17. 開発ツールに関する知識 I

1. 科目の概要

ソフトウェア開発の基本的な進め方を示し、ソフトウェアを開発する際に活用する一連のツール群について、その機能と使い方を説明する。コンパイラやデバッガといった基本的なツールから、バージョン管理ツール、バグ追跡システムまで、様々なツールの活用法を解説する。

2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

習得ポイント	説　明	シラバスの対応コマ
I-17-1. ソフトウェア開発の進め方	ソフトウェア開発に必要なハードウェア、ソフトウェア、ツールの概要を示し、ソフトウェア開発プロセスにおける一連の手順を紹介する。ソフトウェア開発で注意すべきポイントや具体的な開発サイクルなど、実際のソフトウェア開発で活用できるノウハウを紹介する。	1
I-17-2. 各種のツール(コンパイラ、デバッガ、プロファイラ)	ソフトウェア開発環境の全体像を述べ、コンパイラ、デバッガ、プロファイラなど、開発環境を構成する様々な開発ツールの概要、実際の開発における役割りと位置づけ、各ツールの特徴について解説する。	2
I-17-3. 統合開発環境の様々な機能	各種のツールを効果的に組合せ、一連の開発手順をシームレスに提供する統合開発環境について解説する。統合開発環境の歴史や背景、目的、その効果などを紹介し、統合開発環境が持つ特有の機能にも言及する。	2
I-17-4. C言語によるプログラム開発方法	Linuxの開発環境におけるアプリケーションソフトウェア開発の概要として、C言語によるプログラム開発の概要について解説する。Cプログラムの基本を説明し、gccによるコンパイルの手順、生成されるオブジェクトファイルの構造、ライブラリなどについて触れる。	3
I-17-5. バージョン管理ツールの特徴、主なツールと利用方法	改変を重ねて作成するソフトウェア開発の管理に必須のツールであるバージョン管理ツールを紹介する。バージョン管理ツールの機能と特徴、バージョン管理ツールの利用方法を説明し、現在ひろく利用されているOSSのバージョン管理ツール(CVS, Subversion)を紹介する。	4
I-17-6. デバッグの基本(ブレークポイントとデータのトレース)	デバッガを利用したプログラムのデバッグ作業について説明する。データの変化をトレースしバグの原因を究明する方法の基礎、ブレークポイントの設定、データの確認方法など、デバッガを利用したデバッグ作業の基本的な手順を紹介する。	5
I-17-7. アプリ、ミドルウェア等における様々なデバッグ手法	プログラムの部分的な実行や条件を変えてのテスト、ウォッチポイントの設定など、デバッガの様々な機能を紹介する。また、アプリケーションやミドルウェアをデバッグする際に活用できる様々なデバッグ手法を説明する。	5
I-17-8. OSの動作に対する追跡手法	OS の動作を追跡する方法について説明する。通常のアプリケーションソフトウェアのデバッグとの違いについて解説し、カーネルデバッガ、デバッグライト、仮想マシン、リモートデバッグなど、OSをデバッグする様々な方法について説明する。	6
I-17-9. バグ追跡システムの目的、機能と利用方法	バグ追跡システムとは何か、その目的、機能、役割りと位置づけを解説する。継続的なデバッグの実現やバグ対応の生産性向上に寄与することを示し、ソフトウェア製品の品質向上に不可欠なツールであることを説明する。	7
I-17-10. OSSバグ追跡システム「Bugzilla」の利用	オープンソースソフトウェアとして提供されているバグ追跡システムの代表的なシステムであるBugzillaについて解説する。Bugzillaの構成、機能を示し、基本的な使い方を説明する。	7

【学習ガイダンスの使い方】

1. 「習得ポイント」により、当該科目で習得することが期待される概念・知識の全体像を把握する。
2. 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
3. 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

3. IT 知識体系との対応関係

「17. 開発ツールに関する知識Ⅰ」と IT 知識体系との対応関係は以下の通り。

科目名	基本レベル（Ⅰ）							応用レベル（Ⅱ）							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
17. 開発ツールに関する知識	<開発の流れとツール>	<ソフトウェア開発環境の概要>	<Linux開発環境におけるソフトウェアプリケーション開発の概要>	<バージョン管理ツールの活用>	<デバッガによるプログラムデバッグの環境>	<カーネルデバッガを使用したデバッグ>	<バグ追跡システムを使用したデバッグ>	<オープンソース開発ツールの種類と機能>	<統合開発環境を用いた開発手順>	<オープンソース統合開発環境の種類と特徴>	<Linux開発環境におけるソフトウェア開発ワークショップ>	<Linux開発環境におけるソフトウェア開発支援ツール概要>	<ソフトウェア開発ツールの評価>	<Eclipseを用いたソフトウェア開発>	<Eclipseを用いたソフトウェア開発ワークショップ>

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_17.pdf]

