社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－MBL－47 (1)
2008－ITS－35 (1)
2008／11／6

端末開発における開発効率化の一検討

清原 良三　三井 聡　神戸 英利　松本 利夫　小島 泰三

三菱電機(株)

携帯端末やカーナビゲーションシステムのソフトウェア規模が巨大化し，最近では不具合なしの状態での出荷が困難な状況である．こういった大規模なソフトウェアの開発では，例えば通信機能を利用するようなケースにおいてはシミュレータなどでは十分な試験や動作の確認ができないケースも多くあり，実機でしか再現できない障害も多く，実機端末での検証は必須である．ソフトウェア規模が大きくなるとこういった実機端末へのソフトウェアの書き込み時間も長くなり，わずかな修正をして動作を確認しようとしても書込みに時間がかかり非効率的な作業になるケースが多くある．本論文では開発者がソフトウェアの管理などする手間なしにこのようなわずかな修正をした場合に実機端末を高速に書き換える手法を提案し，評価を行いその有効性を示す．

A Study of Development Method for Mobile Devices

Ryozo Kiyohara, Satoshi Mii, Hidetoshi Kambe, Toshio Matsumoto, Taizo Kojima

Mitsubishi Electric Corporation

Due to increasing services of cellular phones(e.g. i-mode), car navigation systems and other embedded devices, it is difficult to release bug-free devices. For development of these type of embedded systems, software should be developed by simulator on the closs platform. But in the phase of system tests, there are a lot of cases which should be confirmed on target devices. In these cases, the cycles of bug fixed and execution should be repeated. Therefore download time of program codes for target devices should be short.

In this paper, we show the fast download algorithm for these devices and evluate it.

1　はじめに

携帯端末やカーナビゲーションシステムのソフトウェア規模が巨大化し，最近では不具合なしでの出荷が困難な状況である．こういった大規模なソフトウェアの開発は，開発期間の関係でH/Wの試作ができあがってからソフトウェアの検証を行うのでは間に合わないため，クロス環境上でシミュレーターなどを利用して開発することも多い．

しかしながら通信機能を利用するようなケースなどシミュレータでは試験や動作の確認ができないケースも多くあるため，実機端末での検証は必須である．ソフトウェア規模が大きくなるとこういった実機端末へのソフトウェアの書き込み時間も長くなり，わずかな修正であっても，実機へのダウンロードに時間がかかり非効率的なケースが多くある．

また，携帯端末やカーナビゲーションシステムなど規模の大きな組込機器で，コンシューマが利用する機器は多くの場合に，ウォーターフォールモデルやスパイラルモデルなどを利用して開発する．組込機器の開発の効率化を目的に図1に示すようなそれぞれのフェーズでの作業の効率化の研究がされており，またソフトウェア資産の再利用などで評価済みのソフトを使うことにより試験のコストも含めて効率化を狙う研究も多数ある[1][2].

しかし，図1に示すような最後のシステム試験の段階では，試験そのものの効率化の取り組みはあるが，実際に不具合を発見してから修正し確認するといった実作業そのものは，物理的なデバイスを利用して確認する必要があることから，環境に依存す

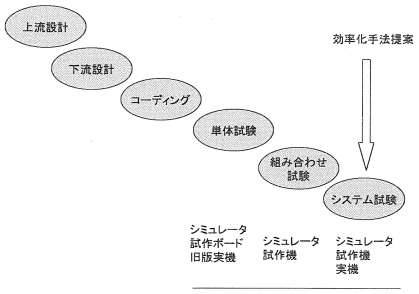

図 1: 典型的な開発モデル

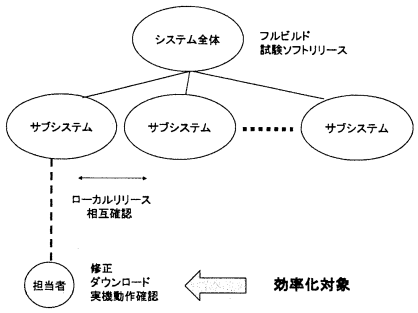

図 2: 試験フェーズでのソフトウェア書き込み

ることが多く，作業者自身の経験による工夫に依存するケースが多い．

本論文ではこのような不具合の修正，実機へ反映，動作の確認といったフェーズの効率化を図るために，実機フラッシュメモリへの書き込みの高速化を実現する方式を提案し，評価し，その効果を示す．

