  Seit einiger Zeit werden PDAs mit einem Linux Betriebssystem angeboten. Die interessanteste Möglichkeit, Anwendungen für eine mobile Linux Version zu entwickeln, bietet die Oberflächenbibliothek Qtopia von Trolltech (http://www.trolltech.com/). Qtopia bietet eine reiche Auswahl an Elementen, die das Gestalten von anspruchsvollen Benutzeroberflächen ermöglicht.

3.6 Papier-Lösung

Eine weitere sehr einfache Lösung wäre, dass die Schätzer die Schäden weiterhin auf Papierformularen bewerten, jedoch in derart standardisierter Form, dass diese automatisch von einer noch zu entwickelnden Software eingelesen werden und die Daten so an die Server-Anwendung weitergereicht werden.

3.7 Plattformen für .NET

3.7.1 .NET Framework

Es handelt sich um eine Softwareplattform mit Laufzeitumgebung von Microsoft, die man Java SE gegenüberstellen kann. Der Vorteil ist eine gewisse Programmiersprachenunabhängigkeit, da es für alle in .NET unterstützten Programmiersprachen ein gemeinsames Zwischenformat gibt. Bekannte .NET Sprachen wären beispielsweise C#, VisualBasic.NET und IronPython. Der Industriepartner verwendet VisualBasic.NET, welches sich in der Syntax stark von Java unterscheidet.

Link:
http://msdn2.microsoft.com/de-de/netframework/default.aspx

3.7.2 .NET Compact Framework 1.0

Das .NET Compact Framework 1.0 wird derzeit von der aformatik für die Anwendung verwendet. Es unterstützt eine Teilmenge des .NET Frameworks, gilt als Microsofts Alternative zu Java ME und ist auf mobile Anwendungen zugeschnitten. Eine Integration in Visual Studio ist gut möglich, als Testwerkzeug dient hier Visual Studio 2003.

Link:
http://msdn2.microsoft.com/de-de/netframework/default.aspx

3.7.3 .NET Compact Framework 3.5

Dies ist die neueste Version des .NET Compact Frameworks, sie ist erst vor wenigen Monaten erschienen. Als Testwerkzeug wird Visual Studio 2008 verwendet. Die wichtigsten Unterschiede zum .NET CF 1.0 sind, dass es deutlich mehr GUI-Elemente gibt, z.B. einen Datepicker. Außerdem unterstützt es Funktionalitäten der Windows Communication Foundation (HTTP, HTTPS, ...) und soll insgesamt eine höhere Ausführungsgeschwindigkeit besitzen, was die Microbenchmark-Tests jedoch erst nachweisen müssten.

Links:
http://msdn2.microsoft.com/de-de/netframework/default.aspx
http://msdn2.microsoft.com/en-us/library/bb397835.aspx

3.7.4 andere Versionen von .NET CF

Vom .NET CF stehen natürlich noch die Versionen zwischen 1.0 und 3.5 zur Verfügung.

3.8 Plattformen für Java

3.8.1 Konfigurationen von Java ME

Bevor die einzelnen Alternativen bei Java ME aufgelistet werden, soll hier kurz vorgestellt werden, in welchen drei Konfigurationen Java ME betrieben werden kann:

- **CLDC (Connected Limited Device Configuration)**
  Dies ist die kleinstmögliche Konfiguration einer Laufzeitumgebung, sie ist jedoch nur für einfache Taschenrechner oder Ähnliches geeignet.
- **MIDP (Mobile Information Device Profile)**
  High Level API mit wenigen Widgets, Low Level API auf Pixelebene
- **CDC (Connected Device Configuration)** hat drei mögliche Profile:
  - Foundation Profile (keine Grafik)
  - Personal Basis Profile (einfache grafische Benutzerschnittstelle)
  - Personal Profile (vollständige Unterstützung von AWT)

Für die Anwendung zur mobilen Schadenserfassung kommen nur das Personal Profile und das Foundation Profile mit eigenen Erweiterungen (wie beispielsweise bei eRCP) in Betracht, alle anderen bieten von Seiten der Grafik und der GUI-Elemente eindeutig zu wenig. Deswegen werden im folgenden nur Lösungen betrachtet, die diese Ebene erreichen.

3.8.2 Java ME via IBM J9

Als Vertreter einer kommerziellen VM für PDAs wird IBMs J9 (Websphere Everyplace Micro Environment) verwendet. Der Preis variiert zwischen 7-9€. Als Testwerkzeug wird Eclipse verwendet. J9 ist eine gängige Lösung für Java ME auf PDA.

Link:
http://www-306.ibm.com/software/wireless/weme/

3.8.3 Java ME via CreMe

CreMe ist eine Virtuelle Maschine für Java (JVM) von NSIcom. Sie bietet eine spezielle Swing-Unterstützung per Update an und ist kompatibel zu den CDC-Profilen Foundation und Personal. Außerdem ist eine Bibliothek verfügbar, die spezielle Windows Mobile Features nutzbar macht, wie z.B. Spezial-Buttons an PDAs.

Link:
http://www.nsicom.com/

3.8.4 Java ME via PhoneME

PhoneME ist ein javabasiertes Projekt, das speziell für mobile Rechner mit Telefonfunktion ausgelegt ist. Nur eine „Advanced“ Version unterstützt das Personal Profile. Sie ist erst seit wenigen Wochen auf PDA verfügbar und befindet sich noch im Versuchsstadium.

Links:
https://phoneme.dev.java.net/
http://wiki.java.net/bin/view/Mobileandembedded/PhoneMEAdvancedPlatformsWindowsMobile
http://community.java.net/mobileandembedded

3.8.5 Java SE via MySaifu

Es konnte nur eine JVM gefunden werden, die Java Standard Edition (SE) auf PDA unterstützt. Es handelt sich um Mysaifu, eine Open Source JVM aus Japan für Windows Mobile. Als Testwerkzeug wird Eclipse verwendet. Oberflächenelemente sind sowohl mit Swing als auch mit AWT realisierbar.

Link:
http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html

3.9 GUI-Bibliotheken für Java

3.9.1 AWT

Beim AWT (Abstract Windows Toolkit) handelt es sich um eine Standard-Programmierschnittstelle, mit der GUI-Oberflächen für Java-Programme realisiert werden können. Standardmäßig werden hierbei die GUI-Komponenten nicht selbst gezeichnet, sondern es werden diejenigen des Betriebssystems genutzt.

Link:
http://java.sun.com/products/jdk/awt/

3.9.2 Swing

Auch bei Swing handelt es sich um eine Standard-Programmierschnittstelle, mit der GUI-Oberflächen für Java-Programme realisiert werden können. Hierbei werden jedoch die GUI-Komponenten selbst zeichnet. Diese sollten deshalb auf verschiedenen Plattformen gleich aussehen.

Link:
http://java.sun.com/docs/books/tutorial/uiswing/index.html

3.9.3 eRCP

Die embedded Rich Client Platform (eRCP) ist ein Framework, das als Konkurrent zum Personal Profile gilt. Mit ihm lassen sich zahlreiche GUI-Elemente realisieren, insbesondere in Kombination

mit dem eSWT (Standard Widget Tool). Es werden spezielle Widgets für mobile Geräte angeboten.

Link:
http://www.eclipse.org/ercp/

4 Plattformvergleich

4.1 Einleitung

Um den Plattformvergleich eine systematische Basis zu geben und um die Auswahl für detaillierte-re Untersuchungen, wie beispielsweise Benchmarktests und GUI-Analysen, einzuschränken, wurde der Vergleich in mehrere Runden aufgeteilt. Die Details werden dabei von der Wurzel des Baumes (siehe Abbildung 1: Übersicht der Lösungen) hin zu den Blättern in jeder Runde höher. Dabei werden nur diejenigen Lösungen weiter untersucht und kommen in die nächste Runde, wenn sie eine realistische Option darstellen.

4.2 Runde I – Art der Lösung

Zunächst einmal werden verschiedenen Grundkonzepte beurteilt, die eine Lösung zur mobilen Schadenserfassung darstellen.

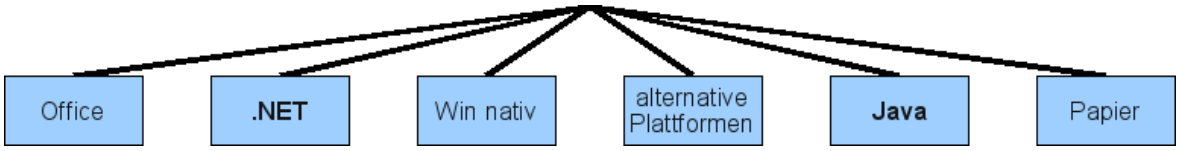


Abbildung 2: Übersicht Runde I

Lösung: Office	
Pro: ● sehr einfach ● sehr wenig bis gar kein Implementierungsaufwand ● kein Einscannen der Formulare wie bei der Papierlösung nötig	Contra: ● Schnittstelle zur Server-Anwendung unklar ● entspricht nicht dem Anspruch des Kunden ● viel zu geringer Komfort
Resultat: Die Gegenargumente wiegen so stark, dass es die Office-Lösung nicht in die nächste Runde schafft.	

Lösung: .NET	
Pro: ● Programm liegt bereits in .NET vor ● Technologie ist weitgehend programmiersprachenunabhängig	Contra: ● Aufwand für eine .NET-Fortbildung ist im konkreten Fall höher als bei Java
Resultat: Da das Programm derzeit in VB.NET CF v1.0 vorliegt, ist klar, dass die .NET-Lösung auf jeden Fall näher untersucht werden muss und somit in die nächste Runde weiterkommt.	

Lösung: Win nativ	
Pro: ● Performance wahrscheinlich etwas besser	Contra: ● Sehr hoher (Neu)-Implementierungsaufwand ● Umfangreiche Einarbeitung und evtl. Schulungen nötig ● Anwendungen sind sehr speziell auf die Plattform zugeschnitten und erschweren somit einen späteren Wechsel zu einer anderen Technologie (hohe Plattformabhängigkeit)
Resultat: Der Grund, weshalb es diese Lösung nicht in die nächste Runde schafft, ist der Mangel an Vorteilen, der dem als groß erachteten Aufwand gegenübersteht.	

Lösung: Alternative Plattformen	
Pro: ● Vorteile unklar	Contra: ● Sehr hoher (Neu)-Implementierungsaufwand ● Umfangreiche Einarbeitung und evtl. Schulungen nötig ● Die Technologien sind teilweise noch sehr neu. Android ist z.B. noch in der Entwicklungsphase und deshalb für anspruchsvolle Auftragssoftware noch nicht geeignet. ● Anwendungen sind sehr speziell auf die Plattform zugeschnitten und erschweren somit einen späteren Wechsel zu einer anderen Technologie (hohe Plattformabhängigkeit)
Resultat: Auch hier ist es sehr ähnlich wie bei der nativen Windows-Programmierung, dass durch die unbekannte Technologie ein großer Aufwand erwartet wird, wobei es keine klaren Vorteile gibt. Außerdem würde es den Rahmen dieser Fachstudie sprengen, Linux auf PDA oder auch Symbian OS genauer zu untersuchen. Deshalb kommt auch diese Lösung nicht in die nächste Runde.	

Lösung: Java	
Pro: ● Synergieeffekte mit anderen Projekten der Firma ● Java ist als gängige Lösung sehr weit verbreitet	Contra: ● Anwendung müsste von .NET nach Java portiert werden
Resultat: Es ist klar, dass Java in die nächste Runde weiterkommen muss..	

Lösung: Papier	
Pro: ● sehr einfache Lösung ● keine Performance-Probleme ● Papier wird meist gerne verwendet ● wird subjektiv oft als sicherer empfunden	Contra: ● Software, die die Papierformulare automatisch einliest, müsste noch entwickelt werden. ● Die Papierlösung entspricht nicht den Vorstellungen der Anwender. ● Es besteht ein sehr hoher Aufwand für die Verwaltung der Formulare und für das Papier.
Resultat: Eine Lösung, die durchaus in Frage kommt und einige wichtige Vorteile, jedoch auch einen hohen Aufwand mit sich bringt. Es würde jedoch in dieser Fachstudie zu weit führen, auf die Erkennungssoftware für die Formulare näher einzugehen, da diese mit Programmen auf mobilen Rechnern nichts mehr zu tun hat. Außerdem ist es fragwürdig, ob die Anwender diese Lösung akzeptieren würden. Aus diesen Gründen kommt diese Idee nicht in die nächste Runde weiter.	

Hier ist eine Übersicht, wie der Stand nach Runde 1 aussieht. Es bleiben .NET und Java zur näheren Untersuchung übrig, die anderen Alternativen werden nicht näher betrachtet.

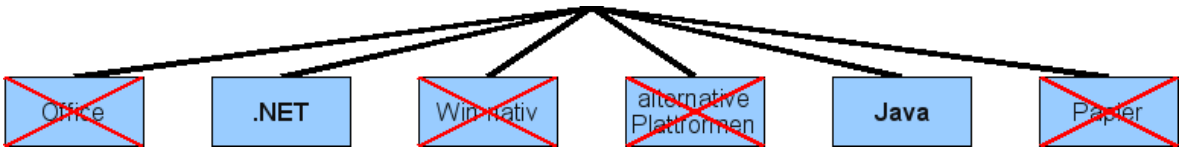


Abbildung 3: Zwischenergebnis nach Runde I

4.3 Runde II – Edition der Plattform

Sowohl .NET als auch Java gibt es in zwei Editionen, die jeweils für mobile und nicht-mobile Rechner ausgelegt sind.

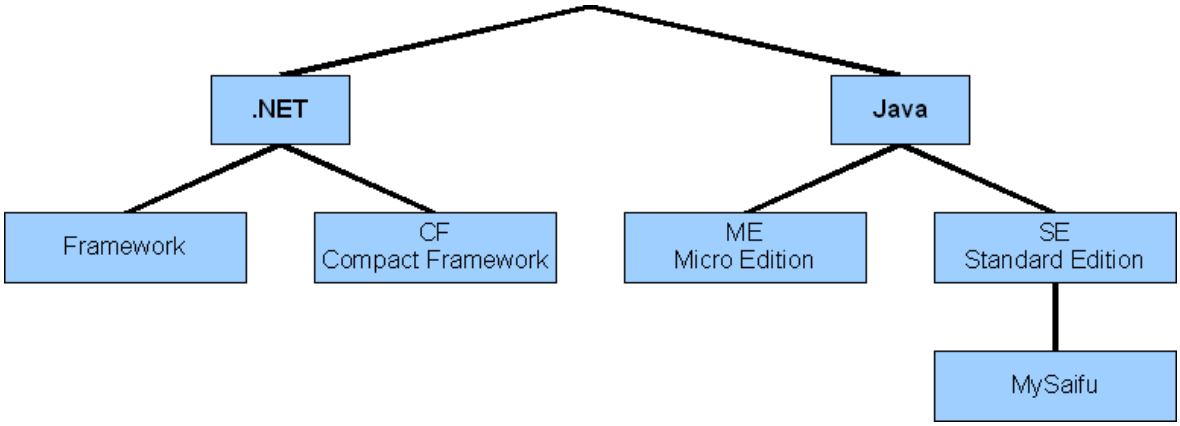


Abbildung 4: Übersicht Runde II

.NET Framework	
Pro: ● bietet mehr Funktionalität als die Compact Framework-Version	Contra: ● für mobile Rechner mit Windows Mobile 5.0 nicht anwendbar
Resultat: Es ist offensichtlich, dass diese Lösung nicht in die nächste Runde weiterkommt, da die Nicht-Kompaktversion auf den entsprechenden PDAs nicht installierbar ist.	

.NET CF (Compact Framework)	
Pro: ● Programm liegt bereits in .NET CF vor	Contra: ● CF bietet weniger Funktionalität als die Nicht-Kompaktversion
Resultat: Da das .NET Framework nicht in Frage kommt, bleibt als .NET Edition nur das CF übrig, welches in die nächste Runde kommt.	

Java SE (Standard Edition) via MySaifu	
Pro: ● bietet mehr Funktionalität als Java ME	Contra: ● Es wurde nur eine JVM gefunden, die Java SE auf PDA unterstützt, MySaifu. ● GUI-Tests haben ergeben, dass MySaifu inakzeptabel langsam ist und eine fehlerhafte Darstellung besitzt. ● MySaifu gibt es nur in einer relativ frühen Version. ● Die Weiterentwicklung des Projektes ist sehr ungewiss.
Resultat: Mangels virtueller Maschinen für Java SE, die auf PDAs laufen und wegen der schlechten Qualität der vorhandenen, kann Java SE keine akzeptable Lösung darstellen und kommt somit nicht weiter in die nächste Runde.	

Java ME (Micro Edition)	
Pro: ● Da weder Java SE (Standard Edition), noch Java EE (Enterprise Edition) hier anwendbar sind, bleibt für die Anwendung von Java nur noch die Micro Edition übrig.	Contra: ● bietet nicht die volle Funktionalität von Java SE
Resultat: Java ME schafft es in die nächste Runde.	