<IT 知識体系上の関連部分>

分野		科目名	1	2	3	4	5	6	7	8	9	10	11	12	13
組織関連事項と情報システム	1	IT-IAS 情報保証と情報セキュリティ	IT-IAS1. 基礎的な問題	IT-IAS2. 情報セキュリティの仕組み（対策）	IT-IAS3. 運用上の問題	IT-IAS4. ポリシー	IT-IAS5. 攻撃	IT-IAS6. 情報セキュリティ分野	IT-IAS7. フォレンジック（情報証拠）	IT-IAS8. 情報の状態	IT-IAS9. 情報セキュリティサービス	IT-IAS10. 脅威分析モデル	IT-IAS11. 脆弱性		
	2	IT-SP 社会的な観点とプロフェッショナルとしての課題	IT-SP1. プロフェッショナルとしてのコミュニケーション	IT-SP2. コンピュータの歴史	IT-SP3. コンピュータを取り巻く社会環境	IT-SP4. チームワーク	IT-SP5. 知的財産権	IT-SP6. コンピュータの法的問題	IT-SP7. 組織の中のIT	IT-SP8. プロフェッショナルとしての倫理的な問題と責任	IT-SP9. プライバシーと個人の自由				
応用技術	3	IT-IM 情報管理	IT-IM1. 情報管理の概念と基礎	IT-IM2. データベース問合わせ言語	IT-IM3. データアーキテクチャ	IT-IM4. データモデリングとデータベース設計	IT-IM5. データと情報の管理	IT-IM6. データベースの応用分野							
	4	IT-WS Webシステムとその技術	IT-WS1. Web技術	IT-WS2. 情報アーキテクチャ	IT-WS3. デジタルメディア	IT-WS4. Web開発	IT-WS5. 脆弱性	IT-WS6. ソーシャルソフトウェア							
ソフトウェアの方法と技術	5	IT-PF プログラミング基礎	IT-PF1. 基本データ構造	IT-PF2. プログラミングの基本的構成要素 [17-1-3,5]	IT-PF3. オブジェクト指向プログラミング	IT-PF4. アルゴリズムと問題解決	IT-PF5. イベント駆動プログラミング	IT-PF6. 再帰							
	6	IT-IPT 技術を統合するためのプログラミング	IT-IPT1. システム間通信	IT-IPT2. データ割り当てと交換	IT-IPT3. 統合的コーディング	IT-IPT4. スクリプティング手法	IT-IPT5. ソフトウェアセキュリティの実現	IT-IPT6. 種々の問題	IT-IPT7. プログラミング言語の概要 [17-1-3]						
	7	CE-SWE ソフトウェア工学	CE-SWE0. 歴史と概要 [17-1-1]	CE-SWE1. ソフトウェアプロセス	CE-SWE2. ソフトウェアの要求と仕様	CE-SWE3. ソフトウェアの設計	CE-SWE4. ソフトウェアのテストと検証	CE-SWE5. ソフトウェアの保守	CE-SWE6. ソフトウェア開発・保守ツールと環境	CE-SWE7. ソフトウェアプロジェクト管理	CE-SWE8. 言語翻訳	CE-SWE9. ソフトウェアのフォールトトレランス	CE-SWE10. ソフトウェアの構成管理 [17-1-4]	CE-SWE11. ソフトェアの標準化	
	8	IT-SIA システムインテグレーションとアーキテクチャ	IT-SIA1. 要求仕様	IT-SIA2. 調達/手配	IT-SIA3. インテグレーション	IT-SIA4. プロジェクト管理	IT-SIA5. テストと品質保証	IT-SIA6. 組織の特性	IT-SIA7. アーキテクチャ						
システム基盤	9	IT-NET ネットワーク	IT-NET1. ネットワークの基礎	IT-NET2. ルーティングとスイッチング	IT-NET3. 物理層	IT-NET4. セキュリティ	IT-NET5. アプリケーション分野	IT-NET6. ネットワーク管理							
	10	CE-NWK テレコミュニケーション	CE-NWK0. 歴史と概要	CE-NWK1. 通信ネットワークのアーキテクチャ	CE-NWK2. 通信ネットワークのプロトコル	CE-NWK3. LANとWAN	CE-NWK4. クライアントサーバコンピューティング	CE-NWK5. データのセキュリティと整合性	CE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング	CE-NWK7. データ通信	CE-NWK8. 組込み機器向けネットワーク	CE-NWK9. 通信技術とネットワーク概要	CE-NWK10. 性能評価	CE-NWK11. ネットワーク管理	CE-NWK12. 圧縮と伸張
			CE-NWK13. クラスタシステム	CE-NWK14. インターネットアプリケーション	CE-NWK15. 次世代インターネット	CE-NWK16. 放送									
	11	IT-PT プラットフォーム技術	IT-PT1. オペレーティングシステム	IT-PT2. アーキテクチャと機構	IT-PT3. コンピュータインフラストラクチャ	IT-PT4. デプロイメントソフトウェア	IT-PT5. ファームウェア	IT-PT6. ハードウェア							
	12	CE-OPS オペレーティングシステム	CE-OPS0. 歴史と概要	CE-OPS1. 並行性	CE-OPS2. スケジューリングとディスパッチ	CE-OPS3. メモリ管理	CE-OPS4. セキュリティと保護	CE-OPS5. ファイル管理	CE-OPS6. リアルタイムOS	CE-OPS7. OSの概要	CE-OPS8. 設計の原則	CE-OPS9. デバイス管理	CE-OPS10. システム性能評価		
コンピュータアーキテクチャとアーキテクチャ	13	CE-CAO コンピュータのアーキテクチャと構成	CE-CA00. 歴史と概要	CE-CA01. コンピュータアーキテクチャの基礎	CE-CA02. メモリシステムの構成とアーキテクチャ	CE-CA03. インタフェースと通信	CE-CA04. デバイスサブシステム	CE-CA05. CPUアーキテクチャ	CE-CA06. 性能・コスト評価	CE-CA07. 分散・並列処理	CE-CA08. コンピュータによる計算	CE-CA09. 性能向上			
複数領域にまたがるもの	14	IT-ITF IT基礎	IT-ITF1. ITの一般的なテーマ	IT-ITF2. 組織の問題	IT-ITF3. ITの歴史	IT-ITF4. IT分野（学科）とそれに関連のある分野（学科）	IT-ITF5. 応用領域	IT-ITF6. IT分野における数学と統計学の活用							
	15	CE-ESY 組込みシステム	CE-ESY0. 歴史と概要	CE-ESY1. 低電力コンピューティング	CE-ESY2. 高信頼性システムの設計	CE-ESY3. 組込み用アーキテクチャ	CE-ESY4. 開発環境	CE-ESY5. ライフサイクル	CE-ESY6. 要件分析	CE-ESY7. 仕様定義	CE-ESY8. 構造設計	CE-ESY9. テスト	CE-ESY10. プロジェクト管理	CE-ESY11. 並行設計（ハードウェア、ソフトウェア	CE-ESY12. 実装
			CE-ESY13. リアルタイムシステム設計	CE-ESY14. 組込みマイクロコントローラ	CE-ESY15. 組込みプログラム	CE-ESY16. 設計手法	CE-ESY17. ツールによるサポート	CE-ESY18. ネットワーク型組込みシステム	CE-ESY19. インタフェースシステムと混合信号システム	CE-ESY20. センサ技術	CE-ESY21. デバイスドライバ	CE-ESY22. メンテナンス	CE-ESY23. 専門システム	CE-ESY24. 信頼性とフォールトトレランス	