2 システム試験フェーズでの不具合修正

携帯端末などのS/W開発過程の中でシステム試験フェーズでは，機能単体や，その機能と関連のある機能との結合の試験では起きないような不具合に起因する．即ち，想定していない非同期イベントや，想定していない状態に他の機能がなっているなど各機能の処理の想定シーケンスの認識違いや実装間違いに起因することが多い．

こういった不具合は携帯電話であれば実機で試験をするフェーズで起こり，シミュレータ環境やボードの環境では複合的な操作もさせにくく再現しないケースが多い．そのため，ソフトウェアを修正して確認するのも実機を使うことになる．

そのため，実機にソフトウェアをダウンロードしては実行し動作を確認するという作業が多くなる．そのためのソフトウェアを毎日ビルドするようなケースも少なくない [3]．また，不具合そのものを解析するためのツール類の研究も盛んに行われている [4]．

もちろん多くの場合，不具合の特定作業とそれに応じた不具合の修正箇所および修正による他の影響を机上で確認した上で実際の修正を行うことが多い．しかし，その修正の有効性を確認する上では実機に書き込みをした上での動作の確認も必須であり，この作業を繰り返すケースも少なくない．

2.1 開発者の動作確認作業

携帯端末のS/W開発課程ではデバッグを行い，バグを特定した後に以下の手順が繰り返されると想定される．

(1) ソースコード修正

(2) クロス環境上でのビルド

(3) ROMイメージの作成

(4) ROMへの書き込み

(5) 動作確認

この課程で，ソースコードの修正は開発者の能力と修正規模に依存する時間がかかる．クロス環境上でのビルドはクロス環境のマシンのスペックに依存した時間で実行できる．多くの人が修正したコードを集積してイメージを作るときはフルビルドすることも多いであろうが，開発者個人が一部の修正を行い，その動作確認を行う目的の場合は部分的なビルドを行うであろう．そのためビルドには時間はかからない．ROMイメージの作成もそれほど時間がかかるわけではない．

ROMへの書き込みは，実機とクロス環境との間の通信速度とフラッシュメモリの消去にかかる時

間で決まる．最近の携帯電話端末は規模が巨大化しており，全体を消去して書き込むには 10 分以上もの時間を待たされることになる．

しかしながら，携帯端末上のほとんどのソフトウエアイメージは変わってないはずであり，いわゆる携帯電話の出荷後のソフトウェア更新 [11] と同程度以上の速度で書き換えができて良いはずである．しかしながら，このような開発途上のソフトウェアでは厳密なバージョン管理などができておらず新版と旧版の差分を抽出するというような作業はできないと想定するべきで，そのままではソフトウェア更新の技術を使うことはできない．

3 関連研究

フラッシュメモリへのソフトウェアの書き込みの高速化に関しては様々な研究が実施され，実際に実現利用されている．例えば，インテル社の EFP[1] 技術などもその一つであるが，主に工場で初期化するようなケースに利用する技術であり，何度も実機を書き換えながら利用するようなケースには利用できない．

ICE が接続できるボードで RAM ベースで動作させられるとしても，実機試験の段階ではこの方法も利用できない．

”携帯電話上のソフトウェアを開発環境上のソフトウェアと同じ状態にする”とは，ファイルの同期の手法と同様と考えることもできる．モバイル環境では非接続状態でのオペレーションにより，ファイルの状態が変わることが多い．これを如何に同じ状態にするかという研究が文献 [6] をはじめとして多くあり，オペレーションを再現するといった方法がある．