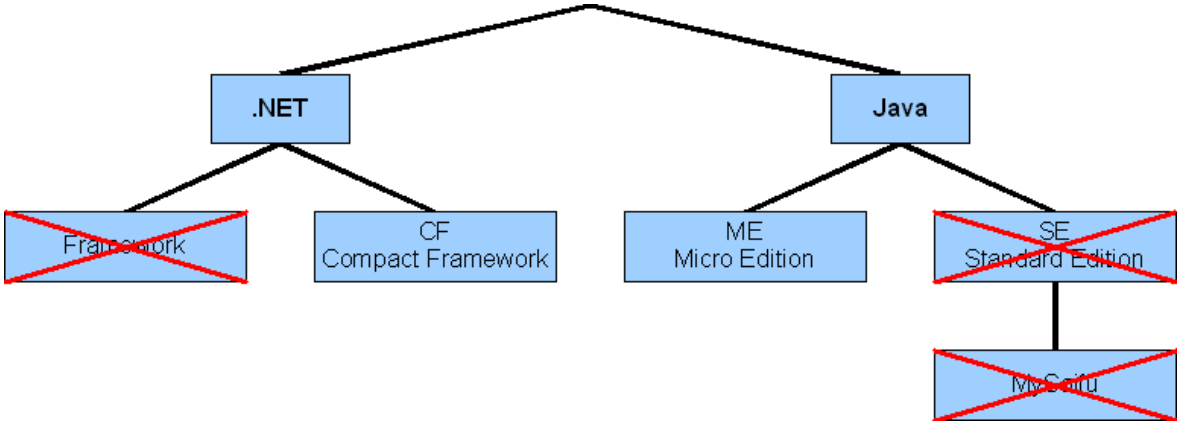


Abbildung 5: Zwischenergebnis nach Runde II

4.4 Runde III – Welche konkrete VM

Es sind nur noch die jeweiligen Kompakt-Versionen von .NET und Java übrig. Insbesondere bei .NET gibt es merkbare Unterschiede in den Versionen. Bei Java gibt es mehrere verschiedene virtuelle Maschinen für PDAs. All das wird in in dieser Runde näher angeschaut. Mit den Siegern dieser Runde werden die Microbenchmark-Untersuchungen durchgeführt. Für die GUI-Untersuchungen gibt es noch eine weitere Runde, in der mögliche GUI-Bibliotheken ausgewählt werden.

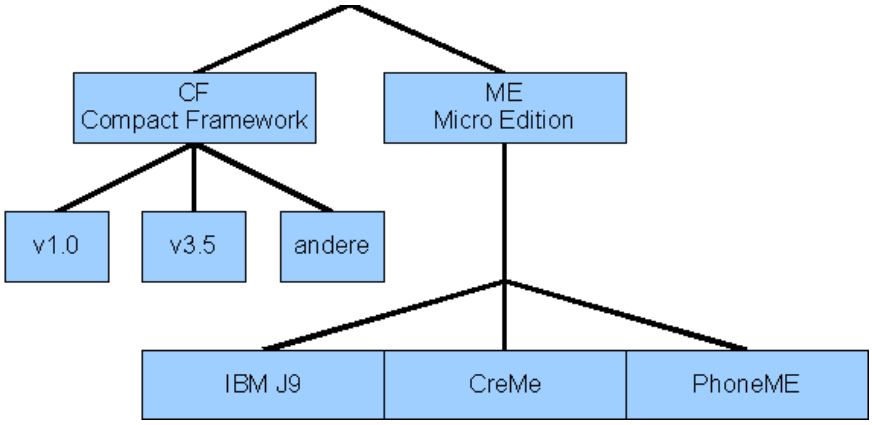


Abbildung 6: Übersicht Runde III

.NET CF v1.0	
Pro: ● Programm liegt bereits in .NET CF v1.0 vor ● integrierte Grafikbibliotheken verfügbar	Contra: ● einige GUI-Elemente nicht vorhanden
Resultat: Da dies genau die Konfiguration der jetzigen Anwendung ist, kommt .NET CF v1.0 klar in die nächste Runde.	

.NET CF v3.5	
Pro: ● Portierungsaufwand wäre nur sehr gering ● sehr neue Version ● bietet viele GUI-Elemente ● soll gute Performanz haben ● integrierte Grafikbibliotheken verfügbar	Contra: ● evtl. mit dem Programm in .NET CF v1.0 nicht abwärtskompatibel
Resultat: Mit sehr wenig Aufwand könnte man mit dieser Lösung einige Vorteile mehr haben, deshalb muss auch .NET CF v3.5 weiter in die nächste Runde kommen.	

.NET CF in anderer Version	
Pro: ● keine Vorteile erkennbar	Contra: ● weder vorhandene noch neueste Version
Resultat: Da es weder die Vorteile von v1.0 hat (keine Portierung nötig), noch die Vorteile von v3.5 (neueste Version), werden andere Versionen als diese beiden im Folgenden nicht weiter berücksichtigt.	

Java ME via IBM J9	
Pro: ● relativ gut installierbar ● weit verbreitet ● Java ME-Programme sind lauffähig	Contra: ● nicht so gute Performance
Resultat: Da die Vorteile überwiegen, wird J9 näher untersucht und darf somit in die nächste Runde weiter.	

Java ME via CreMe	
Pro: ● Plattformunabhängigkeit ● weitere Vorteile nicht erkennbar	Contra: ● schwer installierbar ● Programme kaum lauffähig
Resultat: CreMe sollte eigentlich nicht in die nächste Runde kommen, da es schwerwiegende Nachteile hat. Wir haben uns wegen der geringen Anzahl an Alternativen dennoch dafür entschieden, weitere Untersuchungen mit CreMe durchzuführen.	

Java ME via PhoneME	
Pro: ● zukunftsfähig, da immer mehr Geräte PDA- und Mobilfunk-Funktionalitäten verbinden	Contra: ● schwer installierbar ● PhoneME betont die Funktionalität von Mobiltelefonen, was hier nicht im Mittelpunkt der Anwendung steht
Resultat: PhoneME schafft es nicht in die nächste Runde, da es keine klaren Vorteile hat.	

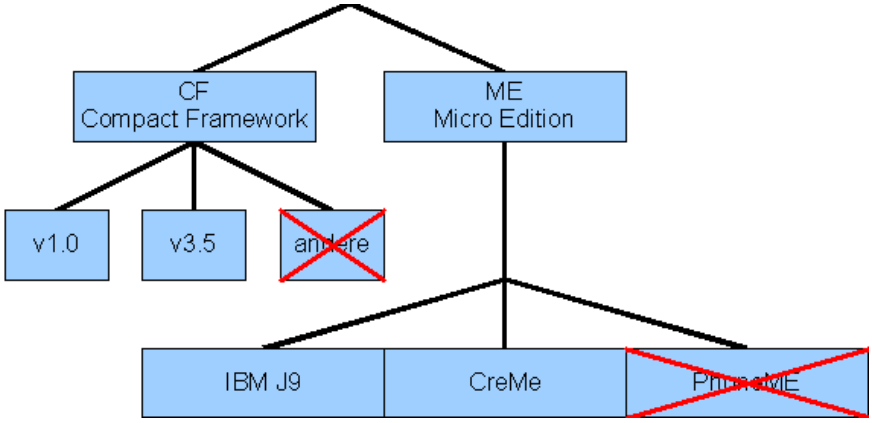


Abbildung 7: Zwischenergebnis nach Runde III

4.5 Runde IV – Kombination mit GUI-Bibliotheken

Die Java-Lösungen müssen noch mit den entsprechenden Lösungen für die GUI-Elemente kombiniert werden. Bei den .NET CF-Versionen entfällt dies, da diese jeweils ihre eigenen Bibliotheken mitbringen, somit kommen diese automatisch weiter.

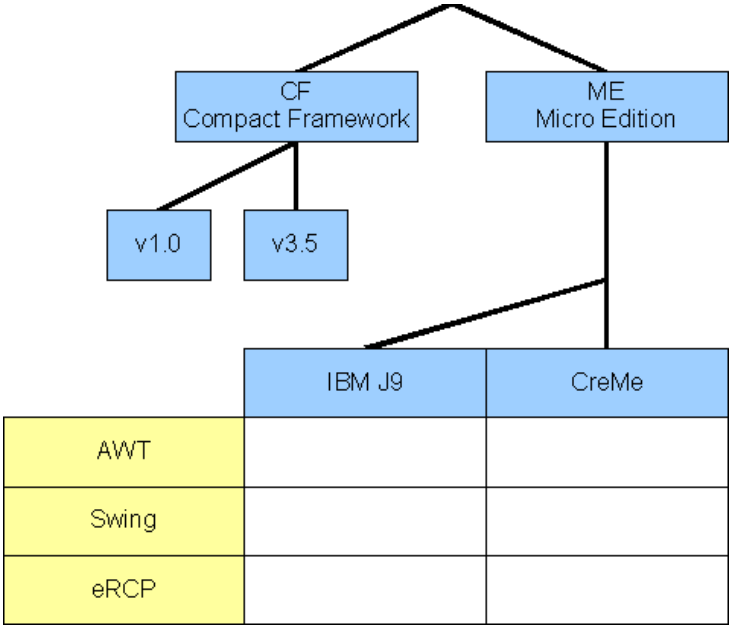


Abbildung 8: Übersicht Runde IV

Die Anforderungen der mobilen Schadenerfassung fordern mehr GUI-Elemente als es die AWT-Bibliotheken bieten können. Diese fehlenden GUI-Elemente von Hand zu programmieren, schließen wir an dieser Stelle deshalb aus, weil es zum Einen einen sehr hohen Aufwand bedeuten würde. Zum Anderen halten wir es für kein gutes Software Engineering, die AWT-Bibliothek sozusagen künstlich zu erweitern, nur um Elemente zu bekommen, die es bereits fertig in anderen Bibliotheken wie etwa Swing gibt. Aus diesem Grund schließen wir AWT von der weiteren Betrachtung aus. Für die übrig gebliebenen Bibliotheken haben wir uns für folgende Kombination entschieden: Swing testen wir mit CreMe, eRCP mit IBM J9. Diese Kombination erschien uns als die Sinnvolls-

te, da es bei eRCP mit IBM J9 bereits (gute) Erfahrungen gibt. Zum Anderen wollten wir die Swing-Bibliotheken auf jeden Fall in die weiteren Test mit einbeziehen, da diese sehr verbreitet sind. Somit ergibt sich nach der letzten Runde folgendes Bild, bei dem noch vier mögliche Technologien übrig sind:

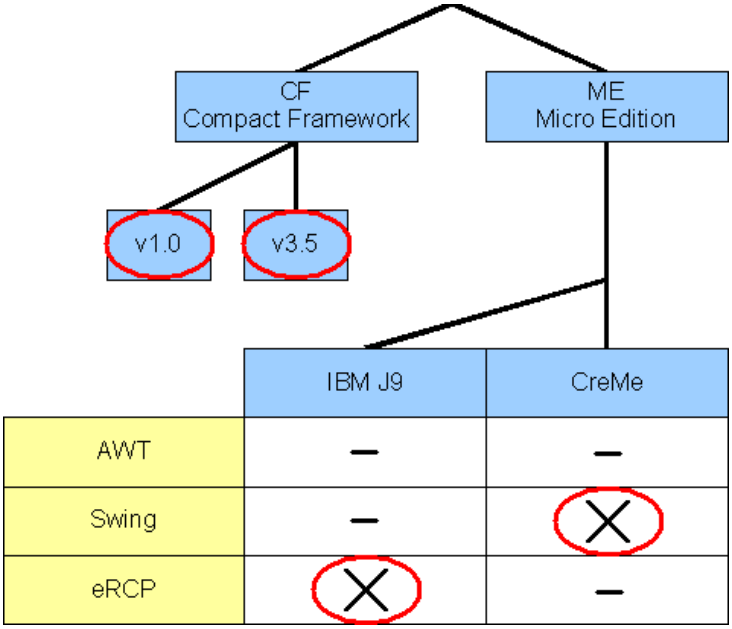


Abbildung 9: Endauswahl nach der letzten Runde

4.6 GUI Bewertung

Im Folgenden werden die oben ausgewählten Konfigurationen anhand ihrer Möglichkeiten, eine Benutzeroberfläche darzustellen, verglichen. Bei der Konfiguration CreMe mit Swing ist darauf zu achten, dass das Erscheinungsbild vom gewählten Look & Feel abhängig ist. Bei eRCP werden teilweise die nativen Elemente des Betriebssystems verwendet, deshalb besteht auch hier eine Abhängigkeit. In beiden Fällen sind die dargestellten Screenshots also nur exemplarisch. Die Bewertung ist in zwei Teile aufgeteilt. Es werden die vorhandenen Elemente der genannten Konfigurationen verglichen und verschiedene Performance-Messungen durchgeführt. Erweiterungen von Dritten wurden nicht in den Vergleich mit einbezogen. Auf absolute Vollständigkeit der Liste wurde nicht Wert gelegt, vielmehr wurde darauf geachtet, dass alle gängigen Elemente, die zum erstellen einer Benutzeroberfläche benötigt werden, vorhanden sind.

4.6.1 Bewertungsmethode

Zu jedem Element werden passende Eigenschaften aufgelistet. Für jede Konfiguration wird dann angegeben, ob sie dieses Element unterstützt. Dabei sind die Bewertungen wie folgt zu verstehen:

Ja	wird unterstützt
Nein	Eigenschaft wird nicht unterstützt
x	Element nicht vorhanden

Natürlich können mit eigenem Programmieraufwand viel mehr Elemente und Eigenschaften realisiert werden, es wurden hier jedoch nur die sofort verfügbaren Elemente betrachtet.

4.6.2 Allgemeines

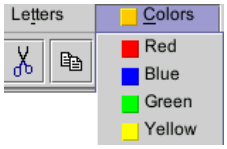
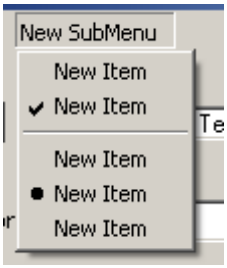
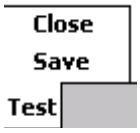
Hier werden allgemeine Funktionen aufgelistet, die für jedes Element verwendet werden können.

	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Tool Tips	Ja	Ja	Nein	Ja

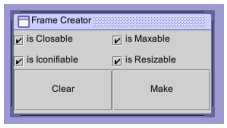
4.6.3 Container

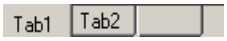
Container sind Elemente, die andere Elemente enthalten können. Sie sind sehr wichtig, um eine übersichtliche und gruppierte Darstellung zu erhalten.

Panel	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot				
Rahmenauswahl	Ja	Nein	Nein	Nein
Scrollbar	Ja (via JScrollPane)	Ja	Ja (muss selbst programmiert werden)	Ja
Splitter	Ja	Nein	Nein	Ja
Status Bar	Ja	Nein	Ja	Ja
Zeichenebene	Ja	Ja	Nein	Nein
Bild möglich	Ja	Nein	Ja (via PictureBox)	Ja (via PictureBox)
Beschriftung möglich	Ja	Nein	Nein	Nein

Menü Bar	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot				
Bilder in Liste möglich	Ja	Nein	Nein	Nein
Radio Button	Ja	Ja	Nein	Nein
Check Box	Ja	Ja	Nein	Nein
Cascade	Ja	Ja	Ja	Ja

Toolbar	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot		x	x	x
Drag & Drop	Ja	x	x	x
Bild auf Buttons möglich	Ja	x	x	x
Text auf Buttons möglich	Ja	x	x	x

Frame	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot		x	x	x
Rahmenauswahl	Nein	x	x	x
Drag & Drop	Ja	x	x	x

Tab	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot		x		
Bild möglich	Ja	x	Nein	Nein
Durchschaltbar/ Platzsparen durch Pfeile	Nein	x	Nein	Nein

4.6.4 Layoutmanager

Layoutmanager bieten eine Möglichkeit, die verwendeten Elemente systematisch anzuordnen und bei einer Größenänderung neu anzuordnen. Auf PDAs sind diese weniger wichtig, da die angezeigte Fläche immer dieselbe Größe hat. Meist wird also nur das absolute Layout benötigt.

Layout	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Absolute	Ja		Ja	Ja
Fill/Flow Layout	Ja		x	x
Grid Layout	Ja		x	x
Grid Bag Layout		x	x	x
Row Layout	x		x	x
Border Layout		x	x	x

4.6.5 Standard Elemente

Hier werden die wichtigsten Elemente aufgelistet, die fast in jeder GUI-Anwendung benötigt werden.

Button	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	One	New Button	Beenden	button1
Bild möglich	Ja	Ja	Nein	Nein
Toggle Button möglich	Ja	Ja	Nein	Nein

Check Box	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	Paint Border	Check Button	CheckBox	checkBox1
Bild möglich	Ja	Nein	Nein	Nein
Dreistufig	Nein	Nein	Ja	Ja

Radio Button	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	Default	Radio Button	RadioButton	radioButton
Bild möglich	Ja	Ja	Nein	Nein

Combo Box	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	Swing			
Bild möglich	Ja	Nein	Nein	Nein
Editierbar	Ja	Ja	Ja	Ja
Schnellauswahl durch Eingabe	Ja	Ja	Ja	Ja

Label	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	Show:	New Label	Label1	label1
Bild möglich	Ja	Ja	Nein	Nein
Hyperlink	Nein	Ja (via HyperLink Element)	Nein	Ja (via LinkLabel)
Als Caption	Nein	Ja (via captioned control)	Nein	Nein

UpDown	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	x*	x	DomainUpDown1	domainUp
Text	x	x	Ja	Ja
Zahlen	x	x	Ja	Ja

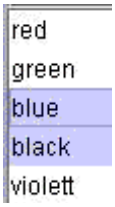
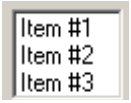
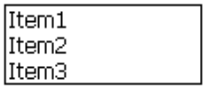
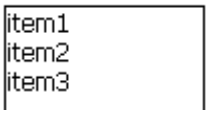
* Es gibt bei Swing zwar den JSpinner, laut NSIcom jedoch nicht in Kombination mit CreMe.