4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識として、開発支援ツールの具体的な OSS 実装の利用方法がある。統合開発環境、バージョン管理ツール、デバッガ、エミュレータ、欠陥追跡システムなどを内容として含む。

科目名	第1回	第2回	第3回	第4回	第5回	第6回	第7回
17.開発ツールに関する知識Ⅰ	(1)ソフトウェア開発プロセスの特徴と開発環境 (2)ソフトウェア実装 (3)アプリケーション開発	(1)ソフトウェア開発環境の必要要素 (2)統合開発環境の構築	(1)プログラムのコンパイルとリンク	(1)バージョン管理ツールの機能 (2)主なオープンソースバージョン管理ツール	(1)基本的なデバッグの方法 (2)複雑に絡み合うアプリケーションとミドルウェア間のデバッグ (3)データベース処理に関連するデバッグ	(1)カーネルデバッガによるカーネルやアプリケーションのデバッグ (2)ハードウェアエミュレータの必要性	(1)欠陥追跡システムの目的 (2)欠陥追跡システムの機能 (2)Bugzilla とは

（網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している）

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-1. ソフトウェア開発の進め方	
対応する コースウェア	第 1 回（開発の流れとツール）	

I-17-1. ソフトウェア開発の進め方

ソフトウェア開発に必要なハードウェア、ソフトウェア、ツールの概要を示し、ソフトウェア開発プロセスにおける一連の手順を紹介する。ソフトウェア開発で注意すべきポイントや具体的な開発サイクルなど、実際のソフトウェア開発で活用できるノウハウを紹介する。

【学習の要点】

＊ ソフトウェア開発に必要な環境は実行環境によって変わるので、注意が必要である。

＊ 最近は、ソフトウェア開発への要求仕様の変化を受け容れるようになってきており、仕様変更に柔軟に対応できる開発環境が求められている。

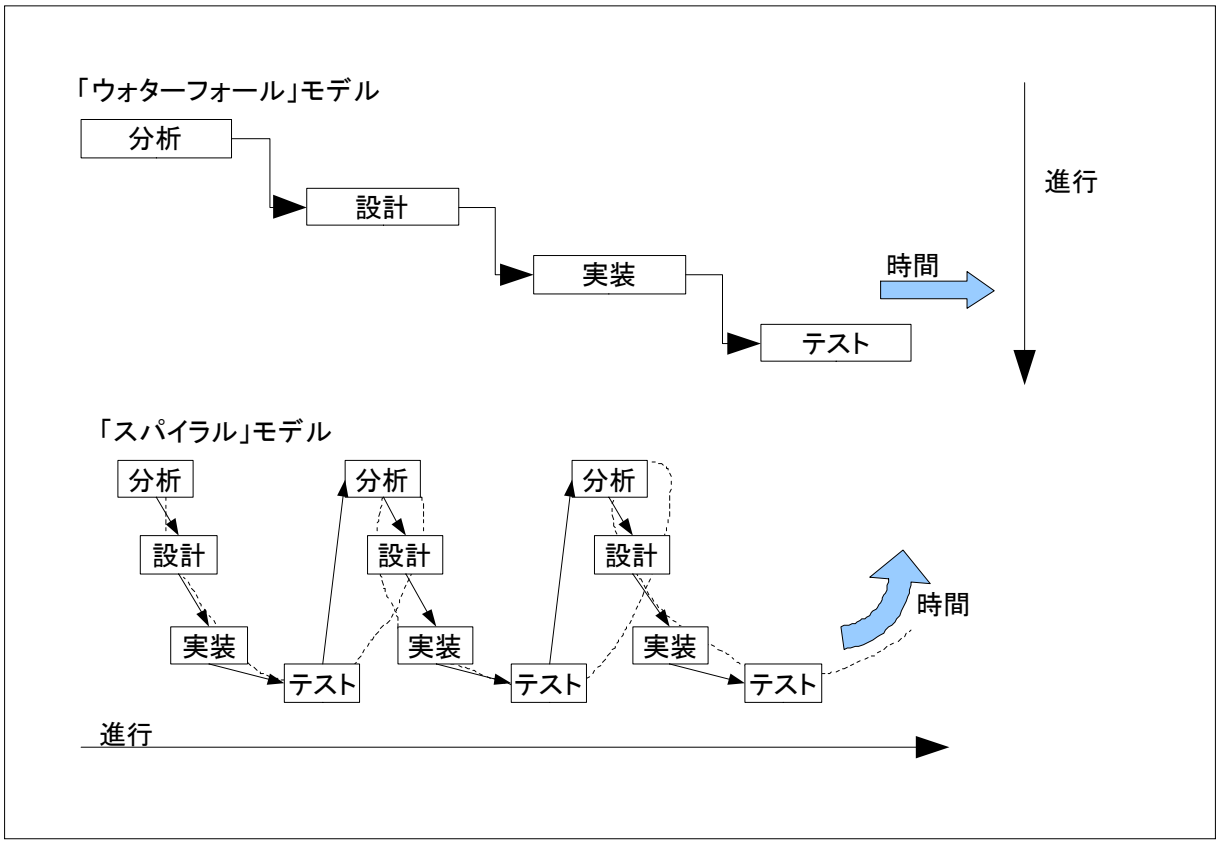

図 I-17-1.　開発プロセス概念図

【解説】

1) ソフトウェア開発に必要なハードウェア、ソフトウェア、ツール

実行環境と同様のハードウェア、ソフトウェアを用いるのが一般的である。ただし、JVM（Java Virtual Machine）などのように、環境となるハードウェアやソフトウェアの違いを吸収するレイヤーがある場合、その上位のレイヤーが同じ環境であればよい場合がある。また、実行環境以外に開発環境独自で必要なツールとして、コンパイラなど、プログラムソースコードを実行形式に変換するツールがある。