しかし，携帯電話上のソフトウェアの変更ではソースコードの修正のオペレーションはわかるかもしれないが，バイナリコードの修正まではバージョン管理をしていない場合には困難と考える．オペレーションが不明な場合のファイル同期に関しも多くの研究が実施され，バイナリの差分を適用する考え方で Rsync [7] や unison[8] のようなツールとして公開されているものもある．

これらはいずれもバージョン管理など必要とはせずに，ネットワークで接続された複数のホスト上のファイルのバイナリレベルの差分を取得して必要な情報のみを送ることにより転送量を少なくする技術である [9]．

同期される方のホストを携帯電話などの端末，変更済みのファイルを持つ方のホストを開発環境のパソコン，実行イメージをファイルとみなすと，これらの技術を有効に利用できると考えた．

しかし，これらの技術はファイルシステム上のファイルを前提としており，端末上のフラッシュメモリ全体を対象とた場合のメモリの使い方を想定したアルゴリズムではない．そのため，そのままでは利用できない．そこで，本論文では，この Rsync のアルゴリズムをベースにした上で，携帯端末の S/W 書き換えに適した改良方式を提案，実装して評価する．

4 Rsync と携帯端末ソフトウェアダウンロードへの適応手法

4.1 Rsync 適用基本アルゴリズム

Rsync を携帯端末のソフトウェア更新に適用する場合の原理を図 3に示す．以下にそのアルゴリズムを順を追って説明する．

(1) 携帯端末上では一定サイズの比較ブロックに実行イメージ全体を分割する．

(2) 比較ブロックごとに Fast hash と呼ぶハッシュ値を計算する．本論文での実装では rolling checksum を利用する．具体的にはチェックサムを計算する際に少しずつビットをずらして加えていく手法である（詳細なアルゴリズムは文献 [9] を参照）．1 バイトずらした位置の rolling checksum 値を計算する際にはほとんど再計算や読込を不要とする効率的な手法である．

(3) 次に Reliable hash と呼ぶ別のアルゴリズムによる hash 値の計算をする．本論文上では MD4 を利用した．

(4) 比較ブロックごとの Fast hash 値と Reliable hash 値のペアを合わせて，携帯端末から開発環境の PC に送る．

[1] Enhanced Factory Programming

(5) 開発環境上のPCではこの情報に基づき，ソフトウエアイメージの最初から比較ブロックサイズ分を比較ブロックとしてFast hashを計算する．

(6) もし，このFast hashと受信済みのFast hashコードの中で一致するものがあれば，さらにReliable hash値を計算し，さらに同一かどうか比較する．ここで，両方が一致すれば，比較ブロックの範囲とhash値の計算元となった携帯端末上の比較ブロックが内容が同じと判断する．つまり，新版のそのコードと同じ比較ブロックが携帯端末上に存在するためデータの転送が不要であると判断する．

(7) 次にもし，一致するhash値がない場合は，その比較ブロックと一致するものはなかったとし，比較する比較ブロックを1バイトずらしたところにする．図3では，開発環境パソコン上のAの部分で，比較範囲を1バイトずつずらしながら計算する．この際，rolling checksumはほとんど再計算不要で，新たな1バイトのみを計算対象に入れるだけで計算できる(詳細な証明は[9]を参照)．これらの計算を繰り返す．

(8) 携帯端末上と新版イメージで同一の情報を探し，一致していない部分だけを携帯端末に転送する．図3では，ab, c, dの部分が一致していないとするとその部分の情報と，一致比較ブロックに関してはアドレスのどこからどこに移動するかの情報のみを送る．

(9) 転送された情報は，携帯電話の差分更新の情報[5]と同じ情報量になる．差分を適用して携帯端末を新版の状態にする．

4.2 携帯端末ソフトウェアダウンロードへの適用

4.2.1 基本メモリ構成

携帯端末のフラッシュメモリ上のイメージは複数のファイルが結合される形と考えて良い．対象となるファイルは実行形式（いわゆるexeやdll形式）やデータなどが含まれ，実行形式ファイルは他ファイルへの参照が解決された形になる．PCなどと違って遅いCPUでも高速に起動するためである．図4に典型的な構成例を示す．