Slider	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot				
Einheiten wählbar / Anzeige	Ja	Nein	Ja	Ja
Füllstand möglich	Ja	Nein	Nein	Nein

Line Edit	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	8	Test	TextBox1	TextBox1
Hintergrund färbbar	Ja	Ja	Ja	Ja
Constraints für Eingaben	Nein	Ja	Nein	Nein
Text Extension	Nein	Ja (via Text Extension Element)	Nein	Nein

Tree View	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	Music Classical Beethoven Brahms	New Item New Item New Item	Node0 Node1 Node3	Node0 Node1
Icons möglich	Ja	Nein	Ja	Ja

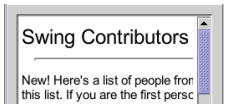
Table	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	LName / Color Albers 0, 255, 0 Andrews 255, 0, 0 Ball 0, 0, 255 Chung 0, 255, 0	New Column New	Sample Column Sample Data	categor id name 0 2 Autos 1 4 Boats 2 3 Cars and

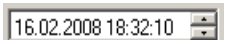
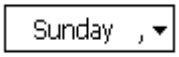
List View	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot				
Variable Darstellung (z.B. nebeneinander)	Ja	Nein	Ja	Ja
Sortierbar	Nein	Ja via SortedList Element	Nein	Nein
Icons	Ja	Ja via ListBox Element	Ja	Ja

Progress Bar	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot				

4.6.6 Spezielle Elemente

Spezielle Elemente sind Funktionen, die meist selbst programmiert werden müssten, aber schon von einigen Konfigurationen standardmäßig angeboten werden.

HTML View	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot			x	Ja

Date Picker	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	x		x	

Calendar	CreME/Swing	J9/eRCP	CF 1.0	CF 3.5
Screenshot	x	x	x	

4.6.7 Fazit

Es wurden bereits im Voraus nur Konfigurationen gewählt, die genügend Möglichkeiten bieten, eine anspruchsvolle Oberfläche zu erstellen. Wie zu sehen ist, bietet die Konfiguration mit Swing die meisten Möglichkeiten, da die Elemente von der Desktop Bibliothek übernommen wurden. Eine Anpassung an die Gegebenheiten auf einem PDA gibt es jedoch nicht. Hier bieten vor allem die Konfigurationen mit eRCP und CF 3.5 einige speziell angepasste Elemente. Das Compact Framework in der ersten Version ist nicht zu empfehlen, da es, wie sich leicht vermuten lässt, nur Nachteile gegenüber der Version 3.5 bietet. Insgesamt lässt sich feststellen, dass eine GUI mit Standardelementen am einfachsten mit dem Compact Framework zu realisieren ist, werden jedoch spezielle Elemente benötigt, bietet Swing die besten Möglichkeiten. ERCP bietet viele auf PDAs angepasste Elemente, weshalb es Swing vorgezogen werden kann, wenn eine Anwendung nur mit gängigen Elementen erstellt werden soll. Durch die Verwendung von nativen Elementen ist die Darstellung zudem flüssiger und schneller aufgebaut, als bei einer Swing-Anwendung. Auch bieten beide Versionen des Compact Frameworks eine gute Performance, die mit der einer eRCP Anwendung zu vergleichen ist.

4.7 Microbenchmark-Test

Im Folgenden werden die verschiedenen Plattformen auf ihre Performance mit Hilfe von Microbenchmarks bewertet. Leider war es trotz einigen Aufwands nicht möglich, diese Tests mit CreMe durchzuführen, da die Test-Programme sich damit als nicht lauffähig erwiesen.
Bei den Microbenchmarks werden elementare Operationen, wie z.B. Arithmetik, Funktionsaufrufe, Instanzieren von Klassen und Arrayzugriffe, gemessen. Dabei wird ein Zeitstempel t_1 vor der Operation und ein Zeitstempel t_2 nach der Operation erstellt, die Laufzeit der Operation ist dann $t_2 - t_1$. Alle hier angegebenen Zeiten haben die Einheit Millisekunden.
Die getesteten Programmiersprachen nutzen Garbage Collection, weswegen der Garbage Collector explizit aufgerufen wurde, um Garbage Collection während des Testfalls zu vermeiden.
Da das Betriebssystems des PDA Multitasking unterstützt, wurde jede Plattform 5 mal gemessen, um Messfehler durch Seiteneffekte von Hintergrundaktivitäten zu vermeiden.

4.7.1 Testfälle

Es wurden 25 Testfälle entworfen, um einen Überblick über die Performance gängiger Operationen zu erhalten.

Testfall 1	Leerer Testfall, um Overhead des Testaufbaus zu messen
Testfall 2	1.000.000 double-word integer Inkrementierungen
Testfall 3	1.000.000 for-loop iterationen
Testfall 4	1.000.000 modulo Operationen
Testfall 5	10.000 String Konkatenationen
Testfall 6	30.000 String Konkatenationen
Testfall 7	1.000.000 Instanzierungen einer leeren Klasse
Testfall 8	1.000.000 Instanzierungen einer simplen Klasse
Testfall 9	1.000.000 Funktionsaufrufe einer leeren Memberfunktion
Testfall 10	1.000.000 Funktionsaufrufe einer leeren statischen Funktion
Testfall 11	1.000.000 Funktionsaufrufe einer double word Integer Setter-Memberfunktion
Testfall 12	1.000.000 Funktionsaufrufe einer statischen double word Integer Setterfunktion
Testfall 13	1.000.000 Funktionsaufrufe einer string Setter-Memberfunktion
Testfall 14	1.000.000 Funktionsaufrufe einer statischen string Setterfunktion
Testfall 15	1.000.000 überladene Funktionsaufrufe einer double word Integer Setter-Member-funktion
Testfall 16	1.000.000 überladene Funktionsaufrufe einer statischen double word Integer Setter-funktion
Testfall 17	1.000.000 überladene Funktionsaufrufe einer string Setter-Memberfunktion
Testfall 18	1.000.000 überladene Funktionsaufrufe einer statischen string Setterfunktion
Testfall 19	1.000.000 direkte virtuelle Funktionsaufrufe einer string Getter-Memberfunktion
Testfall 20	1.000.000 indirekte virtuelle Funktionsaufrufe einer string Getter-Memberfunktion
Testfall 21	1.000.000 Arrayzugriffe auf die gleiche Position
Testfall 22	1.000.000 Arrayzugriffe auf eine per Modulo 1000 berechnete Position
Testfall 23	1.000.000 Arrayzugriffe auf eine per Modulo array.Length berechnete Position
Testfall 24	1.000 Iterationen über ein 1.000 elementiges Array mit Hilfe von for
Testfall 25	1.000 Iterationen über ein 1.000 elementiges Array mit Hilfe von for-each (nur bei Compact Framework)

Als Beispiel eines Testaufbaus sei Testfall 2 aufgeführt, der 1.000.000 32-bit Integer inkrementiert. Der Rest der Testfälle befindet sich im Anhang B.

Testfall 2	1.000.000 double-word integer Inkrementierungen

	`int var = 0;`
	`t1 = GetTimeStamp();`
	`while(var < 1.000.000)`
	`  var = var + 1;`
	`t2 = GetTimeStamp();`

4.7.2 Ergebnisse

In der folgenden Tabelle sind die gemessenen Werte aufgeführt. Die Werte jeder einzelnen Spalte sind dabei die Mittelwerte aus den 5 Läufen. Die Werte aller Messungen befinden sich im Anhang B.

	CF 1.0	CF 3.5	J9
Testfall 1	3,4	2	0,6
Testfall 2	25,4	21,2	163
Testfall 3	27,8	21,8	163
Testfall 4	313,4	204,2	592,6
Testfall 5	11945,2	1596,8	3203,6
Testfall 6	103138	14251,6	28204,6
Testfall 7	5326,2	702,6	744,4
Testfall 8	5610,6	875,6	1037,8
Testfall 9	54,2	31	458,8
Testfall 10	25,4	30,8	449,8
Testfall 11	240,4	23,6	540,6
Testfall 12	206,4	29	445
Testfall 13	254,8	40,2	592
Testfall 14	206,2	40,4	674
Testfall 15	238,8	23,2	539,6
Testfall 16	206,2	41	453,2
Testfall 17	255	39	592,4
Testfall 18	206	40	670,4
Testfall 19	322,6	132,4	525
Testfall 20	322,4	132,4	523,8
Testfall 21	83,8	38,8	274
Testfall 22	373,2	227,8	530,4
Testfall 23	404,6	241,2	557
Testfall 24	80,6	50,4	269,4
Testfall 25	103,2	71,8	

4.7.3 Fazit

Zwischen den Versionen 1.0 und 3.5 des Compact Frameworks ist ein deutlicher Performanceunterschied auszumachen. Die Version 3.5 ist in vielen Fällen 10 mal schneller als die alte Version. Da sich der Wechsel zwischen 1.0 und 3.5 einfach gestaltet, wird angeraten, auf die neue Version zu wechseln.

Bei dem Vergleich von Java und .NET ist der Gewinner .NET, in keinem der betrachteten Fälle (bis auf den Trivialtestfall 1) ist J9 schneller als CF 3.5.