2) ソフトウェア開発プロセスにおける一連の手順

開発プロセスは一般に次のようなフェーズでの手順で考えられているが、「テストファースト」など、プロセスの順序や分類方法にはさまざまな考え方がある。

* 要求分析
  要件を明確にし、ソフトウェアが満たすべき機能を決定する。
* システム設計
  要求をどのようにして機能として実現するかを決定する。
* プログラム設計
  機能をどのようにプログラムで実現するかを決定する。
* コーディング
  設計にしたがってプログラムを作成し、コンパイルする。
* テスト
  プログラムが意図したとおり動作するかを検査する。
* 保守
  運用中に発見された不具合の修正などを行う。

3) 具体的な開発サイクルと注意すべきポイント

開発プロセスの進め方には「ウォーターフォール」モデルや「スパイラル」モデルなど、いろいろなモデルがある。ウォーターフォールモデルでは、ひとつのフェーズが完了してから次のフェーズに入り、テストが完了した段階でリリースし、保守に入る。一方、スパイラルモデルでは、大まかな要件が決まった段階でテストまで進めてプロトタイプをリリースし、小さな単位で要求分析からテストまでを繰り返しながら、要求に応じた肉付けをしていく。いずれのモデルも一長一短であり、各モデルの長所、短所に注意して開発する必要がある。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-2. 各種のツール(コンパイラ、デバッガ、プロファイラ)	
対応する コースウェア	第 2 回 (ソフトウェア開発環境の概要)	

I-17-2. 各種のツール(コンパイラ、デバッガ、プロファイラ)

ソフトウェア開発環境の全体像を述べ、コンパイラ、デバッガ、プロファイラなど、開発環境を構成する様々な開発ツールの概要、実際の開発における役割と位置づけ、各ツールの特徴について解説する。