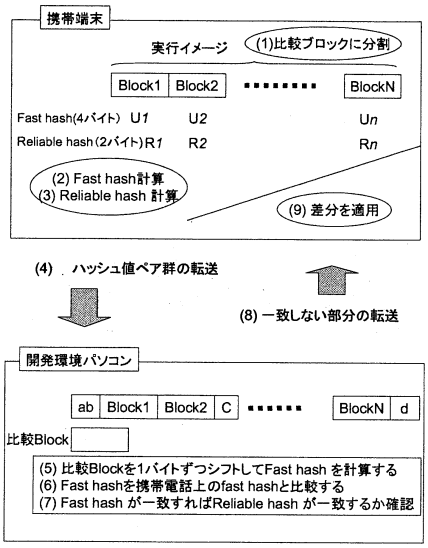

図 3: 携帯端末ソフトウェア更新へのRsync適用

4.2.2 携帯端末の制約

古いプログラムコードの入っている携帯端末に新しいプログラムコードをダウンロードする機能にRsyncを適用する場合の携帯端末の制約はRAMの容量である．オリジナルのRsyncは古いファイルに差分を適用して新しいファイルを作成するアルゴリズムである．携帯端末上のリンク済みのコードを一つのファイルと見れば良い．しかし，フラッシュメモリ上のコードは一旦RAMにコピーして

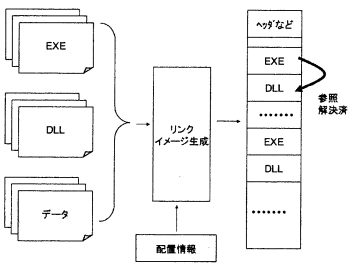

図 4: 典型的なメモリ構成

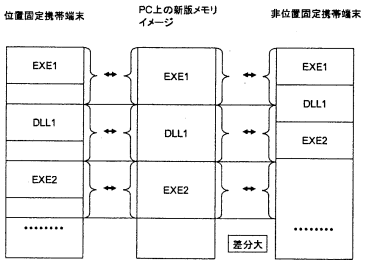

図 5: メモリ分割

から再度書き直す必要があるが，フラッシュメモリ上のコードをすべて RAM 上にコピーできるほど RAM を搭載している携帯端末は一般的でない．

そこでフラッシュメモリ上のプログラムコードを分割して RAM の容量に合わせて更新の対象とすることになる．図 5の左側に示すように分割し，分割した単位ごとに Rsync を適用する．

4.2.3 処理シーケンス

各フェーズのシーケンスを図 6に示す．最初に処理の開始要求を開発環境 PC 側から USB やシリアルケーブルで接続された携帯端末に要求する．この段階で埋め込まれた書き換えプログラムが動作する場合もあれば，開発環境の PC 側から書き換えプログラムをダウンロードして実行する場合もある．いずれにしろその後の動作シーケンスに入ることができる．

(a) 携帯電話側ではフラッシュメモリ上のデータを読み出し，ハッシュ計算する．その間に PC 側では新版のメモリイメージをロードしておく．この処理は一般にハッシュを計算する処理の方が時間がかかるため処理時間はハッシュ計算の時間で決まる．