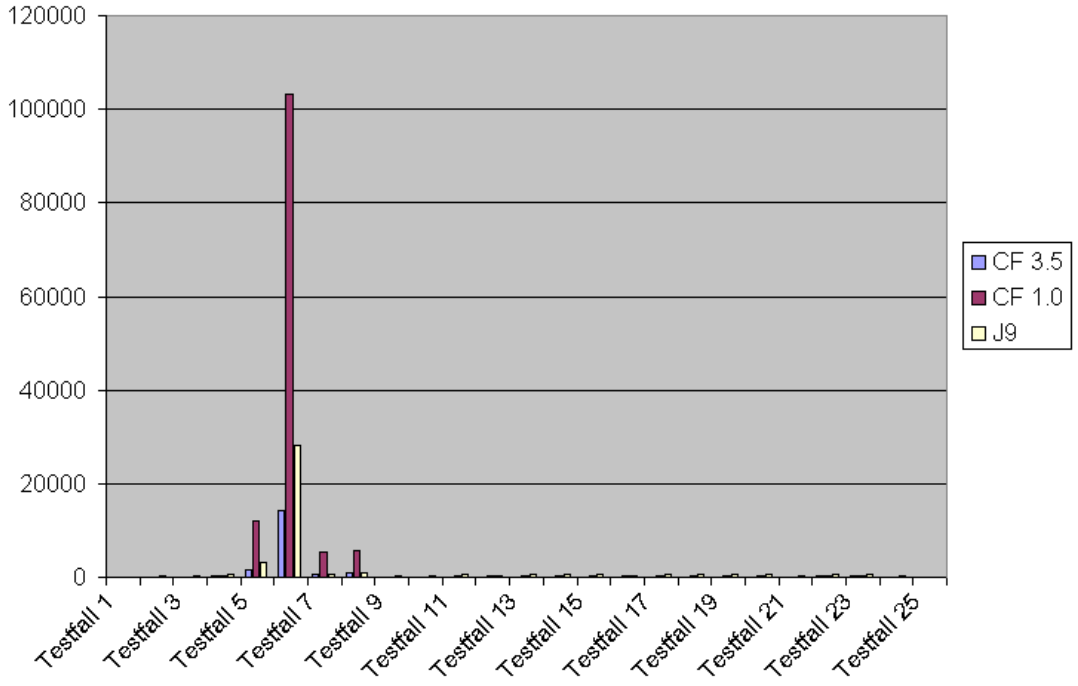


Abbildung 10: ***Ausführungszeiten*** *der Microbenchmark-Testfälle*

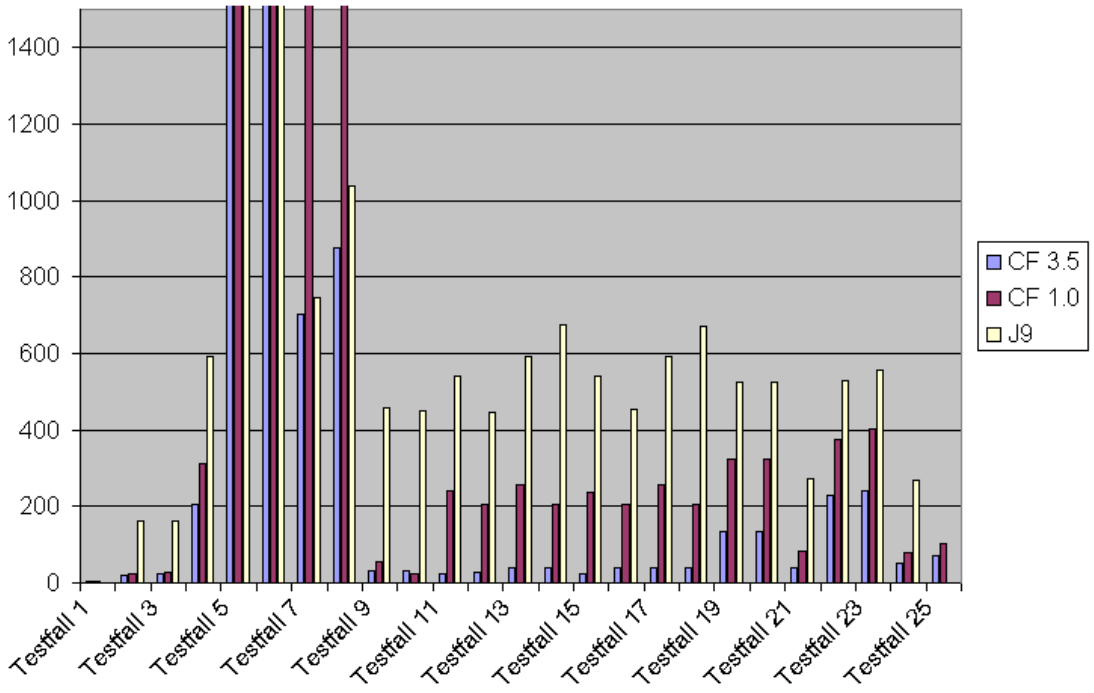


Abbildung 11: ***Ausführungszeiten*** *der Microbenchmark-Testfälle (skaliert)*

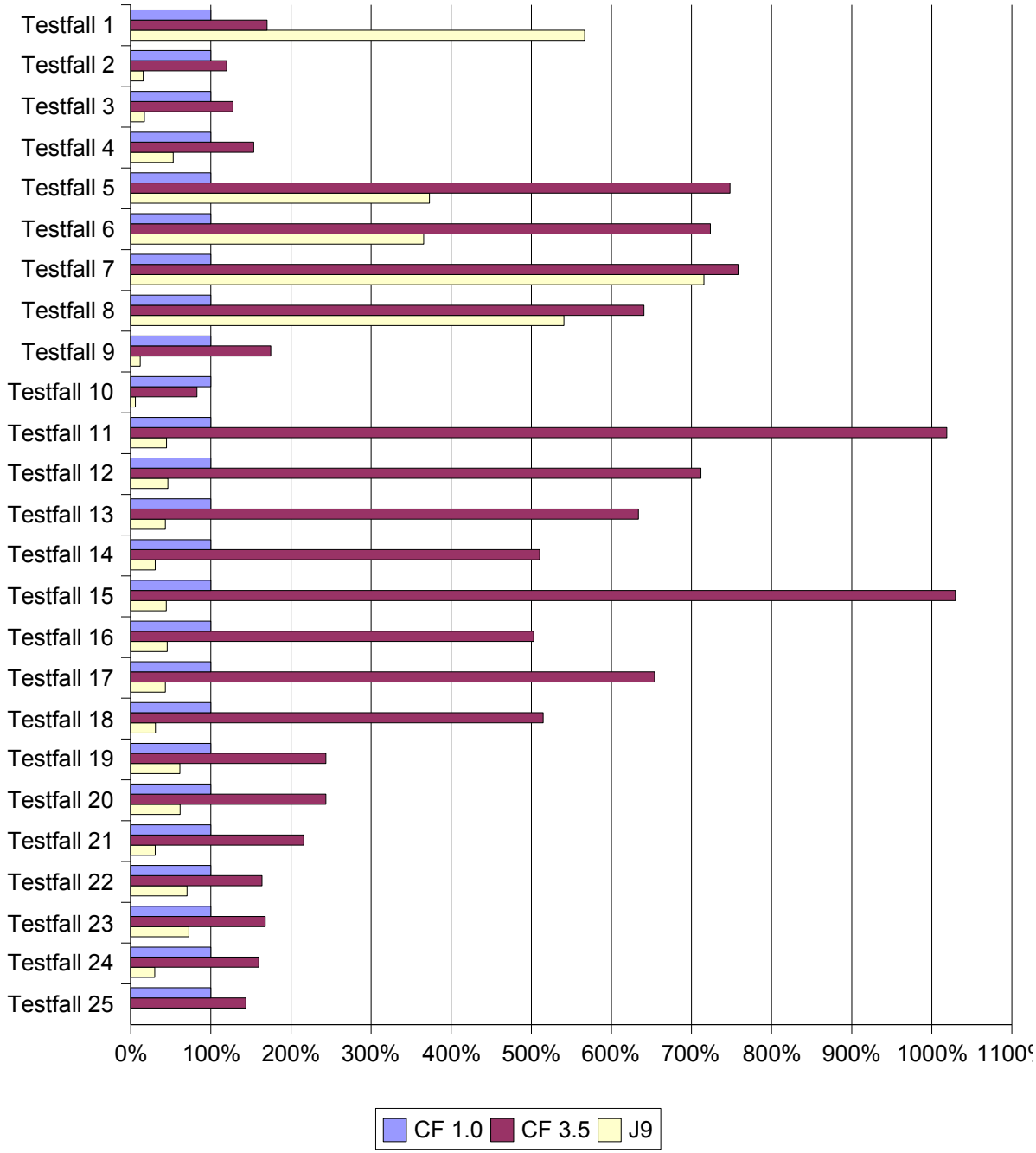


*Abbildung 12: Auf CF 1.0 genormte **Geschwindigkeiten** der Testfälle*

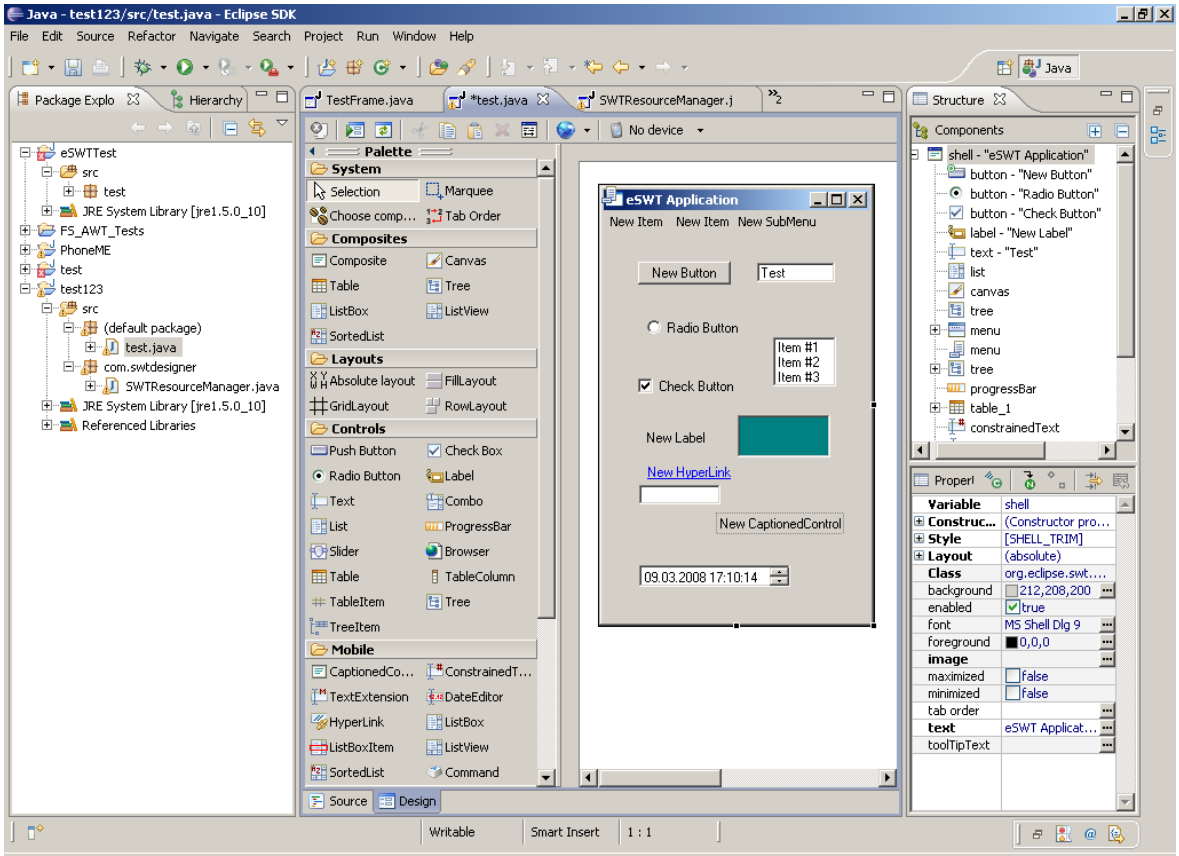
4.8 Entwicklungsumgebung

Im Folgenden wird die vom Hersteller empfohlene Entwicklungsumgebung für alle Konfigurationen bewertet.

4.8.1 J9 und eRCP

	Produkt	Kosten	Einrichtungsaufwand
Entwicklungsumgebung	Eclipse 3.2	keine	gering
GUI Designer	eRCP Designer	keine	mittel
Emulator (Debugging)	Lotus Expeditor Toolkit, Active Sync, Microsofts Pocket PC Emulator	~ 100€	hoch
Device (Debugging)	Lotus Expeditor, Active Sync	~ 100€	hoch

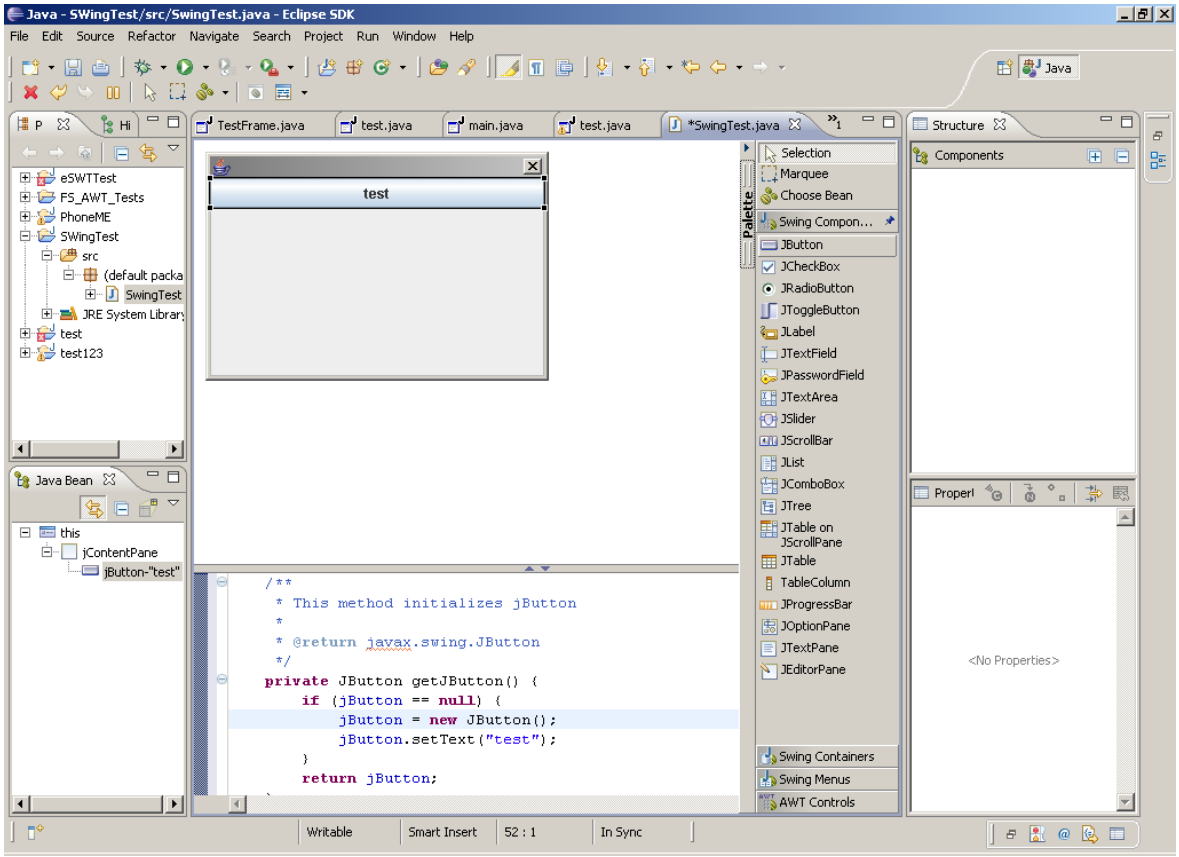
Der Einrichtungsaufwand der gesamten Entwicklungsumgebung ist hoch, da für jede Funktionalität ein extra Produkt installiert werden muss. Zudem sind die nötigen Schritte nicht immer klar, weshalb auf Hilfeseiten im Internet zurückgegriffen werden muss. Ist die Umgebung einmal eingerichtet, stehen alle nötigen Funktionen zur Verfügung.



4.8.2 CreME und Swing

	Produkt	Kosten	Einrichtungsaufwand
Entwicklungsumgebung	Eclipse 3.2	keine	gering
GUI Designer	Eclipse Visual Editor	keine	mittel
Emulator (Debugging)	Es gibt einen Emulator, jedoch wurde keine Möglichkeit zum direkten Debugging gefunden	keine	-
Device (Debugging)	Keine Möglichkeit gefunden	-	-

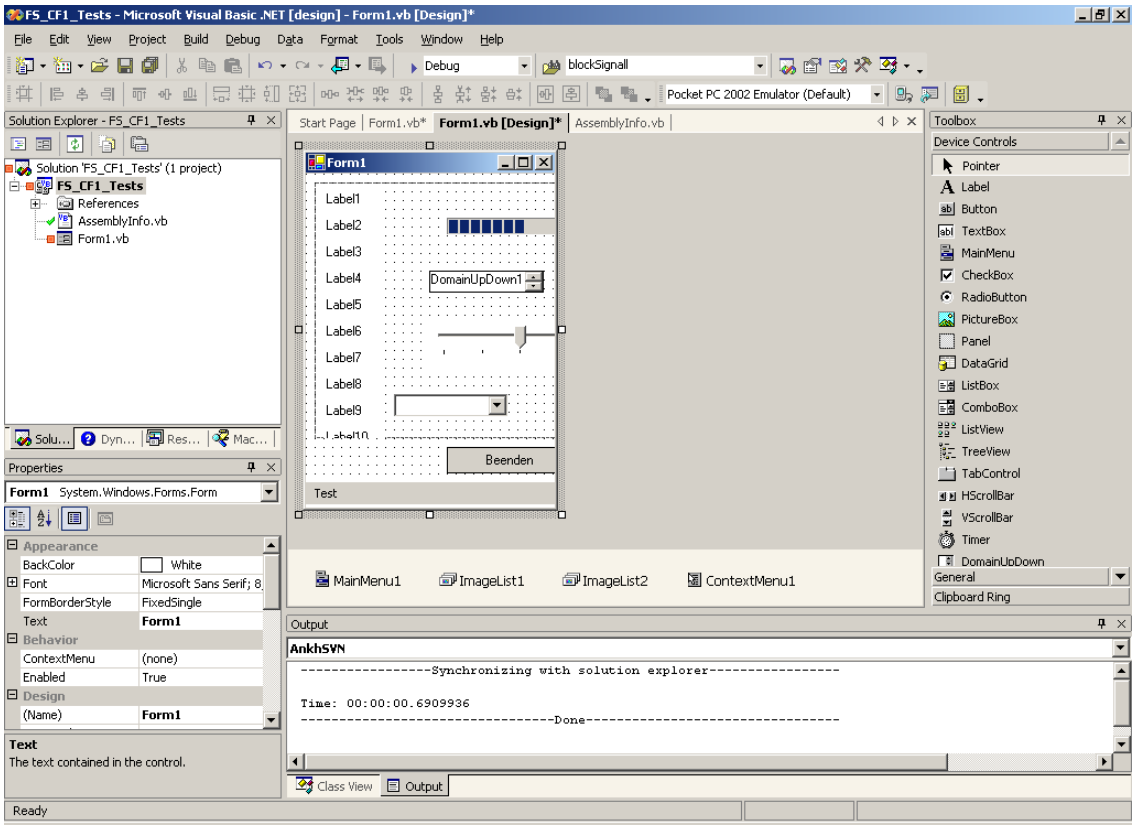
NSICom bietet keine für CreMe zugeschnittene Entwicklungsumgebung, deshalb wird auch hier auf Eclipse zurückgegriffen. Um die Swing-GUI zu erstellen kann der Visual Editor verwendet werden. Allerdings kann es zu Problemen kommen, da nur eine Untermenge von Swing unterstützt wird. Es existiert ein Emulator für CreMe, allerdings konnten wir keine Informationen dazu finden, ob er zu Debugging Zwecken an eine Entwicklungsumgebung angeschlossen werden kann. Insgesamt bleiben also einige wünschenswerte Funktionen offen. Ein weiteres schwerwiegendes Problem ist, dass es praktisch keine Fehlermeldungen von der JVM gibt und deshalb das Finden von Fehlern sehr schwierig ist.



4.8.3 .NET Compact Framework 1.0

	Produkt	Kosten	Einrichtungsaufwand
Entwicklungsumgebung	Microsoft Visual Studio 2003 (verschiedene Ausführungen)	~ 800€	gering
GUI Designer	Microsoft Visual Studio 2003 (verschiedene Ausführungen)	~ 800€	keiner
Emulator (Debugging)	Microsoft Visual Studio 2003 (verschiedene Ausführungen)	~ 800€	keiner
Device (Debugging)	Microsoft Visual Studio 2003 (verschiedene Ausführungen)	~ 800€	keiner

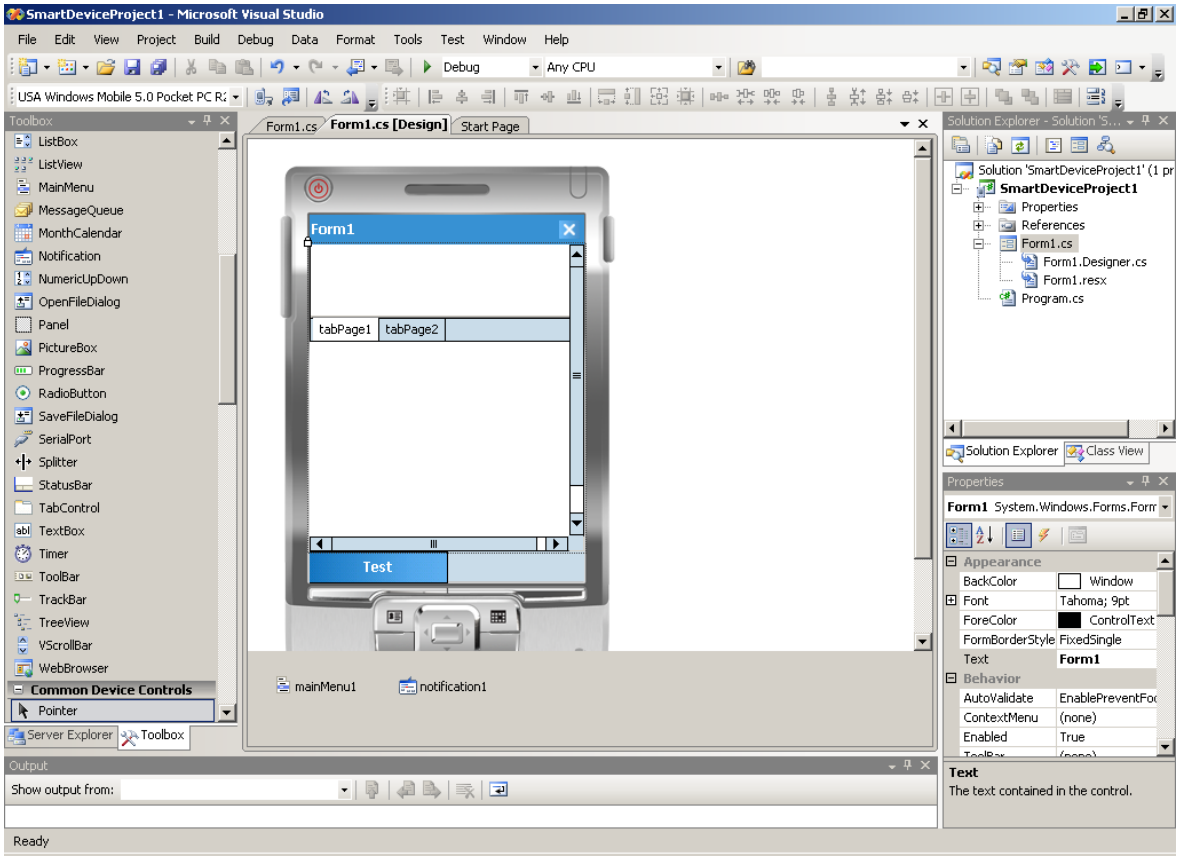
Die Entwicklungsumgebung für das Compact Framework ist von Microsoft sehr gut gemacht. Es genügt, Visual Studio mit den nötigen Komponenten für die Entwicklung für mobile Geräte zu installieren. Danach sind schon alle nötigen Einstellungen vorgenommen, um im mitgelieferten Emulator oder auf einem über Active Sync verbundenen PDA zu debuggen.
Es sind also alle nötigen Funktionen vorhanden und sehr leicht einzurichten.