【学習の要点】

＊ ソフトウェア開発においては、さまざまなツールの利用が欠かせないものとなっている。

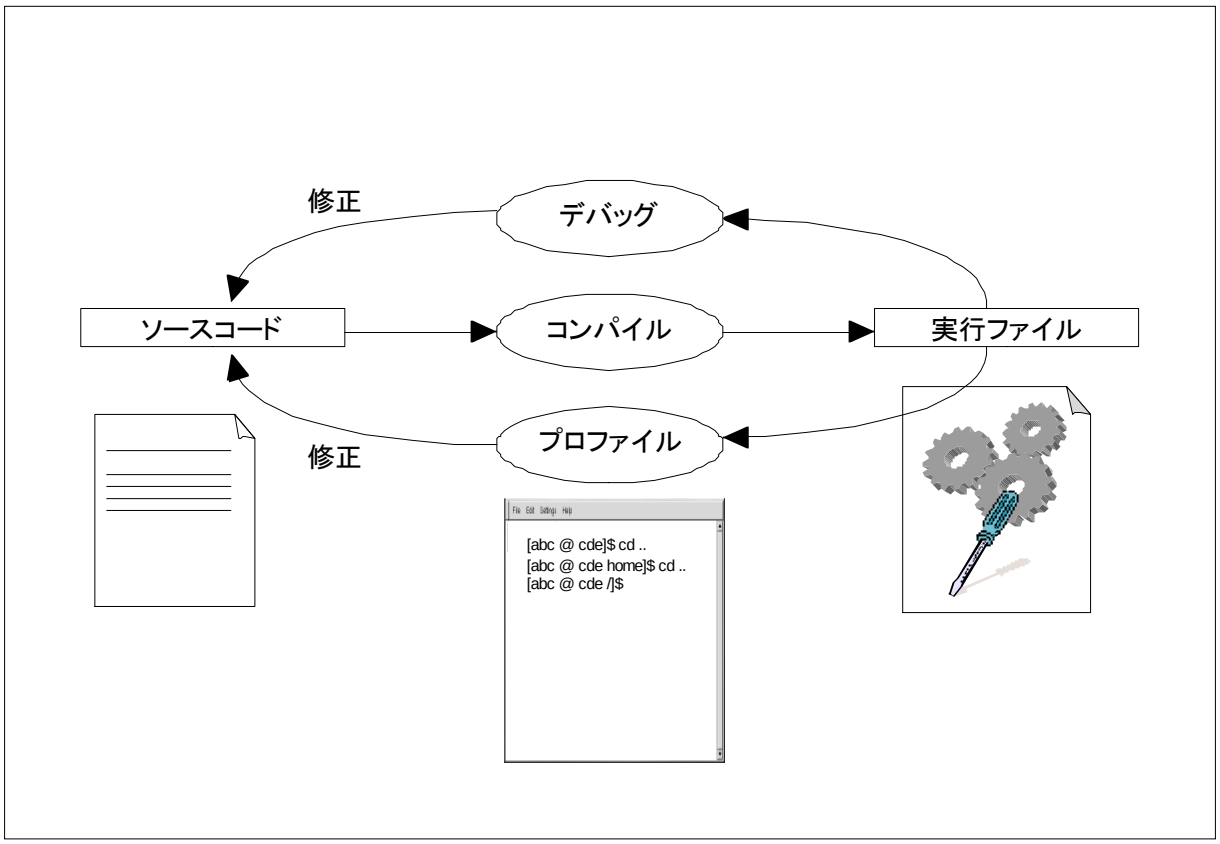

図 I-17-2.　ソフトウェア開発概念図

【解説】

1) ソフトウェア開発環境を取り巻く様々な作業と役割

ソフトウェア開発のコーディング～テストのフェーズでは、プログラムリリースに伴う様々な作業がある。

* アセンブル
  アセンブリから機械語への変換。アセンブルを行うツールをアセンブラという。
* コンパイル
  高水準言語のソースコードから機械語またはアセンブリへの変換。コンパイルを行うツールをコンパイラという。
* プリプロセス
  ソースコードをコンパイルする前の事前処理。プリプロセスを行うツールをプリプロセッサという。
* リンク
  ライブラリ(部品化されたプログラム)の結合。リンクを行うツールをリンカという。
* ビルド
  コンパイルやリンクを行って、実行可能ファイルを作成する作業。
* デバッグ
  バグの原因を突きとめバグを取り除く作業。デバッグを「支援」するツールをデバッガという。
* プロファイル
  プログラム実行時の情報の収集・解析。プロファイルを行うツールをプロファイラという。
* トレース
  プログラムの処理の実行順序やデータの監視。トレースを行うツールをトレーサという。
* ロード
  実行可能ファイルをメモリに読み込んで実行する作業。ロードを行うツールをローダという。
* その他
  ソースコードの差分の取得/適用など。

2) 各種のツール

* gcc
  GNU プロジェクトによって開発されているコンパイラ。当初は C 言語専用のコンパイラだったが、現在は C 以外に C++、Ada、Fortran、Java など様々な言語向けのコンパイラのセットになっている。コンパイルだけでなく、プリプロセス、リンカ ld を利用したリンクなども行えるようになっている。
* make
  UNIX 系 OS で古くから利用されている、ビルド処理を自動化するツール。Makefile(または makefile)に一連の処理を記述する。ビルドだけでなく、たとえば「make install」でソフトウェアのインストールが行える。
* configure
  Makefile を作成するためのシェルスクリプト。「./configure」「make」「make install」の 3 つのコマンド実行だけで、簡単にソフトウェアのビルドとインストールが出来るように作成されている。OS などの環境に依存する部分も吸収されるように作られていることが多い。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-3. 統合開発環境の様々な機能	
対応する コースウェア	第 2 回（ソフトウェア開発環境の概要）	

I-17-3. 統合開発環境の様々な機能

各種のツールを効果的に組合せ、一連の開発手順をシームレスに提供する統合開発環境について解説する。統合開発環境の歴史や背景、目的、その効果などを紹介し、統合開発環境が持つ特有の機能にも言及する。