(b) ハッシュ値の転送を行う．処理時間は転送速度と転送データ量で決まる．

(c) 開発環境 PC 上で一致比較ブロックを検索し，差分データを作成する．ほぼ PC の能力で実行時間が決まる．

(d) 差分データを転送する．処理時間は差分データの大きさで決まる．

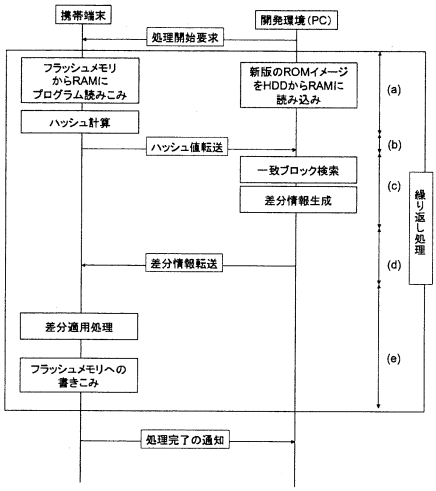

図 6: 適用処理シーケンス

(e) 差分データを展開し，差分適用処理を行ってフラッシュメモリに書き込む．差分適用時間は RAM 上での処理でありほとんど無視でき，フラッシュメモリへの書込み量で処理時間が決まる．

5 Rsync のソフトウエアダウンローダへの適用上の課題

5.1 比較ブロックサイズ

Rsync によるデータ転送量即ち，ハッシュ値と差分情報のサイズは比較ブロックのサイズに依存する．比較ブロックのサイズを大きくすれば，比較ブロック数が減るため転送するハッシュ値の数は減るが，一致する比較ブロックは少なくなり差分サイズは大きくなる．転送速度に関係するため，環境に依存して決める必要がある．

また，端末上のイメージとサーバ上のイメージの違いの特性によっても効果が変わる．よって，端末の特性に適した比較ブロックサイズの決定が重要となる．

5.2 信頼性

5.2.1 ハッシュ衝突

比較ブロックサイズはRsyncの信頼性に影響する．比較ブロックサイズを小さくすると比較ブロック数そのものが増えるため，ハッシュ衝突の可能性が増大する．ハッシュ衝突確率を$1/2^d$とするためにはハッシュ値のビット数を以下の式で示す値以上にする必要があることが既に示されている[10]．ビット数が多ければ転送量も増えるため，データ転送速度との関係で調整する必要がある．

$$Bits = log_2(n) + log_2(n/b) + d \quad (1)$$

n : ファイルサイズ
b : 比較ブロックサイズ

5.2.2 完全性

複数のハッシュ値が一致したからと言って，必ず全く同じである保証はない．そのため，信頼性をより確保するためには書き込み終了後に全体のチェックサムを計算するなどのベリファイは必須である．ベリファイにより，誤った場合の検出は可能である．しかし，その場合は全体のダウンロードが必要になるが，開発の中で，偶然そういうケースが起こる程度であれば通常のダウンロードの高速化には有効であると考える．

5.3 不具合修正の影響

不具合の修正を行うと多くの場合，改修したファイルのサイズが変わる．それによって改修していない部分のファイルの配置がずれる．ファイル間の参照解決済みでメモリに書き込まれているとこの位置ずれの影響は大きく，これはメモリの更新の高速化には好ましくない．出荷後のソフトウェア書き換えではこのようなことを防ぐ工夫[11]が提案されているが，出荷前の開発フェーズでの開発者が書き込んでは繰り返し動作確認をするような場合は手間も多くなるため適切でない．そのため，ある程度余裕を見た配置を予めしておくことにより極力影響を排除する必要がある．

また，携帯端末のRAMの容量の問題から部分的な比較となる場合に，差分抽出対象が図5右側のように適切でない部分になる可能性があるため，できる限り位置固定にしておくべきと考える．

表 1: 評価環境の性能データ

項目	性能	備考
転送 PC→端末	139KB/秒	
転送 端末→PC	70KB/秒	
圧縮 (PC)	705KB/秒	4Kbyte 単位圧縮
展開 (端末)	1.41MB/秒	4KByte 単位圧縮データの展開
フラッシュ 書込	2.13MB/秒	
フラッシュ 読込	4.27MB/秒	
チェックサム 計算	204.08MB/秒	

表 2: 評価対象データの特性

パターン名	更新消去ブロック数
A	14
B	409

6 評価

6.1 評価プラットフォームと評価対象プログラム

評価のため，典型的な携帯電話端末用のボードを試作した上で評価した．主な評価環境のスペックを表1に示す．なお，数値はカタログスペックではなく，実際にボード上で動作させた上での測定値から計算した性能値である．