4.8.4 .NET Compact Framework 3.5

	Produkt	Kosten	Einrichtungsaufwand
Entwicklungsumgebung	Microsoft Visual Studio 2008 (verschiedene Ausführungen)	~ 950€	gering
GUI Designer	Microsoft Visual Studio 2008 (verschiedene Ausführungen)	~ 950€	keiner
Emulator (Debugging)	Microsoft Visual Studio 2008 (verschiedene Ausführungen)	~ 950€	keiner
Device (Debugging)	Microsoft Visual Studio 2008 (verschiedene Ausführungen)	~ 950€	keiner

Genau wie beim Visual Studio 2003 genügt eine Installation um alle nötigen Funktionen nutzen zu können. Hinzu kommen Emulatoren für aktuelle Windows Versionen.



4.8.5 Fazit

Abschließend lässt sich sagen, dass sich die Entwicklungsumgebung für das Compact Framework viel einfacher einrichten lässt als für die verschiedenen Java Plattformen. Beim Compact Framework wird alles aus einer Hand geliefert, deshalb ergibt sich ein stimmiges Gesamtbild. Alle Funktionen sind da, wo man sie vermutet. Auch der GUI Designer ist durch die auf PDAs angepasste Formulargröße sehr einfach zu bedienen.
Eclipse hingegen kann mit vielen Plugins angepasst werden und bietet deshalb die größere Funktionenvielfalt. Leider ist es sehr schwierig, die Entwicklungsumgebung für mobile Geräte anzupassen und die nötigen Funktionen zu finden. Eine gute Möglichkeit für das Debugging ist es, die Programme mit dem PC und der dort installierten JVM zu testen, allerdings ist hier natürlich nicht garantiert, dass sich das Programm auf einem PDA genauso verhält.
Natürlich bieten Eclipse und Visual Studio alle allgemeinen Funktionen wie Versionskontrolle, Fehler Highlighting etc. Hier entscheidet der Geschmack, welche Entwicklungumgebung besser gefällt.

4.9 Inbetriebnahme

4.9.1 J9 und eRCP

Nötige Schritte:

1. J9 entpacken und auf den PDA kopieren
2. eRCP entpacken und auf den PDA kopieren
3. Programm als .jar auf den PDA kopieren
4. entweder einen Link erstellen, oder über den eRCP Starter starten

Die Installation ist recht aufwendig, da vieles separat installiert werden muss. Zudem ist es schwierig, die richtigen Versionen zu finden. Eigentlich sollte eRCP mit einer J9 Version, die nur das Foundation Profile unterstützt verwendet werden. Diese konnte allerdings nicht mehr von der IBM Homepage heruntergeladen werden, deshalb war es nötig, die Version mit Personal Profile zu verwenden und alle Links anzupassen. Auch ist der Speicherverbrauch sehr hoch, neben eRCP sollten keine anderen Programme mehr auf dem PDA ausgeführt werden

4.9.2 CreME und Swing

Nötige Schritte:

1. CreME auf dem PC installieren, an dem ein PDA angeschlossen ist
2. Swing update als .cab-Datei auf den PDA kopieren und ausführen
3. Programm als .jar file auf den PDA kopieren (jars werden automatisch mit CreME ausgeführt)

Leider funktionierte die Installation von CreME bei uns nicht wie gewollt. Die .cab-Datei konnte nicht automatisch auf den PDA kopiert werden. Also musste die Datei von Hand auf den PDA kopiert und dort ausgeführt werden.

4.9.3 .NET Compact Framework 1.0 und 3.5

Nötige Schritte:

1. Installationsprogramm auf dem PC, an dem der PDA angeschlossen ist, ausführen
2. Programm auf den PDA kopieren und ausführen

Wie bei CreME wird auch hier ein Installationsprogramm auf dem PC gestartet. Allerdings funktionierte dieses problemlos.

4.9.4 Fazit

Abschließend lässt sich sagen, dass die Inbetriebnahme am einfachsten mit den beiden Versionen des Compact Frameworks gelingt. Es werden die wenigsten Schritte benötigt und Fehlermöglichkeiten sind durch die automatische Installation weitgehend ausgeschlossen. Durch die fehlgeschlagene Installation von CreME war einiger Aufwand nötig, um die für den PDA passenden .cab-Dateien für die Einrichtung zu finden. Noch schwieriger ist die Installation von eRCP, es muss eine zur eRCP Version und zum PDA passende J9 Version gefunden werden. Außerdem muss genau auf den Speicherverbrauch geachtet werden, da eRCP zusammen mit J9 schon den Großteil des Speichers in einem PDA belegt. Dies ist für einen Laien nicht zumutbar. Das Compact Framework könnte hingegen ohne weiteres vom Anwender selbst installiert werden.

5 Portierung

5.1 Vorgehensweisen

Unter Portierung verstehen wir hier die Umwandlung eines in .NET CF vorliegenden Programmes nach Java ME. Prinzipiell gibt es folgende Möglichkeiten, diese Portierung vorzunehmen:

direkt manuell	
Der Java-Code wird von Hand neu geschrieben, wobei als Vorlage dazu der Code in .NET CF (z.B. VB.NET CF) dient.	
Pro: ● sehr einfaches Konzept ● Fehler im alten Code fallen beim Umschreiben eher auf	Contra: ● 1:1 Übersetzung ist durch die Unterschiedlichkeit der Programmiersprachen oft nicht möglich ● schlechte Strukturen im alten Code werden übernommen ● hoher Aufwand

indirekt manuell	
Auch hier wird der Java Code von Hand neu geschrieben, jedoch dient hier als Vorlage nicht der alte Code, sondern eine entsprechende Architektur-Beschreibung der Strukturen mit ihren Schnittstellen.	
Pro: ● programmiersprachenspezifische Struktur des Codes wird nicht mit übernommen ● Fehler im alten Code werden nicht übernommen	Contra: ● eventuell muss eine Architekturbeschreibung erst angefertigt werden ● hoher Aufwand, der einer kompletten Neuimplementierung gleicht

automatisch	
Hierbei übersetzt ein Programm den Code automatisch von .NET CF nach Java ME.	
Pro: ● geringer Aufwand	Contra: ● Nachbearbeitung des Ergebnisses ist auf jeden Fall nötig. ● es gibt wenig Programme, die ein direktes Umwandeln nach Java unterstützen, viele wandeln nach C# oder J# ● programmiersprachenspezifische Struktur passt eventuell nicht zu umgewandelter Sprache ● manchmal werden Code-Kommentare nicht mitübersetzt

Da der Aufwand bei einer manuellen Portierung als sehr hoch eingeschätzt wird, empfehlen wir, falls überhaupt eine Portierung stattfinden soll, die automatische Alternative. Hier ist eine Nachbearbeitung des Codes zwar auch nötig, diese ist jedoch deutlich weniger aufwändig.

5.2 Automatische Portierung

Im Folgenden werden vier spezielle Möglichkeiten vorgestellt und bewertet, eine automatische Portierung zu realisieren.

CodeTranslator von CarlosAg	
Link: http://www.carlosag.net/Tools/CodeTranslator/Default.aspx	
Pro: ● kostenlos	Contra: ● nur als Online-Version verfügbar ● Übersetzung nur von VB.NET nach C# (und umgekehrt) ● hoher Nachbearbeitungsaufwand für Java ME
Fazit: Für einzelne Dateien brauchbar, für größere Projekte jedoch nicht zu empfehlen.	

.NET Translator von Octopus	
Link: http://www.remotesoft.com/octopus/	
Pro: ● Testversion online, Vollversion offline	Contra: ● mehrere Übersetzungkombinationen, jedoch nicht VB.NET nach Java ● hoher Nachbearbeitungsaufwand für Java ME ● noch in Beta-Stadium (Stand 03/2008) ● Preis auf Anfrage
Fazit: Noch nicht ausgereift und deshalb für größere Projekte nicht zu empfehlen.	

VBeeJ von BeeLucid	
Link: http://www.beelucid.com/	
Pro: ● kostenlose Offline Testversion verfügbar ● sehr einfache Installation ● sehr einfache Bedienung ● ausführliche Fehlermeldungen ● gutes Ergebnis ● Kommentare werden auch umgewandelt ● relativ geringer Nachbearbeitungsaufwand	Contra: ● Vollversion kostet 299 Dollar
Fazit: Überzeugendes Programm. Auf jeden Fall empfehlenswert.	

NET2Java	
https://net2java.dev.java.net/GettingStarted.html	
Pro: ● kostenlose Opensource-Software als Plugin von NetBeans ● kann VB.NET oder C#.NET nach Java portieren ● ausführliche Fehlermeldungen ● sehr gutes Ergebnis ● Kommentare werden auch umgewandelt ● minimaler Nachbearbeitungsaufwand ● ganze Projekte per Mausklick auf einmal umwandelbar ● weitere Mechanismen, wie z.B. das automatische Einfügen von TODO-Kommentare im Java-Code bei nicht umwandelbaren Elementen ● relativ schnelles Umwandeln innerhalb weniger Minuten, selbst wenn das Projekt sehr groß ist	Contra: ● NetBeans und mindestens JDK 1.5 müssen installiert werden (beides ist allerdings kostenlos) ● NET2Java muss erst eingerichtet werden (in 15 Minuten problemlos möglich) ● ganz ohne Nachbearbeitung des Codes geht es nicht
Fazit: Das Programm überzeugt auf ganzer Linie. Lediglich einige Dinge, die das Programm nicht wissen kann, wie etwa abweichende GUI-Bibliotheken zwischen VB.NET und Java können nicht automatisch umgewandelt werden, sondern werden automatisch mit TODO-Kommentaren versehen.	

5.3 Fazit

Der CodeTranslator von CarlosAg und der .NET Translator von Octopus sind nicht dazu geeignet, ganze Projekte sinnvoll von VB.NET nach Java zu portieren.
Das kommerzielle Programm VBeeJ von BeeLucid wandelt gut um und hat eine gute Handhabung. Installation und Bedienung sind sehr einfach machbar.
An NET2Java kommt jedoch keines der vorangehenden Programme heran. Auch wenn hier der Einrichtungsaufwand (mit NetBeans) etwas höher ist, so überzeugt die darüberhinaus auch noch kostenlose Opensource-Software NET2Java in allen Punkten und geht hier deutlich als Testsieger hervor. Natürlich geht es auch bei NET2Java nicht ganz ohne eine Nachbearbeitung des Codes, diese wird jedoch auf ein Minimum beschränkt.

6 Untersuchungsergebnisse

6.1 Empfohlene Plattform

Die abschließende Frage, welche der untersuchten Plattformen nun als die beste eingestuft werden kann, wird hier beantwortet. Pro untersuchtem Aspekt werden jeweils zwei Punkte an .NET und/oder Java verteilt, nur bei der Literaturrecherche und der Inbetriebnahme wird wegen der geringeren Relevanz nur einer vergeben.

Nach der **Literaturrecherche** ergab sich nach der Auswertung der Quellen eine Tendenz in Richtung .NET CF, besonders in Hinblick auf die Performance und die WebService-Integration. Auch kann man bei .NET per P/Invoke Teile die hochperformant laufen müssen, in C/C++ auslagern, was bei Java ME nicht möglich ist, da das JNI bei Java ME nicht vorhanden ist. Zwischenstand:

.NET 1:0 Java

In der Untersuchung der Möglichkeiten, eine **GUI** zu realisieren, haben sich nach der Auswahl der verschiedenen Konfigurationen nur wenige Alternativen ergeben. Es wurden nur die Lösungen genommen, die genügend Möglichkeiten bieten, eine anspruchsvolle Oberfläche zu erstellen.
Hier fällt das Ergebnis relativ ausgeglichen aus. Wie zu sehen ist, bietet die Konfiguration mit Swing die meisten Möglichkeiten, da die Elemente von der Desktop Bibliothek übernommen wurden. Außerdem bietet Swing die besten Möglichkeiten, wenn spezielle Elemente benötigt werden. Somit erhält Java einen Punkt.
Insgesamt lässt sich jedoch feststellen, dass eine GUI mit Standardelementen am einfachsten mit dem Compact Framework zu realisieren ist. Auch bieten sowohl CF 1.0, als auch CF 3.5 eine gute Performance, die mit der einer eRCP Anwendung zu vergleichen ist. Also erhält auch .NET hier einen Punkt.

.NET 2:1 Java

Beim **Microbenchmark-Test**, also der Untersuchung der elementaren Rechen- und Speicheroperationen der verschiedenen Plattformen ergab sich zwischen der Version 1.0 und 3.5 des Compact Frameworks ein deutlicher Performanceunterschied. Die Version 3.5 ist in vielen Fällen 10 mal schneller als die alte Version. Da sich der Wechsel zwischen 1.0 und 3.5 einfach gestaltet, wird angeraten, auf die neue Version zu wechseln. Bei dem Vergleich von Java und .NET ist der Gewinner .NET, in keinem der betrachteten Fälle (bis auf einen Trivialtestfall) ist J9 schneller als CF 3.5. Durch dieses eindeutige Ergebnis bekommt .NET hier beide Punkte. Der Zwischenstand:

.NET 4:1 Java

Es wird eng für Java. Vielleicht können die **Entwicklungsumgebungen** für Java punkten. Man merkt jedoch schnell, dass sich die Entwicklungsumgebung für das Compact Framework, Visual Studio, viel einfacher einrichten lässt als die für die verschiedenen Java Plattformen. Beim Compact Framework wird alles aus einer Hand geliefert und ergibt deshalb ein stimmiges Gesamtbild. Alle Funktionen sind da, wo man sie vermutet. Auch der GUI Designer ist durch die auf PDAs angepasste Formulargröße sehr einfach zu bedienen.
Auf Seiten von Java hingegen sieht es so aus, dass Eclipse mit vielen Plugins angepasst werden kann und deshalb die größere Funktionenvielfalt bietet. Leider ist es sehr schwierig, die Entwicklungsumgebung für Mobile Geräte anzupassen und die nötigen Funktionen zu finden. Es ist jedoch nicht ausgeschlossen, dass ein sehr erfahrender Eclipse-Benutzer sich zügig zurechtfindet. Hier entscheidet auch der Geschmack, welche Entwicklungsumgebung besser gefällt. Deswegen erhalten hier beide je einen Punkt, auch wenn wir die .NET-Lösung für gelungener halten. Der Zwischenstand:

.NET 5:2 Java

Die **Inbetriebnahme** einer Anwendung gelingt am einfachsten mit den beiden Versionen des Compact Frameworks. Es werden die wenigsten Schritte benötigt. Auf Seiten von Java war durch die fehlgeschlagene Installation von CreME einiger Aufwand nötig, um die für den PDA passenden .cab-Dateien für die Einrichtung zu finden. Noch schwieriger ist die Installation von eRCP, es muss eine zur eRCP Version und zum PDA passende J9 Version gefunden werden. Außerdem muss genau auf den Speicherverbrauch geachtet werden, da eRCP zusammen mit J9 schon den Großteil des Speichers in einem PDA belegt. Dies ist für einen Laien nicht zumutbar. Das Compact Framework könnte hingegen ohne weiteres vom Anwender selbst installiert werden. Dieser Punkt geht ganz eindeutig nach .NET. Der Endstand:

.NET 6:2 Java

Wir empfehlen somit die Verwendung von **.NET CF 3.5 mit Visual Studio**. Der Entwicklungsstand der Java-Plattform für mobile Endgeräte ist zwar ausreichend, um komplexe Applikationen zu entwickeln, aber der dazu nötige Einrichtungsaufwand ist deutlich höher. Dazu kommt die schlechtere Performance und die Kosten der Laufzeitumgebung pro PDA bei den Java-Lösungen. Ansonsten besitzt eRCP jedoch viel Potential, eine gute Alternative zu werden.

6.2 Empfehlung für den Industriepartner

Unabhängig von diesem Resultat kann es im Einzelfall dennoch sinnvoll sein, eine Neuimplementierung in Java vorzunehmen. In diesem Fall dürfen aber, wie bereits erläutert, keine Performance- oder Komfort-Verbesserungen erwartet werden, es sei denn man nutzt den Wechsel nach Java zugleich für eine komplette Neuimplementierung der Applikation. Sollte sich der Industriepartner dazu entscheiden, nach Java zu wechseln, so empfehlen wir folgendes Vorgehen:
Die **Portierung** von VB.NET nach Java von wichtigen Teilen (z.B. Backend-Klassen) oder auch vom ganzen Code kann automatisch erfolgen, um mit wenig Aufwand ein Grundgerüst der Applikation zu erhalten. Die kostenlose Opensource-Software **NET2Java** ist der eindeutige Sieger bei den Programmen zur automatischen Portierung. Auch wenn hier der Einrichtungsaufwand (mit NetBeans) etwas höher ist, so überzeugt NET2Java in allen Punkten. Natürlich geht es auch bei NET2Java nicht ganz ohne eine Nachbearbeitung des Codes, diese wird jedoch auf ein Minimum beschränkt.
Sollte es dann zu Performance-Problemen kommen, gibt es noch die relativ simple Methode, einen PDA mit schnellerem Prozessor zu kaufen. Kurztests haben gezeigt, dass die Unterschiede dazu zwischen einem 300MHz-PDA und einem 600Mhz-PDA sehr groß sind, wobei sich die Preise für den schnelleren PDA in Grenzen halten.