【学習の要点】

＊ 統合開発環境では、コンパイルから実行、ファイル管理、デバッグまでをひとつのシステムで行うことが出来る。

＊ 統合開発環境には複数の言語に対応しているものもあり、異なる言語のソフトウェアを同様の環境で開発することが可能である。

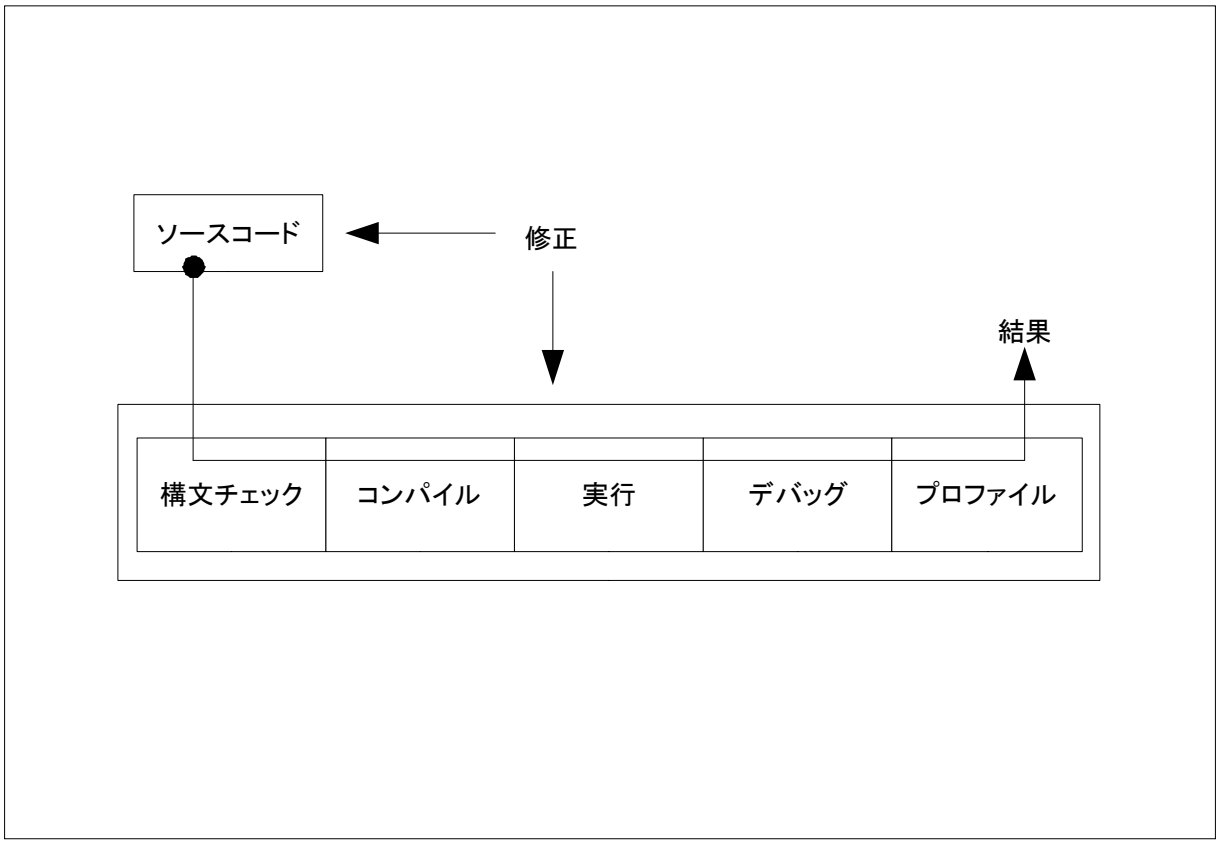

図 I-17-3.　統合開発環境概念図

【解説】

1) 統合開発環境の歴史と背景

1983 年、Borland 社は、当時の OS である CP/M 用に、Pascal 言語の開発環境「Turbo Pascal」を発売した。Turbo Pascal は、エディタ、コンパイラ、リンカを統合しており、パーソナルコンピュータ向けの統合開発環境に分類される、最も初期の製品のひとつである。その後の OS である MS-DOS、Windows においても、商用の統合開発環境は大きなシェアを占めてきた。一方、UNIX 系の OS では、高機能なエディタである Emacs の文化が根付いており、UNIX や C 言語の思想とのギャップもあり、統合開発環境が浸透してこなかったのではないかと考えられる。しかし、Java 言語の普及と、2001 年にオープンソース化された統合開発環境 Eclipse の普及によって、UNIX 系の OS にも統合開発環境が広がりを見せている。

2) 統合開発環境の目的(期待される効果)、特徴

統合開発環境は、主として生産性と品質の向上を目的としており、以下のような特徴がある。

* 多くのツールを GUI から利用できる
* 巨大なソフトウェアを開発する場合の開発者への負担が小さい
* 個々のツールの連携がとられている
* 個々のツールの機能を超えた、統合開発環境特有の機能を利用できる

3) 統合開発環境が持つ特有の機能

* プロジェクト管理
  開発で作成される複数のファイルをまとめて「プロジェクト」として一括管理でき、複数ファイルにまたがった依存関係の把握などが容易になる。
* 透過的なバージョン管理
  バージョン管理ソフトを個別に利用するのに比べ、扱いが容易になる。
* GUI アプリケーション開発
  GUI を使って GUI を開発する、つまり直観的な GUI 開発が可能となる。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-4. C 言語によるプログラム開発方法	
対応する コースウェア	第 3 回（Linux 開発環境におけるソフトウェアアプリケーション開発の概要）	

I-17-4. C 言語によるプログラム開発方法

Linux の開発環境におけるアプリケーションソフトウェア開発の概要として、C 言語によるプログラム開発の概要について解説する。C プログラムの基本を説明し、gcc によるコンパイルの手順、生成されるオブジェクトファイルの構造、ライブラリなどについて触れる。