また，評価対象データは，携帯電話のソフトウエアイメージを利用した．これに大して少し修正を入れた場合のデータAパターンと，このパターンで位置ずれもおきるようにしたデータBパターンを利用して評価した．いずれもフラッシュメモリのサイズ的には約90Mbyteである．

また，ページサイズは4KByte，消去ブロックは128KByteのメモリを使用した．更新パターンによる更新対象となった消去ブロックの数を表2に示す．

6.2 実行速度

6.2.1 ブロックサイズとハッシュ計算性能

図6における区間(a)の処理性能をブロックサイズを変更しながら測定した結果を図7に示す．．ハッシュの計算速度はブロックサイズを大きくすれ

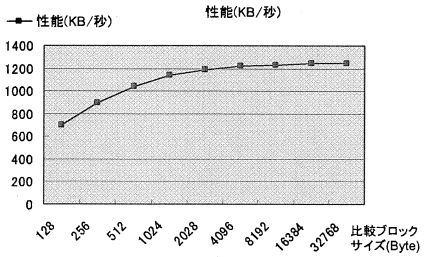

図 7: ブロックサイズとハッシュ計算性能

表 3: ハッシュ値の要するビット数

	分割単位 **				
比較ブロックサイズ *	1M	4M	16M	64M	分割なし
128	50	52	54	56	57
256	49	51	53	55	56
512	48	50	52	54	55
1024	47	49	51	53	54
2048	46	48	50	52	53
4096	45	47	49	51	52
8192	44	46	48	50	51

* サイズの単位は byte

** 単位は bit

ば良いように見えるが，4KB で一定になる．即ちページのサイズで速度が決まるため，ハッシュの計算時間よりむしろメモリのリード時間に依存しているのでブロックサイズは気にする必要がない．

6.2.2 ハッシュ値

式 (1) に基づいて，ハッシュ衝突確率が 0.001 未満になるようなハッシュ値に必要なビット数を計算した結果を表 3に示す．この結果のみ見ると，比較対象を小さくした上でブロック数を大きくすれば転送サイズを小さくし，高速化が図ることができる．

6.2.3 差分サイズの違い

更新パターン A, B に関して，分割単位を 16Mbyte とした上で，比較ブロックサイズを変更して差分サイズを調べた．結果を表 8, 表 9に示す．不一致部と差分サイズの差は小さい．ブロックサイズにほぼ比例して差分サイズは大きくなるため，データ転送量の観点からは比較ブロックは小さければ小さいほど良いことになる．

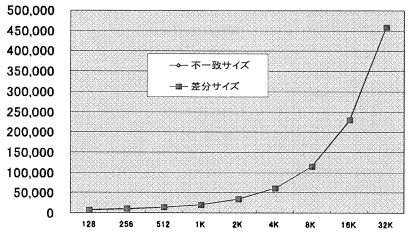

図 8: パターン A 差分サイズ

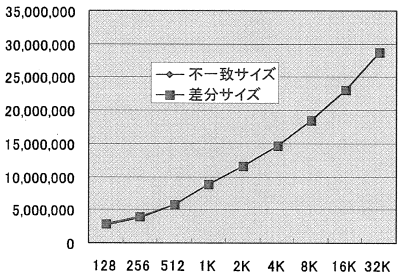

図 9: パターン B 差分サイズ

また，位置ずれの有無の差は非常に大きいこともわかる．

6.2.4 トータルダウンロード時間

これまでの測定結果から，トレードオフの関係が多いことがわかる．そこで，ブロックサイズによるトータルなダウンロード時間を表 4に示す．パターン A の場合で，比較ブロックサイズを 16K にする場合が良く，パターン B の場合では，比較ブロックサイズを 512 バイトに設定することにより，1 分 30 秒から 3 分強の間でダウンロードできることがわかる．これらの仕組みを使わずにすべてダウンロードする場合が 12 分であったため，効果があることがわかる．