7 Rückblick

Das Fachstudien-Projekt lief insgesamt gut. Es gab auch keine größeren Abweichungen zur Planung am Anfang. Während der Fachstudie hat es sich jedoch als sinnvoll erwiesen, die grobe Inhaltsstruktur des Finaldokuments schon sehr früh zu erstellen. Dadurch hatten wir so etwas wie einen roten Faden, der uns nach der relativ langen Analysephase half, alle Inhalte strukturiert anzugehen. Etwas schwieriger als erwartet war es, die verschiedenen Technologien auf dem PDA zum Laufen zu bekommen.
Die Planung und die Voranalyse liefen somit problemlos, lediglich die Literaturrecherche hat weniger gebracht als wir uns erhofft hatten. Auch das hat uns gezeigt, dass es noch großes Entwicklungspotential im Bereich der mobilen Rechner gibt. Die darauf folgende Gegenstandsanalyse hat uns geholfen, klarer zu sehen, was eigentlich genau untersucht werden soll. Sie ist weiterzuempfehlen. Im Nachhinein gesehen wäre es aber wahrscheinlich effizienter gewesen, an dieser Stelle bereits mit dem Projektbericht der Fachstudie zu beginnen, anstatt viele einzelne Dokumente im Lauf der Fachstudie zu erstellen.
Der Zwischenvortrag war sehr wichtig für uns, da wir dort den Zwischenstand unserer Fachstudie präsentieren und darüber diskutieren konnten. Es stellte sich heraus, dass wir zur besseren Nachvollziehbarkeit der in der Fachstudie untersuchten Inhalte das zu behandelnde Szenario deutlicher hervorheben sollten. Damit könnten Leser und Zuhörer der Fachstudie der Thematik besser folgen.
Die Zusammenarbeit mit dem Industriepartner war positiv, wir konnten in mehreren gemeinsamen Treffen vom Anfang bis zum Ende alles klären was es zu klären galt. Jederzeit konnten wir uns mit Nachfragen an unsere Ansprechpartner vor Ort wenden. Zwei wesentliche Punkte waren es, dass es uns sowohl ermöglicht wurde, einen PDA leihweise für die Tests zur Verfügung zu bekommen, als auch Einsicht in den Code der Anwendung zu bekommen. Ohne diese zwei Punkte wäre eine detaillierte Untersuchung nicht möglich gewesen.
Die allgemeine Untersuchung erfolgte ohne Berücksichtigung des Codes der Anwendung. Dagegen war die speziellen Untersuchung auf den Code der Anwendung zugeschnitten. Diese Trennung erleichtert es, diese Fachstudie auch für ähnliche Anwendungen im Bereich der mobilen Rechner zu verwenden. Da jedoch in diesem Bereich die Geschwindigkeit mit der sich Neuerungen ergeben sehr hoch ist, ist es fraglich, ob die in dieser Fachstudie gemachten Aussagen eine lange Gültigkeit besitzen.
Der Umfang des Projektberichtes ist mit 99 Seiten letztenendes größer geworden als ursprünglich angedacht. Der jetzt noch folgende Abschlussvortrag wird die Fachstudie abrunden. Es wird nicht nötig sein, den am Anfang geplanten Puffer in der Zeitplanung zu nutzen.
Die wöchentlichen Treffen mit unserem Betreuer waren sehr gut. Zum Einen bekamen wir immer wertvolle Verbesserungsvorschläge und Hinweise. Ein herzliches Dank dafür. Zum Anderen dienten die Treffen auch zu unseren eigenen Kontrolle aller Zwischenergebnisse, insbesondere der Meilensteine.
Wir hoffen, dass die Ergebnisse dieser Fachstudie dazu beitragen, dass der Industriepartner die beste Wahl in dieser Vielzahl von möglichen Alternativen treffen wird und wünschen ihm viel Erfolg dabei.

8 Rahmenbedingungen und Planung der Fachstudie

8.1 Beteiligte Personen

Bearbeiter	Tobias Arnold, Ivan Bogicevic, Nico Gau
Prüfer	Prof. Dr. Jochen Ludewig
Betreuer	Markus Knauß
Industriepartner	Andreas Itzelberger und Stefanie Schaible (aformatik GmbH & Co. KG)

8.2 Vertragliche Anforderungen an die Projektdurchführung

Der Vertrag der Fachstudie ist eine Vereinbarung zwischen der Fakultät 5 der Universität Stuttgart und den Bearbeitern der Fachstudie. Thema und Aufgabenstellung wurden vereinbart und sind bindend. Bei einer Unterbrechung oder Verlängerung muss ein Antrag nach Maßgabe der entsprechenden Prüfungsordnung gestellt werden.
Die „Durchführungsrichtlinien für Studenten der Informatik an der Universität Stuttgart“ und die Formvorschriften der Fakultät 5 wurden beachtet.
Die Bearbeiter behalten das Urheberrecht an der Arbeit. Sind daran andere Personen maßgeblich beteiligt, so steht diesen ein entsprechender Anteil des Urheberrechts zu.
Die Bearbeiter übertragen der Universität Stuttgart ein zeitlich unbefristetes, nicht ausschließliches, nicht übertragbares, unwiderrufliches Nutzungsrecht an den Ergebnissen der Arbeit. Die Arbeit unterliegt der Sperrfrist bis 31.12.2012.

8.3 Lieferumfang

Das Projekt resultiert in diesem Projektbericht. Sein Soll-Umfang beträgt etwa 50 Seiten. Eine Kopie des Projektberichts wird in Papierform und im pdf-Format an den Industriepartner ausgehändigt. Der Prüfer erhält eine offizielles Exemplar in Papierform. Eine Kopie davon wird in der Informatik-Bibliothek der Universität Stuttgart verwahrt.
Alle weiteren Dokumente, die zusätzlich zum Projektbericht entstehen, sind nicht Teil des Lieferumfangs.
Dem Industriepartner, und bei Interesse auch dem Prüfer und dem Betreuer, werden zusätzlich der Sourcecode von entstandenen Benchmark- und Testprogrammen zur Verfügung gestellt.

8.4 Leistungen der Auftraggeber

8.4.1 Industriepartner

Einige wesentliche Fragestellungen der Fachstudie konnten nur mit Hilfe eines mobilen Rechners (PDAs) mit Windows Mobile beantwortet werden. Hierfür wurde uns freundlicherweise ein solcher von der aformatik GmbH & Co. KG für mehrere Monate zu Testzwecken zur Verfügung gestellt.

8.4.2 Universität

Das Institut für Software Engineering (ISTE) der Universität Stuttgart ermöglichte uns einen Zugang zu den Arbeitsräumen des Instituts.

8.5 Richtlinien für die Fachstudie

8.5.1 Konfigurationsmanagement

Zur Verwaltung der entstehenden Dokumente wurde ein SVN Repository verwendet. Das Repository wurde von den Projektteilnehmern privat verwaltet, es waren keine Anstrengungen seitens der Universität oder des Industriepartners nötig.

8.5.2 Design-Richtlinien

Alle Dokumente, die während der Fachstudie entstanden, hatten ein einheitliches Design. Insbesondere enthält jedes Dokument eine Versionshistorie.

8.6 Entwicklungsprozess

Die Fachstudie wird nach dem Phasenmodell durchgeführt. Dabei ist das Projekt in fünf einzelne, streng sequenziell ausgeführte Phasen eingeteilt. Jede Phase besteht aus ein oder mehreren Arbeitspaketen. Zu jedem Arbeitspaket wird ein Dokument erstellt, welches die jeweiligen Resultate festhält.
Am Ende jeder Phase gibt es einen Meilenstein, der das Ziel der jeweiligen Phase definiert. Dieses ist erreicht, wenn alle Arbeitspakete abgeschlossen sind. Das beinhaltet die Fertigstellung der angegebenen Dokumente mit einer abschließenden Prüfung durch die Projektbearbeiter.
Während des Projekts fanden mehrere Treffen mit dem Industriepartner statt.

8.7 Externe Meilensteine

Folgende externe Meilensteine sind für die aformatik und die Universität Stuttgart relevant:

- Projektbeginn ist der 22.10.2007
- In der Woche vom 26.11.2007 bis zum 01.12.2007 findet eine Analyse bei der aformatik statt (weitere Analysetermine können folgen)
- In der Woche vom 14.01.2008 bis zum 19.01.2008 findet eine Zwischenpräsentation statt
- Im April 2008 findet ein Abschlussvortrag statt
- Das Projekt endet am 21.04.2008

8.8 Phasen der Entwicklung

Die einzelnen Phasen der Entwicklung sind zusammen mit den Arbeitspaketen und den dazugehörigen Meilensteinen befinden sich in Anhang C.

8.9 Dokumentationsplan

8.9.1 Produktdokumente

Das einzige zum Gesamtprodukt gehörende Dokument ist der Projektbericht am Ende, da in ihm alle anderen relevanten Dokumente der Fachstudie enthalten sind.

8.9.2 Nachführung

Spätestens am Ende einer Phase werden bisher erstellte Dokumente auf eine Aktualisierung überprüft, und diese wird, falls nötig, ausgeführt. Der Projektplan wird ständig, spätestens jedoch am Ende eines Arbeitpaketes aktualisiert.

8.10 Prüfungen

Alle Dokumente werden internen Prüfungen unterzogen und dem Betreuer vorgelegt. Eine externe Prüfung der Dokumente unter Teilnahme des Industriepartners ist am Ende der Kundenanalyse vorgesehen.

8.11 Abnahmeprozedur

Der Projekt gilt als beendet, wenn folgende Punkte erfüllt sind:

- Prof. Ludewig akzeptiert diesen Projektbericht
- der Abschlussvortrag wurde erfolgreich durchgeführt

8.12 Berichtswesen

Es findet ein wöchentliches Treffen zwischen dem Betreuer und den Projektteilnehmern statt.

Temin: Montags, 13 Uhr
Ort: Büro Markus Knauß 1.242

8.13 Risiken

8.13.1 Mangelnde Hardwareinfrastruktur

Eine realistische Untersuchung der .NET und Java-Technologien für mobile Rechner und speziell auch der Applikation des Kunden setzt voraus, Zugang zu einem PDA zu erhalten. Wäre dies nicht oder nur in sehr eingeschränktem Maß möglich gewesen, hätten viele grundlegende Untersuchungen (wie etwa Performance-Tests) praktisch nicht ermöglicht werden können.

Eintrittswahrscheinlichkeit: mittel
Schadensausmaß: hoch
Das Problem ist nicht aufgetreten, wir bekamen einen PDA zur Verfügung gestellt.

8.13.2 Probleme mit der Softwareinfrastruktur

Es könnte zu Problemen mit der Systemsoftware des PDAs (z.B. Windows CE/Mobile) oder einer Emulator-Software kommen.

Eintrittswahrscheinlichkeit: hoch
Schadensausmaß: niedrig
Es gab zwar kleinere Probleme, die jedoch behoben werden konnten.

8.13.3 Kostenpflichtige Software

Während der Fachstudie könnte es Sinn machen, Software in einen Vergleich miteinzubeziehen, welche kostenpflichtig ist. Hier kann die Verfügbarkeit nicht sichergestellt werden.

Eintrittswahrscheinlichkeit: mittel
Schadensausmaß: mittel
Das Problem ist nicht aufgetreten, es gab von jeder Software zumindest eine entsprechende Testversion.

8.13.4 Verzögerungen

Durch unvorhersehbare Abwesenheit externer Projektbeteiligter könnte es zu Verzögerungen in der Planung kommen.

Eintrittswahrscheinlichkeit: gering
Schadensausmaß: gering
Das Problem ist nicht aufgetreten.

8.13.5 Ausfall eines Teammitglieds

Der Fall, dass ein Teammitglied aus unbekannten Gründen vorzeitig aus dem Team aussteigt und so die Fachstudie zu zweit beendet werden oder auch abgebrochen werden muss ist denkbar, jedoch sehr unwahrscheinlich.

Eintrittswahrscheinlichkeit: gering
Schadensausmaß: hoch
Das Problem ist nicht aufgetreten.

8.14 Gantt-Diagramm

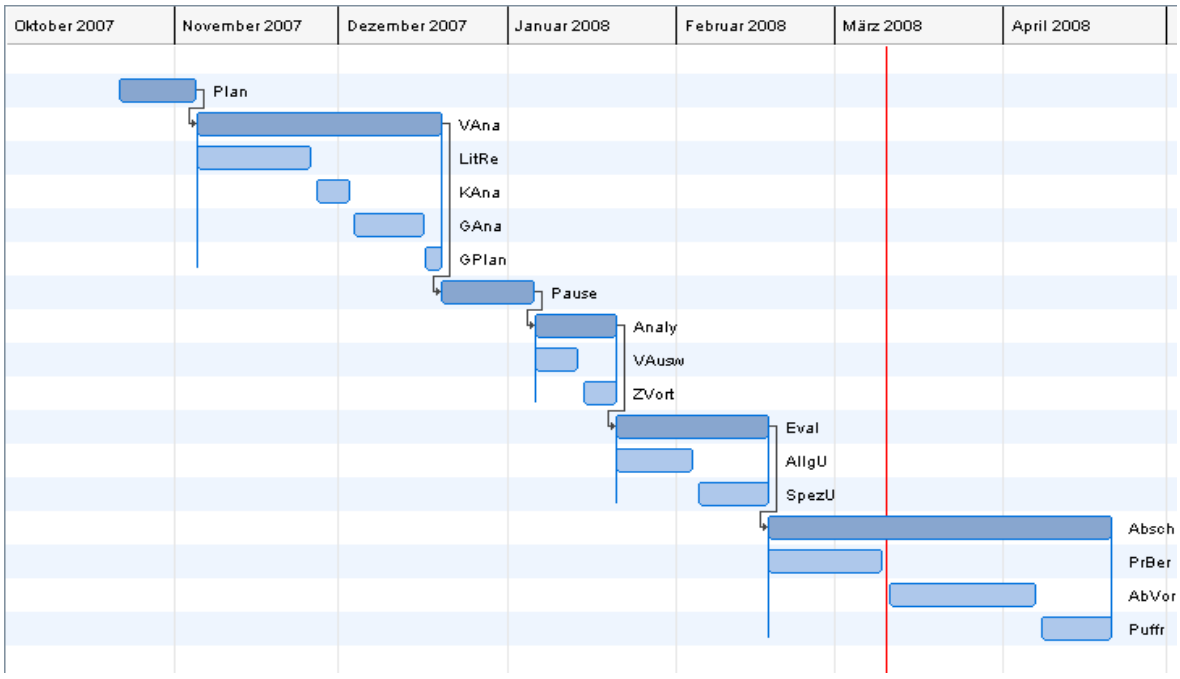


Abbildung 13: Gantt-Diagramm

8.15 Termindrift-Diagramm

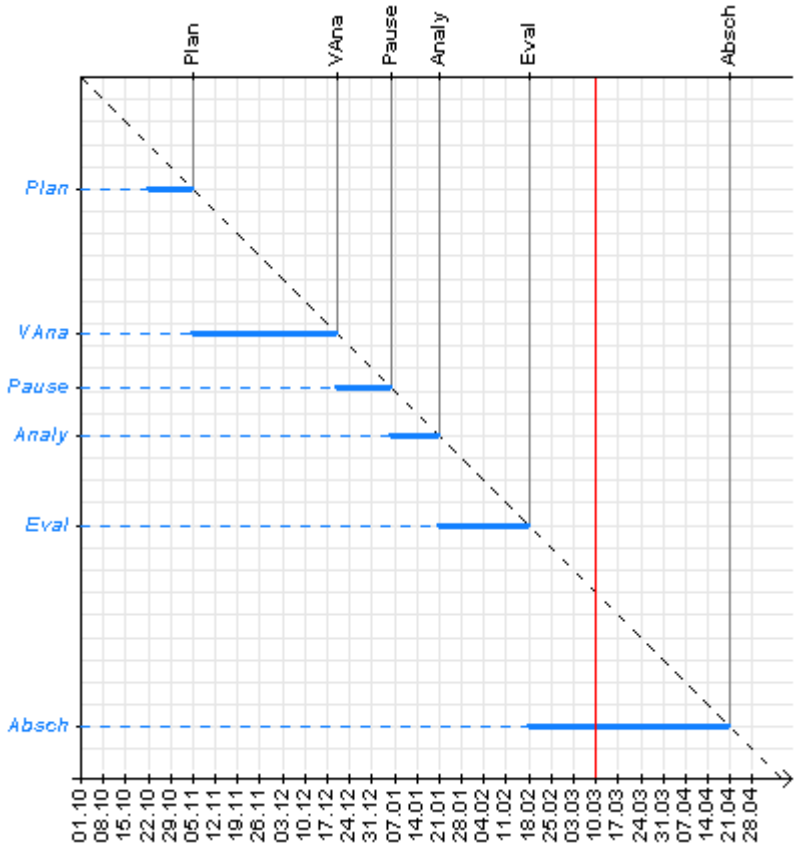


Abbildung 14: Termindrift-Diagramm

Anhang A – Literaturliste

.Net Compact Framework

Real-Time Behavior of the .NET Compact Framework

Link:	http://kurser.iha.dk/eit/itwce01/Articles/Real-Time%20Behavior%20of%20the%20.NET%20-Compact%20Framework.pdf
Quelle:	Maarten Struys, Michel Verhagen, 03.07.2003
Typ:	Artikel
Zusammenfassung:	Erklärt wie man in .NET per P-Invoke (aufrufe in DLLs, z.B. in C++ programmiert) Prozeduren aufrufen kann, falls man real-time Anforderungen benötigt -> man kann die kritischen Teile in C/C++ programmieren und in .NET aufrufen.
Wissenschaftlicher Anspruch:	4
Relevanz:	3
Verlässlichkeit:	2
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	3 (vielleicht kann man die SQL Routinen zur Not auslagern, Hinweis: bei Java2ME gibt es keine Möglichkeit per JNI C/C++ aufzurufen)