【学習の要点】

＊ Linux カーネルのほとんどは C 言語で開発されている。

＊ C 言語は C++、C#、Java など、多くの後発言語に影響を与えている。

＊ gcc は C コンパイラ等を含むパッケージで、ほとんどの UNIX 系 OS に移植されている。

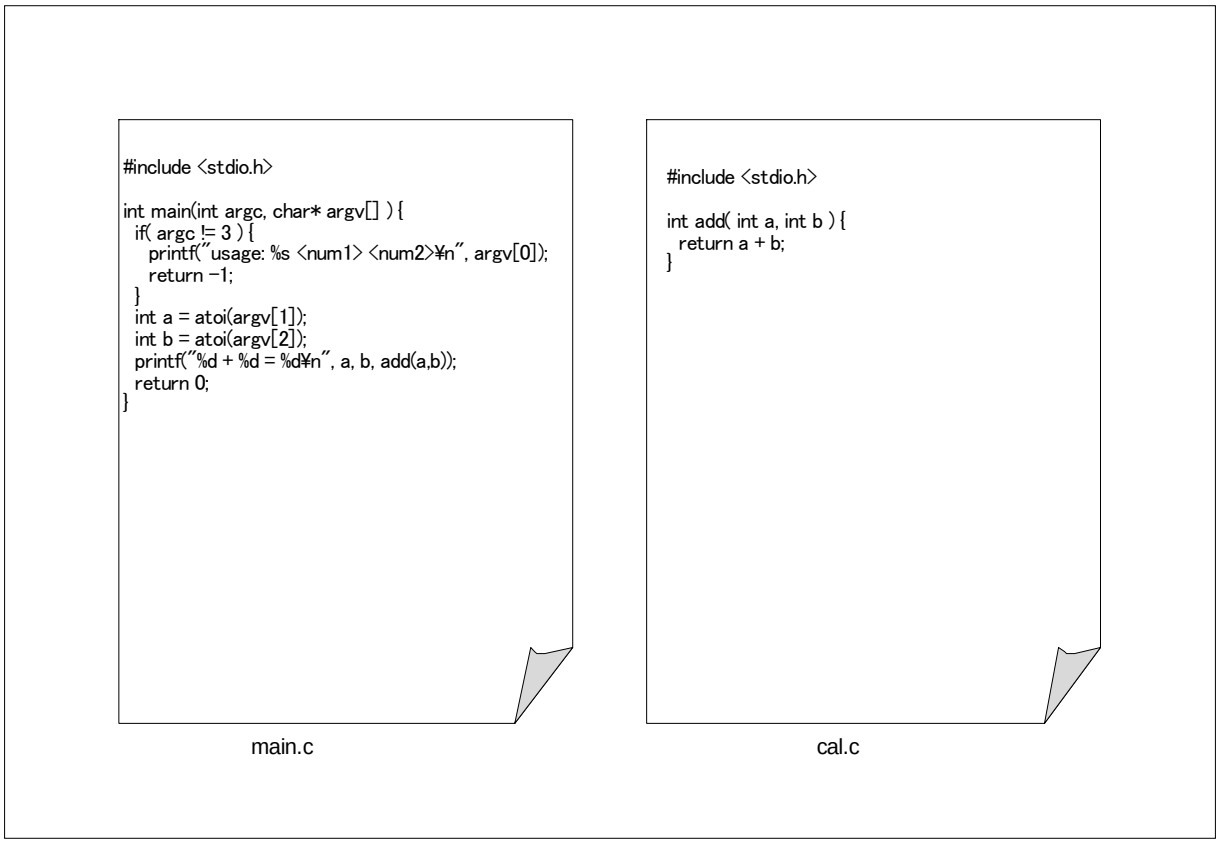

図 I-17-4.　C 言語のサンプルコード

【解説】

1) C プログラムの基本

C のプログラムは、関数と呼ばれる一連の処理のまとまりを組み合わせて作成される。C では最初に main()という関数が呼び出されることになっている。関数は、個別に処理を定義するか、外部のライブラリを参照して利用する。

2) gcc によるコンパイルとリンクの手順

例えば、ソースファイルが main.c、calc.c の 2 つである場合、以下のような手順を踏む。

* 各ソースファイルのコンパイル
  gcc –c [ファイル名]（-c を指定した場合はコンパイルまではするが、リンクはしない）
  この操作により、オブジェクトファイル main.o, calc.o が生成される。
* リンク（実行可能ファイル add が作成される）
  2 つのオブジェクトファイルをリンクして実行形式を作成する。
  gcc [オブジェクトファイルのリスト(*.o)] -o [実行形式名]

3) 生成されるオブジェクトファイルの構造

コンパイルによって生成されるオブジェクトファイルは、いくつかのセクションによって構成されている。主なセクションは以下の通りである。
[Inside Linux Software (佐藤竜一著 翔泳社 2007 年発行) p.41 より引用]

* text
  実際の機械語コードが格納されるセクション。ロード後は読み込み専用となる
* data
  事前に確保済みのデータが格納されるセクション。ロード後は読み書き共に可能となる
* bss
  事前に確保済みだが、初期値が設定されていないデータ用のセクション。オブジェクトファイル内に bss 用の領域は存在しないが、ロード後には必要な領域が割り当てらる
* rodata セクション
  読み込み専用のデータが格納されるセクション
* comment
  オブジェクトファイルの説明用文字列が格納されるセクション。GCC は、ここに GCC のバージョンなどを書き込む

4) ライブラリ

ライブラリを参照することで、個別に関数を定義しなくとも、ライブラリで定義された関数を呼びだすことができるようになる。ライブラリには以下のような種類がある。

* 静的ライブラリ（.a）
  中にオブジェクトファイルが格納されているアーカイブファイル。
* 共有ライブラリ（.so）
  ロード時に動的にリンクされる。実行可能ファイルには動的リンクに必要な情報が書かれる。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-5. バージョン管理ツールの特徴、主なツールと利用方法	
対応する コースウェア	第 4 回（バージョン管理ツールの活用）	

I-17-5. バージョン管理ツールの特徴、主なツールと利用方法

改変を重ねて作成するソフトウェア開発の管理に必須のツールであるバージョン管理ツールを紹介する。バージョン管理ツールの機能と特徴、バージョン管理ツールの利用方法を説明し、現在ひろく利用されている OSS のバージョン管理ツール(CVS、Subversion)を紹介する。