このように，位置ずれがあるかないかで適切なブロックサイズが異なってくる．これは消去すべきブロック数即ち，フラッシュメモリへの書き込み時間が大きく異なることに起因する．

表 4: トータルダウンロード時間 (秒)

比較ブロックサイズ*	パターン A	パターン B
128	225.65	269.91
256	161.03	212.98
512	128.00	193.37
1,024	112.22	199.23
2,048	104.62	210.01
4,096	100.96	226.26
8,192	99.55	249.82
16,384	98.79	278.04
32,768	100.10	314.71

* サイズの単位は byte

6.3 議論

パターン A,B のいずれの場合においても効果があり，かなり早く書き換えることができることがわかった．しかし，ターゲットデバイスの仕様は様々であり，適したパラメタ設定をしなければ最適な効果はでないと考える．また，同じターゲットデバイスでも不具合の修正の仕方などでパターン A とパターン B で効果的なパラメターが変わる．

そのため，より詳細に修正パターンと書き換え量の関係，およびフラッシュメモリの書き換え速度と転送速度の関係などをパラメタライズ化し，常に適したメモリの分割方法やアラインメントの設定をするなどの仕組みが必要と考える．これらの設定を開発者が毎回設定するようでは本末転倒になりかねないため，自動的に高速に判定するような仕組みも必要と考える．

7 おわりに

携帯端末の開発フェーズ終盤での実機 H/W へのソフトウエアダウンロードの高速化手法に関して提案，評価した．提案手法は，モバイル端末などにおけるファイル同期化手法の中で有効な手法である Rsync のアルゴリズムをベースにした手法を適用することで効果があることを示した．Rsync を単純に適用するのではなく，プログラム位置の固定と適当なサイズに比較ブロックサイズを選ぶことが重要であることを示した．今後，議論で述べたような各種パラメターの関係解析および自動設定に関して検討を進めていく予定である．

参考文献

[1] 大原茂之: 日本の組込みシステム産業と技術者育成の課題 - 統計データに見る実態と見える課題, 情報処理, vol.46,no.2,pp.161-168(2005)

[2] 野中誠: 組み込みソフトウェア特性に基づくプロジェクト構築, 情報処理, vol.46,no.6,pp.684-690(2005)

[3] Cusumano M., MacCormack A., Kemerer F. Chris and et al.: Software Development Worldwide: The State of the Practice, IEEE Software, Vol.20,No.6,pp.28-34(2003)

[4] 二上貴夫: 組み込みソフトウェア開発技術：6. 組み込みソフト開発支援ツール, 情報処理, vol.45,no.7,pp.704-708(2004)

[5] 清原良三, 三井聡, 栗原まり子ほか: 携帯電話ソフトウェア更新のためのバージョン間差分表現方式, 電子情報通信学会論文誌 B, vol.J89-B,no.4,pp.478-487,2006.

[6] Balasubramaniam S. and Pierce B. : What is a File Synchronizer ?, ACM MobiCom '98, pp.98-108, 1998.

[7] Rsync: Rsync,(on line),available from <http://rsync.samba.org/> (accessd 2008-08-29).

[8] Pierce B.:Unison File Synchronizer, (on line), available from <http://www.cis.upenn.edu/ bcpierce/unison/1index.html> (accessd 2008-08-29).

[9] Tridgell A. and MacKerras P.:The rsync algorithm,Technical Report TR-CS-96-05, Australian National University, 1996.

[10] Irmak U., Mihaylov S. and Suel T.: Improved Single-Round Protocols for Remote File Synchronization,IEEE Infocom Conference2005, Vol.3, pp.1665-1676,March 2005.

[11] 清原良三, 栗原まり子, 古宮章裕ほか: 携帯電話ソフトウェアの更新方式, 情報処理学会論文誌, vol.46,no6.pp.1492-1500,2005