.NET Compact Framework Programming

Link:	http://simonstl.com/compact/
Quelle:	Simon St.Laurent, 22.02.2005
Typ:	Buch (nicht fertiggestellt)
Zusammenfassung:	Beschreibt die Programmierung für .NET CE. Es wird kein Vergleich zwischen Java und .NET gegeben.
Wissenschaftlicher Anspruch:	-
Relevanz:	-
Verlässlichkeit:	-
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	-

.NET Compact Framework Entwicklerhandbuch

Link:	http://msdn2.microsoft.com/de-de/library/f44bbwa1.aspx
Quelle:	Microsoft
Typ:	Online-Handbuch
Zusammenfassung:	Allgemeine Einführung in das .NET Compact Framework.
Wissenschaftlicher Anspruch:	-
Relevanz:	-
Verlässlichkeit:	-
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	2

Java ME

Java Platform, Micro Edition

Link:	http://de.wikipedia.org/wiki/Java_Platform,_Micro_Edition
Quelle:	Wikipedia
Typ:	Artikel
Zusammenfassung:	Nur sehr grober Überblick.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	1
Bewertung	4

Sun will Java Micro Edition einstellen

Link:	http://www.zdnet.de/news/tkomm/0,39023151,39158552,00.htm
Quelle:	ZDNet, 22.10.2007
Typ:	Artikel
Zusammenfassung:	Beschreibt, dass Sun Java2ME einstellen will, da schrittweise die Standard Edition für Mobilgeräte eingesetzt werden soll.
Wissenschaftlicher Anspruch:	-
Relevanz:	-
Verlässlichkeit:	-
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	-

Diplomarbeit Portierung von Desktop Java Anwendungen nach Java ME

Link:	http://stz-ida.de/download/forschung/da/da_falk_11284.pdf
Quelle:	Universität Karlsruhe, Sven Falk 15.8.2003
Typ:	Diplomarbeit
Zusammenfassung:	Es werden Java ME und dazu passende Laufzeitumgebungen beschrieben. Außerdem wird diskutiert wie man Java Desktop Anwendungen auf mobile Geräte portiert.
Wissenschaftlicher Anspruch:	2
Relevanz:	2
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	4
Bewertung	2

Seminarvortrag TU München Java ME Wireless Kommunikation

Link:	http://wwwspies.in.tum.de/MVS/sem06/contents/MobileServicePlattformen_Folien.ppt
Quelle:	TU München, Tsvetomir Matov, Datum Unbekannt, sieht aktuell aus
Typ:	Seminarvortrag
Zusammenfassung:	Vorstellung Java ME mit APIs, Schwerpunkt Wireless Kommunikation, .NET compact als alternative, Ausarbeitung wäre gut
Wissenschaftlicher Anspruch:	2
Relevanz:	2
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	2
Bewertung	2

Java Micro Edition

Link:	http://java.sun.com/javame/index.jsp
Quelle:	Sun
Typ:	Dokumentation
Zusammenfassung:	Einführung und Beschreibung der Java Micro Edition
Wissenschaftlicher Anspruch:	-
Relevanz:	-
Verlässlichkeit:	-
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	2

Vergleiche zwischen .NET Compact und Java ME

Seminar Uni Koblenz dieses Semester

Link:	http://www.uni-koblenz.de/FB4/Institutes/IWVI/AGHampe/Teaching/2007ws/Sem_BKS
Quelle:	Uni Koblenz, 2007/2008
Typ:	Seminar
Zusammenfassung:	Hat noch nicht statt gefunden. Ein Thema: Java vs. .NET CE sollte weiter verfolgt werden
Wissenschaftlicher Anspruch:	-
Relevanz:	-
Verlässlichkeit:	-
Unvoreingenommenheit:	-
Aktualität	-
Bewertung	-

Presse Artikel Wirtschaftsinformatiker entwickeln Mobile Anwendungen

Link:	http://www.presse.uni-augsburg.de/unipress/up20042/artikel_23.shtml
Quelle:	Universität Augsburg, 11.7.2004
Typ:	Pressemitteilung
Zusammenfassung:	Ohne weitere Erklärungen wird behauptet, dass aufgrund von Erfahrungen für Handys Java ME und für PDA .NET Compact besser geeignet ist.
Wissenschaftlicher Anspruch:	2
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	4
Aktualität	2
Bewertung	2

Vergleichsartikel auf dotnetguru zwischen .NET und Java ME von 2002

Link:	http://www.dotnetguru.org/articles/J2MEvsSDE/J2MEvsSDE.htm
Quelle:	Dotnetguru.org , Sami Jaber, Datum 2002.
Typ:	Vergleichsartikel
Zusammenfassung:	Leider Französisch, vergleicht .NET und Java ME auf technischem Standpunkt. Fazit: Java Multiplatform, .NET Windows CE
Wissenschaftlicher Anspruch:	4
Relevanz:	2
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	4
Bewertung	2

Mobile Computing Betriebssysteme und Entwicklungsumgebungen

Link:	http://www.4fo.de/download/MobileComputing.pdf
Quelle:	4FriendsOnly.com Internet Technologies AG, 12.07.2002
Typ:	Ausarbeitung eines Hauptseminars
Zusammenfassung:	Beschreibt Entwicklungsumgebungen für mobile Endgeräte. Schwerpunkt ist Java2ME, da zu diesem Zeitpunkt .NET CE noch nicht sehr verbreitet war.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	4
Bewertung	4

Schaffung einer internetbasierten Softwaretankstelle zur Herstellung der technischen Voraussetzungen für die Teilnahme am DIANE Ad-hoc Netzwerk

Link:	http://wwwipd.ira.uka.de/~obreiter/dasa/2003Liu.pdf
Quelle:	Universität Karlsruhe, 10.06.2003
Typ:	Studienarbeit
Zusammenfassung:	Enthält unter Anderem einen Vergleich zwischen .NET CE und Java2ME. In dem Vergleich wird .NET als schneller beschrieben und hervorgehoben das .NET bereits WebService-Unterstützung integriert hat, während bei Java extra Bibliotheken benötigt werden.
Wissenschaftlicher Anspruch:	2
Relevanz:	2
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	3
Bewertung	2

Details zu Java 2 Micro Edition

Link:	http://www.tecchannel.de/webtechnik/soa/401850/index5.html
Quelle:	TecChannel, 22.04.2002
Typ:	Artikel
Zusammenfassung:	Nur grober Überblick über Java2ME Plattform.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	4
Bewertung	4

Pervasive Computing

Link:	http://www.cs.uni-potsdam.de/bs/teaching/ss2006/pervasive.html#Helal
Quelle:	Uni Potsdam
Typ:	Vortrag mit Ausarbeitung
Zusammenfassung:	Oberflächlicher kurzer Vergleich mit wenig Begründungen
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	4
Unvoreingenommenheit:	3
Aktualität	3
Bewertung	4

Mobile Service Platformen (Java MIDP 2.0 und verwandte JSRs)

Link:	http://wwwspies.in.tum.de/MVS/sem06/contents/MobileServicePlattformen_Ausarbeitung.pdf
Quelle:	Technische Universität München
Typ:	Artikel
Zusammenfassung:	Kapitel 3: Java ME, Kapitel 7: mit .NET Compact Framework im Vergleich Ausführlicher und vielseitiger Artikel, der viele beim Vergleich zu beachtende Aspekte aufzählt.
Wissenschaftlicher Anspruch:	2
Relevanz:	3
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	2
Bewertung	2

Microsoft Versus Java: The Next Frontier

Link:	http://www.informationweek.com/story/showArticle.jhtml?articleID=6503512
Quelle:	InformationWeek
Typ:	Artikel
Zusammenfassung:	Langer, allgemeiner Artikel, der erläutert, weshalb ein features-to-features Vergleich bei weitem nicht ausreicht. Beide Technologien könnten sehr viel.
Wissenschaftlicher Anspruch:	3
Relevanz:	3
Verlässlichkeit:	3
Unvoreingenommenheit:	4
Aktualität	4
Bewertung	4

Java ME vs. .Net Compact Framework

Link:	http://www.ise.canberra.edu.au/un7107/presentation2007/J2ME%20VS%20NET.ppt
Quelle:	University of Canberra
Typ:	Präsentation
Zusammenfassung:	Der Vergleich greift einige einzelne Aspekte der Technologien auf, begründet aber die Feststellungen kaum. Der Performance-Vergleich ist aufgrund der Unterschiedlichkeit der getesteten Systeme fragwürdig.
Wissenschaftlicher Anspruch:	3
Relevanz:	3
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	2
Bewertung	3

Mobile Development

Link:	http://en.wikipedia.org/wiki/Mobile_development
Quelle:	Wikipedia
Typ:	Artikel
Zusammenfassung:	Schöner Vergleich mit großer Tabelle, der viele verschiedene Aspekte nennt und diese klar bewertet.
Wissenschaftlicher Anspruch:	4
Relevanz:	2
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	1
Bewertung	2

J2ME vs .Net Compact

Link:	http://www.cityware.org.uk/index.php?option=com_content&task=view&id=21&Itemid=41
Quelle:	Cityware
Typ:	Bericht
Zusammenfassung:	Sehr ausführlicher (teils tabellarischer) Vergleich der Features, der jedoch einige Punkte nicht begründet.
Wissenschaftlicher Anspruch:	3
Relevanz:	3
Verlässlichkeit:	2
Unvoreingenommenheit:	3
Aktualität	1
Bewertung	3

Eine Mobile Business Anwendung mit zeitnahem Datenabgleich

Link:	http://wwwbayer.in.tum.de/aktuelles/GI-Workshop/themen/osterhold.pdf
Quelle:	Fachhochschule Gelsenkirchen (Veröffentlichung von der Technischen Universität München)
Typ:	Ausarbeitung eines Vortrags
Zusammenfassung:	Erläutert nur kurz einige wenige Aspekte im Vergleich (Kap. 5), der Rest ist praktisch nicht relevant.
Wissenschaftlicher Anspruch:	3
Relevanz:	4
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	3
Bewertung	3

Mobile Applications

Link:	http://www.ise.canberra.edu.au/un7107/presentation2007/Mobile%20Applications.ppt
Quelle:	University of Canberra
Typ:	Präsentation
Zusammenfassung:	Enthält ein paar interessante Pro/Contra Argumente im Vergleich, ansonsten eher kurz. Inhalte und Vergleichstabelle kommen bereits an anderer Quelle vor.
Wissenschaftlicher Anspruch:	3
Relevanz:	3
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	1
Bewertung	3

Programming interactive real-time games over WLAN for pocket PCs with J2ME and .NET CF

Link:	http://portal.acm.org/citation.cfm?id=1103603
Quelle:	University of Vienna (publiziert über ACM)
Typ:	Wissenschaftlicher Artikel
Zusammenfassung:	Sehr ausführlicher Performance-Analyse mit Auswertung. Leider starker Fokus auf Spieleent-wicklung und WLAN.
Wissenschaftlicher Anspruch:	2
Relevanz:	3
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	2
Bewertung	2

Ein XML-basiertes Modell für synchrone Gruppenarbeit auf gemeinsamen Informationsräumen

Link:	http://deposit.ddb.de/cgi-bin/dokserv?idn=968862853&dok_var=d1&dok_ext=pdf&filename=968862853.pdf#page=190
Quelle:	Universität Kassel (publiziert über die Deutsche Nationalbibliothek)
Typ:	Dissertation
Zusammenfassung:	Nur vereinzelt Passagen über .NET CE und Java ME (ab S.36). Ansonsten nicht relevant.
Wissenschaftlicher Anspruch:	3
Relevanz:	5
Verlässlichkeit:	4
Unvoreingenommenheit:	3
Aktualität	3
Bewertung	4

Mobile Information Access

Link:	http://wwwmayr.informatik.tu-muenchen.de/konferenzen/Jass05/courses/6/Papers/12.pdf
Quelle:	Technische Universität München
Typ:	Paper
Zusammenfassung:	Kapitel 3.4 enthält einen Abschnitt mit ausführlich beschriebenem Vergleich. Teilweise fehlen jedoch die Begründungen.
Wissenschaftlicher Anspruch:	2
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	3
Bewertung	3

.NET vs. Java - Mobilklienter og web services

Link:	http://www.dmal.dk/CV/java%20vs%20.net.pdf
Quelle:	Universität Aalboerg (Dänemark)
Typ:	Wissenschaftliche Ausarbeitung
Zusammenfassung:	Sehr ausführliche Untersuchung des Themas. Leider auf dänisch.
Wissenschaftlicher Anspruch:	2
Relevanz:	3
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	2
Bewertung	3

Mobile multimodal applications on mass-market devices: experiences

Link:	http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4312863
Quelle:	IEEE
Typ:	Wissenschaftlicher Artikel
Zusammenfassung:	Sehr allgemeiner Artikel über mobile Anwendungen. Kaum relevante Teile.
Wissenschaftlicher Anspruch:	3
Relevanz:	5
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	4
Bewertung	4

Vergleiche zwischen .NET und Java SE & Java EE

Vergleichsartikel zwischen .NET und Java EE von 2007

Link:	http://whitepapers.zdnet.co.uk/0,1000000651,260040089p,00.htm
Quelle:	zdnet, 2007
Typ:	Vergleichsartikel
Zusammenfassung:	Aktueller Vergleich zwischen .NET und Java EE
Wissenschaftlicher Anspruch:	3
Relevanz:	3
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	1
Bewertung	3

Vergleich von Tecchannel zwischen Sun One und .NET von 2002

Link:	http://www.tecchannel.de/webtechnik/entwicklung/401577/
Quelle:	techchannel, 2001.
Typ:	Vergleichsartikel
Zusammenfassung:	Vergleich zwischen Sun One und .NET aus nicht technischer Sicht, eher aus business Strategie-Sicht. Zu alter Artikel.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	3
Aktualität	5
Bewertung	5

Vortrag welche Strategien Sun und Microsoft mit One und .NET verfolgen

Link:	http://www.cpp-entwicklung.de/downld/Strategievergleich.pdf
Quelle:	Siemens, Dr.Thomas Wieland, Datum unbekannt.
Typ:	Vortrag
Zusammenfassung:	Ein Vergleich zwischen Sun One und .NET, es wird auf die Technologien eingegangen. Schwerpunkt liegt auf eBusiness Anwendungen .
Wissenschaftlicher Anspruch:	3
Relevanz:	4
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	3
Bewertung	3

Oreillynet Vergleich zwischen .NET und Java 2000

Link:	http://www.oreillynet.com/pub/a/oreilly/java/news/farley_0800.html
Quelle:	Oereillynet, Jim Farely, Datum 2000.
Typ:	News
Zusammenfassung:	Ein Vergleich zwischen .NET und Java EE. Allerdings sehr alt.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	4
Unvoreingenommenheit:	4
Aktualität	6
Bewertung	4

Performancevergleich Java und .NET auf Bentuser.com

Link:	http://www.bentuser.com/article.aspx?ID=323
Quelle:	Bentuser, Andy, Datum 2005.
Typ:	Vergleichsartikel
Zusammenfassung:	Java 1.5 und .NET 2.0 werden auf Performance hin verglichen.
Wissenschaftlicher Anspruch:	4
Relevanz:	4
Verlässlichkeit:	3
Unvoreingenommenheit:	2
Aktualität	3
Bewertung	3

Master Thesis Umstieg von Java auf .NET

Link:	http://www.engineering.leeds.ac.uk/comp/pg/documents/panesar.pdf
Quelle:	Bentuser, Mandip Singh Panesar, Datum 2004/2005.
Typ:	Master Thesis
Zusammenfassung:	.NET Guide für Java Entwickler, vergleichend, Mono Framework
Wissenschaftlicher Anspruch:	2
Relevanz:	4
Verlässlichkeit:	2
Unvoreingenommenheit:	2
Aktualität	3
Bewertung	3

Anhang B – Microbenchs

Testfälle

Testfall 1	Leerer Testfall um Overhead des Testaufbaus zu messen
	`t1 = GetTimeStamp();` `t2 = GetTimeStamp();`

Testfall 2	1.000.000 double-word integer Inkrementierungen

	`int var = 0;`
	`t1 = GetTimeStamp();`
	`while(var < 1.000.000)`
	`  var = var + 1;`
	`t2 = GetTimeStamp();`

Testfall 3	1.000.000 for-loop iterationen

	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000.000; ++i) {}`
	`t2 = GetTimeStamp();`

Testfall 4	1.000.000 modulo Operationen

	`int var = 0;`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000.000; ++i)`
	`  var = i modulo 1.000;`
	`t2 = GetTimeStamp();`

Testfall 5	10.000 String Konkatenationen*

	`string var = "";`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 10.000; ++i)`
	`  str += "a";`
	`t2 = GetTimeStamp();`

* auf die Verwendung von StringBuffer wurde verzichtet

Testfall 6	30.000 String Konkatenationen

	`string var = 0;`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 30.000; ++i)`
	`  str += "a";`
	`t2 = GetTimeStamp();`

Testfall 7	1.000.000 Instanzierungen einer leeren Klasse

	`class Empty`
	`{`
	`};`
	
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000.000; ++i)`
	`  Empty e = new Empty();`
	`t2 = GetTimeStamp();`

Testfall 8	1.000.000 Instanzierungen einer simplen Klasse

	`class Simple`
	`{`
	`  private int _intMember;`
	`  private string _stringMember;`
	
	`  public Simple()`
	`  {`
	`  _intMember = 0;`
	`  _stringMember = "test";`
	`  }`
	`};`
	
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000.000; ++i)`
	`  Simple s = new Simple();`
	`t2 = GetTimeStamp();`

Testfall 9	1.000.000 Funktionsaufrufe einer leeren Memberfunktion

```
class FnTest
{
      private int _intMember;
      private string _stringMember;

      public FnTest()
      {
            _intMember = 0;
            _stringMember = "test";
      }

      public EmptyFn()
      {
      }
};

FnTest fn = new FnTest();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      fn.EmptyFn();
t2 = GetTimeStamp();
```

Testfall 10	1.000.000 Funktionsaufrufe einer leeren statischen Funktion

```
class StaticFnTest
{
      static public EmptyStaticFn()
      {
      }
};

t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      StaticFnTest.EmptyStaticFn();
t2 = GetTimeStamp();
```

Testfall 11	1.000.000 Funktionsaufrufe einer double word Integer Setter-Memberfunktion

```
class FnTest
{
      private int _intMember;
      private string _stringMember;

      public FnTest()
      {
            _intMember = 0;
            _stringMember = "test";
      }

      public SetInt32(int val)
      {
            _intMember = val;
      }
};

FnTest fn = new FnTest();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      fn.SetInt32(i);
t2 = GetTimeStamp();
```

Testfall 12	1.000.000 Funktionsaufrufe einer statischen double word Integer Setterfunktion

```
class StaticFnTest
{
      static private _staticInt;

      static public StaticSetInt32(int val)
      {
            _staticInt = val;
      }
};

t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      StaticFnTest.StaticSetInt32(i);
t2 = GetTimeStamp();
```

Testfall 13	1.000.000 Funktionsaufrufe einer string Setter-Memberfunktion

```
class FnTest
{
      private int _intMember;
      private string _stringMember;

      public FnTest()
      {
            _intMember = 0;
            _stringMember = "test";
      }

      public SetString(string val)
      {
            _stringMember = val;
      }
};

FnTest fn = new FnTest();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      fn.SetString("test123");
t2 = GetTimeStamp();
```

Testfall 14	1.000.000 Funktionsaufrufe einer statischen string Setterfunktion

```
class StaticFnTest
{
      static private _staticString;

      static public StaticSetString(string val)
      {
            _staticString = val;
      }
};

t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      StaticFnTest.StaticSetString("test123");
t2 = GetTimeStamp();
```

Testfall 15	1.000.000 überladene Funktionsaufrufe einer double word Integer Setter-Member-funktion

```
class FnTest
{
      private int _intMember;
      private string _stringMember;

      public Set(int val)
      {
            _intMember = val;
      }

      public Set(string val)
      {
            _stringMember = val;
      }
};

FnTest fn = new FnTest();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      fn.Set(i);
t2 = GetTimeStamp();
```

Testfall 16	1.000.000 überladene Funktionsaufrufe einer statischen double word Integer Setter-funktion

```
class StaticFnTest
{
      static private _staticInt;
      static private _staticString;

      static public StaticSet(int val)
      {
            _staticInt = val;
      }

      static public StaticSet(string val)
      {
            _staticString = val;
      }
};

t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      StaticFnTest.StaticSet(i);
t2 = GetTimeStamp();
```

Testfall 17	1.000.000 überladene Funktionsaufrufe einer string Setter-Memberfunktion

```
class FnTest
{
      private int _intMember;
      private string _stringMember;

      public Set(int val)
      {
            _intMember = val;
      }

      public Set(string val)
      {
            _stringMember = val;
      }
};

FnTest fn = new FnTest();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      fn.Set("test123");
t2 = GetTimeStamp();
```

Testfall 18	1.000.000 überladene Funktionsaufrufe einer statischen string Setterfunktion

```
class StaticFnTest
{
      static private _staticInt;
      static private _staticString;

      static public StaticSet(int val)
      {
            _staticInt = val;
      }

      static public StaticSet(string val)
      {
            _staticString = val;
      }
};

t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      StaticFnTest.StaticSet("test123");
t2 = GetTimeStamp();
```

Testfall 19	1.000.000 direkte virtuelle Funktionsaufrufe einer string Getter-Memberfunktion

```
class Base
{
      virtual public string Get()
      {
            return "test123";
      }
};

class Derived extends Base
{
      public string Get()
      {
            return "test123";
      }
};

Derived polymorph = new Derived();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      polymorph.Get();
t2 = GetTimeStamp();
```

Testfall 20	1.000.000 indirekte virtuelle Funktionsaufrufe einer string Getter-Memberfunktion

```
class Base
{
      virtual public string Get()
      {
            return "test123";
      }
};

class Derived extends Base
{
      public string Get()
      {
            return "test123";
      }
};

Base polymorph = new Derived();
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      polymorph.Get();
t2 = GetTimeStamp();
```

Testfall 21	1.000.000 Arrayzugriffe auf die gleiche Position

```
int[] array = new int[1000];
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      array[87] = i;
t2 = GetTimeStamp();
```

Testfall 22	1.000.000 Arrayzugriffe auf eine per Modulo 1000 berechnete Position

```
int[] array = new int[1000];
t1 = GetTimeStamp();
for(int i = 0; i < 1.000.000; ++i)
      array[i modulo 1000 ] = i;
t2 = GetTimeStamp();
```

Testfall 23	1.000.000 Arrayzugriffe auf eine per Modulo array.Length berechnete Position

	`int[] array = new int[1000];`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000.000; ++i)`
	`  array[i modulo array.Length] = i;`
	`t2 = GetTimeStamp();`

Testfall 24	1.000 Iterationen über ein 1.000 elementiges Array mit Hilfe von for

	`int[] array = new int[1000];`
	`int val;`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000; ++i)`
	`{`
	`  for(int j = 0; j < 1.000; ++j)`
	`  val = array[j];`
	`}`
	`t2 = GetTimeStamp();`

Testfall 25	1.000 Iterationen über ein 1.000 elementiges Array mit Hilfe von for-each (nur bei Compact Framework)

	`int[] array = new int[1000];`
	`int val;`
	`t1 = GetTimeStamp();`
	`for(int i = 0; i < 1.000; ++i)`
	`{`
	`  foreach(int j in fixedArray)`
	`  val = array[j];`
	`}`
	`t2 = GetTimeStamp();`

Ergebnisse Compact Framework 1.0

Alle Werte in Millisekunden.

	Run 1	Run 2	Run 3	Run 4	Run 5	Mittel
Testfall 1	3	4	4	3	3	3,4
Testfall 2	25	25	26	26	25	25,4
Testfall 3	29	29	27	27	27	27,8
Testfall 4	313	314	313	314	313	313,4
Testfall 5	11978	12154	11830	11886	11878	11945,2
Testfall 6	103443	103376	102955	102902	103014	103138
Testfall 7	5195	5190	5870	5196	5180	5326,2
Testfall 8	5633	5604	5604	5615	5597	5610,6
Testfall 9	54	54	54	54	55	54,2
Testfall 10	26	25	25	26	25	25,4
Testfall 11	239	245	240	239	239	240,4
Testfall 12	207	207	206	206	206	206,4
Testfall 13	255	254	255	255	255	254,8
Testfall 14	206	207	206	206	206	206,2
Testfall 15	239	238	239	239	239	238,8
Testfall 16	206	206	206	207	206	206,2
Testfall 17	256	255	255	254	255	255
Testfall 18	206	206	206	206	206	206
Testfall 19	323	323	322	322	323	322,6
Testfall 20	322	322	323	323	322	322,4
Testfall 21	85	83	83	84	84	83,8
Testfall 22	378	372	372	372	372	373,2
Testfall 23	403	402	408	402	408	404,6
Testfall 24	79	80	79	85	80	80,6
Testfall 25	104	103	102	104	103	103,2

Summe	130207	130278	130210	129553	129621	**129973,8**

Ergebnisse Compact Framework 3.5

Alle Werte in Millisekunden.

	Run 1	Run 2	Run 3	Run 4	Run 5	Mittel
Testfall 1	2	2	2	2	2	2
Testfall 2	21	21	21	21	22	21,2
Testfall 3	22	22	22	22	21	21,8
Testfall 4	244	194	194	194	195	204,2
Testfall 5	1710	1542	1594	1578	1560	1596,8
Testfall 6	14642	14135	14090	14110	14281	14251,6
Testfall 7	709	701	700	700	703	702,6
Testfall 8	886	869	863	891	869	875,6
Testfall 9	31	31	31	31	31	31
Testfall 10	31	30	31	31	31	30,8
Testfall 11	24	24	23	24	23	23,6
Testfall 12	29	29	29	29	29	29
Testfall 13	46	39	39	38	39	40,2
Testfall 14	40	40	41	40	41	40,4
Testfall 15	23	24	23	23	23	23,2
Testfall 16	41	40	41	42	41	41
Testfall 17	39	39	39	39	39	39
Testfall 18	40	40	40	40	40	40
Testfall 19	132	132	133	133	132	132,4
Testfall 20	133	133	132	132	132	132,4
Testfall 21	39	39	38	39	39	38,8
Testfall 22	236	229	229	229	216	227,8
Testfall 23	238	237	238	238	255	241,2
Testfall 24	50	51	50	50	51	50,4
Testfall 25	72	71	72	72	72	71,8

Summe	19480	18714	18715	18748	18887	**18908,8**

Ergebnisse J9

Alle Werte in Millisekunden.

	Run 1	Run 2	Run 3	Run 4	Run 5	Mittel
Testfall 1	1	1	0	0	1	0,6
Testfall 2	162	162	163	163	165	163
Testfall 3	163	163	165	162	162	163
Testfall 4	464	482	1088	464	465	592,6
Testfall 5	3213	3188	3218	3205	3194	3203,6
Testfall 6	28158	28379	28149	27971	28366	28204,6
Testfall 7	740	744	740	747	751	744,4
Testfall 8	1032	1059	1033	1032	1033	1037,8
Testfall 9	462	461	463	454	454	458,8
Testfall 10	416	583	417	417	416	449,8
Testfall 11	539	545	539	540	540	540,6
Testfall 12	444	445	445	445	446	445
Testfall 13	592	592	592	591	593	592
Testfall 14	692	667	666	676	669	674
Testfall 15	539	539	544	538	538	539,6
Testfall 16	445	446	444	445	486	453,2
Testfall 17	591	596	592	591	592	592,4
Testfall 18	675	667	676	667	667	670,4
Testfall 19	525	525	524	526	525	525
Testfall 20	522	530	522	523	522	523,8
Testfall 21	273	274	274	274	275	274
Testfall 22	530	531	530	530	531	530,4
Testfall 23	556	555	556	563	555	557
Testfall 24	269	269	268	273	268	269,4
Testfall 25						

Summe	42003	42403	42608	41797	42214	**42205**

Anhang C – Phasen der Entwicklung

Planungsphase (22.10.2007 – 05.11.2007)

Arbeitspaket Planung (bis 05.11.2007):

Es wird das grobe Vorgehen der Fachstudie und die Rahmenbedingungen für das Projekt definiert. Diese werden zusammen im Projektplan festgehalten. Er enthält auch eine zeitliche Einordnung der einzelnen Phasen.

Dokument: Projektplan

Meilenstein Planungsphase

Voranalyse (06.11.2007 – 21.12.2007)

Arbeitspaket Literaturrecherche (bis 26.11.2007):

Zunächst soll herausgefunden werden, ob es bereits Erfahrungen mit .NET und Java auf mobilen Rechnern gibt, und falls ja, wie diese ausfallen. Die Recherche wird primär im Internet, jedoch auch in der Literatur durchgeführt.

Dokument: Literaturrecherche

Arbeitspaket Kundenanalyse (bis 03.12.2007):

Ausgehend von den bisherigen Resultaten wird ein Fragenkatalog erstellt, der die derzeitige und die gewünschte Situation des Kunden klären soll. Dieser Fragenkatalog soll helfen, in einem daraufhin stattfindenden offenen Analysegespräch mit dem Kunden alle weiteren offenen Fragen zu klären. Dieses Gespräch fand am 28.11.2007 statt.

Dokument: Kundenanalyse

Arbeitspaket Gegenstandsanalyse (bis 17.12.2007):

Bei dieser Analyse ohne direkte Beteiligung des Kunden werden die theoretischen Möglichkeiten der .NET- und Java-Technologien in Bezug auf mobilen Rechnern, sowie alle weiteren benötigten Technologien (wie z.B. die verwendete Datenbanklösung) untersucht. Dazu werden Alternativen gesucht, analysiert und bewertet.

Dokument: Gegenstandsanalyse

Arbeitspaket Gegenstandsplanung (bis 21.12.2007):

Hier werden die Ergebnisse der vorherigen drei Arbeitspakete genutzt um genau zu klären, was im Rahmen dieser Fachstudie untersucht werden soll. Falls eine Erweiterung oder Veränderung der Aufgabenstellung nötig ist, wird das an dieser Stelle mit allen Beteiligten abgesprochen.

Dokument: Gegenstandsplanung

Meilenstein Voranalyse

Analyse (06.01.2008 – 21.01.2008)

Arbeitspaket Vorauswertung (06.01.2008 – 14.01.2008):

In Bezug auf die Gegenstandsplanung werden erste konkrete Tests am PDA durchgeführt und bewertet. Diese Resultate fließen in die bisher gewonnenen Erkenntnisse mit ein, woraus ein vorläufiger Zwischenstand der Fachstudie entsteht und dieser dokumentiert wird.

Dokument: Vorauswertung

Arbeitspaket Zwischenvortrag (bis 21.01.2008):

Dieser Zwischenstand wird in einem Vortrag allen Beteiligten der Fachstudie präsentiert. Im Anschluss daran findet eine offene Diskussion statt, bei der alle Fragen geklärt und eventuell die Richtung der Fachstudie korrigiert werden kann.

Dokument: Zwischenvortrag

Meilenstein Analyse

Evaluierung (22.01.2008 – 18.02.2008)

Arbeitspaket Allgemeine Untersuchung (bis 04.02.2008):

Hier findet die eigentliche Untersuchung der Eignung von Java für mobile Rechner mit Windows CE/mobile statt. Ein detaillierter, jedoch allgemeiner Vergleich der Technologien mit Ergebnissammlung findet statt. Der genaue Inhalt dieses Arbeitspaketes wird erst nach dem Zwischenvortrag festgelegt, beispielsweise könnten die Migrationsmöglichkeiten mit ihren Aufwänden untersucht werden.

Dokument: AllgemeineUntersuchung

Arbeitspaket Spezielle Untersuchung (bis 18.02.2008):

Basierend auf den bisher gewonnenen Erkenntnissen werden nun konkrete Tests und Untersuchungen speziell an der Applikation des Kunden durchgeführt und dokumentiert. Auch hier wird der genaue Inhalt des Arbeitspaketes erst nach dem Zwischenvortrag festgelegt.

Dokument: SpezielleUntersuchung

Meilenstein Evaluierung

Projektabschluss (19.02.2008 – 21.04.2008)

Arbeitspaket Projektbericht (bis 10.03.2008):

Alle Ergebnisse der Fachstudie werden in einem abschließenden Projektbericht ausgearbeitet. Insbesondere wird außer dem allgemeinen und dem speziellen Teil eine abschließende Empfehlung an den Kunden formuliert.

Dokument: Projektbericht

Arbeitspaket Abschlussvortrag (bis 07.04.2008):

Der Projektbericht wird als Präsentation aufbereitet und vor allen an der Fachstudie beteiligten Personen gegen Anfang April vorgetragen.

Dokument: Abschlussvortrag

Puffer (bis 21.04.2008):

Um eventuelle Verzögerungen abzufangen, ist bis zum Projektende noch ein Puffer eingeplant.

Meilenstein Projektabschluss

Abbildungsverzeichnis

Abbildung 1: Übersicht der Lösungen..... 19
Abbildung 2: Übersicht Runde I..... 25
Abbildung 3: Zwischenergebnis nach Runde I..... 27
Abbildung 4: Übersicht Runde II..... 27
Abbildung 5: Zwischenergebnis nach Runde II..... 29
Abbildung 6: Übersicht Runde III..... 29
Abbildung 7: Zwischenergebnis nach Runde III..... 31
Abbildung 8: Übersicht Runde IV..... 31
Abbildung 9: Endauswahl nach der letzten Runde..... 32
Abbildung 10: Ausführungszeiten der Microbenchmark-Testfälle..... 43
Abbildung 11: Ausführungszeiten der Microbenchmark-Testfälle (skaliert)..... 43
Abbildung 12: Auf CF 1.0 genormte Geschwindigkeiten der Testfälle..... 44
Abbildung 13: Gantt-Diagramm..... 62
Abbildung 14: Termindrift-Diagramm..... 63

Wir versichern, diese Fachstudie selbständig durchgeführt und nur die angegebenen Quellen verwendet zu haben.

Tobias Arnold	Ivan Bogicevic	Nico Gau