【学習の要点】

＊ システム開発の大規模化に伴って、バージョン管理ツールは欠かせないものとなっている。

＊ CVS は OSS のバージョン管理ツールとして歴史があり、安定した GUI クライアントが利用できる。

＊ Subversion は CVS の改良版として開発され、ディレクトリ構成の変更などに対応している。

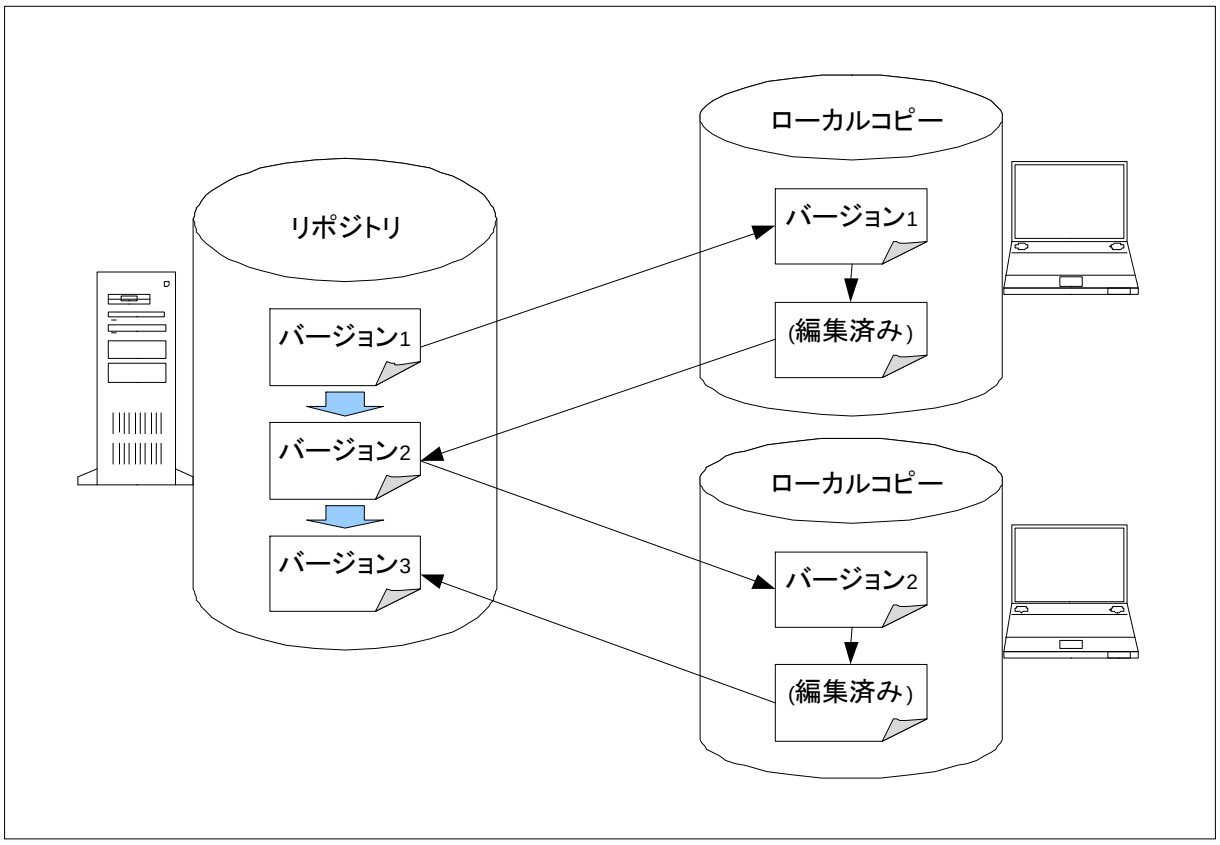

図 I-17-5.　バージョン管理概念図

【解説】

1) バージョン管理ツールの機能と特徴

バージョン管理ツールとは、ファイルの変更履歴を統一的に管理するためのツールである。管理対象となるファイルの変更情報をリポジトリと呼ばれるデータベースに格納する。リポジトリに登録されたファイルに対し、行われた変更を履歴として保存し、履歴を遡って任意の時点での状態に戻す事ができる。

2) バージョン管理ツールの主な利用方法

* リポジトリの作成
  一括管理したい単位でリポジトリを作成する。
* ローカルコピーの作成
  リポジトリの管理対象ファイルのコピーを作成する。
* ローカルコピーのアップデート
  リポジトリから情報を取得し、ローカルコピーを最新または指定したリビジョンの状態にする。
* 変更の適用
  ローカルコピーに加えた変更をリポジトリに登録する。
* 差分の確認
  ファイルに加えた変更の差分を確認する。

3) CVS

CVS は、OSS のバージョン管理ツールで、広く利用されている。CVS の特徴は以下のようなものである。

* ファイル単位にリビジョン番号が振られて管理される。
* 利用実績が豊富である。
* 統合開発環境など、他のソフトウェアとの機能連携が充実している。
* 動作の安定した GUI のクライアントツールが利用できる。

4) Subversion

Subversion は、CVS に代わるものとして開発されている OSS のバージョン管理ツールである。CVS に比べて以下のような特徴があり、普及が期待されている。

* ファイルだけでなくディレクトリのバージョン管理が可能である。
* ファイルのコピー/移動/名称変更/消去といった操作への対応が強化されている。
* ネットワーク利用前提で設計されている。
* バイナリファイルが効率的に扱える。
* ソースツリー全体に対してリビジョン番号が振られる。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-6. デバッグの基本(ブレークポイントとデータのトレース)	
対応する コースウェア	第 5 回 (デバッガによるプログラムデバッグの環境)	

I-17-6. デバッグの基本(ブレークポイントとデータのトレース)

デバッガを利用したプログラムのデバッグ作業について説明する。データの変化をトレースしバグの原因を究明する方法の基礎、ブレークポイントの設定、データの確認方法など、デバッガを利用したデバッグ作業の基本的な手順を紹介する。

【学習の要点】

* プログラムのデバッグを行う際、デバッガを用いないと、デバッグに費やす時間が大幅に増えてしまう。
* ブレークポイントを適切に設定することで、バグの発生箇所を早期に見出すことができる。

```
[user@term02 c]$ gdb sample
GNU gdb Red Hat Linux (6.5-25.el5rh)
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db library
"/lib/i686/nosegneg/libthread_db.so.1".

(gdb) break add
Breakpoint 1 at 0x80483b7: file calc.c, line 4.
(gdb) r
Starting program: /home/user/work/c/sample
usage: /home/user/work/c/sample <num1> <num2>

Program exited with code 0377.
(gdb) backtrace
No stack.
(gdb) r 3 4
Starting program: /home/user/work/c/sample 3 4

Breakpoint 1, add (a=3, b=4) at calc.c:4
4         return a + b;
(gdb) backtrace
#0  add (a=3, b=4) at calc.c:4
#1  0x0804843b in main (argc=3, argv=0xbfc2d454) at main.c:10
(gdb) cont
Continuing.
3 + 4 = 7

Program exited normally.
(gdb) quit
[user@term02 c]$
```

図 I-17-6.　デバッガの実行例

【解説】

1) デバッガとは

デバッガとは、プログラムのデバッグを支援するツールである。デバッガを用いずにプログラムのデバッグを行う場合、ソースコードを注意深く読み返し、意図しない値が入っている疑いのある変数があれば、その変数を出力するコードを追記し、再度コンパイルして実行することで値を見る、といった作業を繰り返していくことになる。しかし、大規模なプログラムのソースコードから疑わしい変数を見極めるのは難しいことが多く、結果として前述した作業を何度繰り返しても問題のある個所が見つからず、バグの特定に至る前に大幅に時間を費やしてしまうこともある。

このような状況では、デバッガを活用することでデバッグ作業を効率化することができる。プログラムコンパイル時のオプションでデバッグ情報を埋め込むように指定し、デバッガからそのプログラムを起動することで、任意の個所でプログラム実行を中断し、そのタイミングでの変数の値を表示させるといったことができる。

なお、デバッガを用いてプログラムのデバッグを行う場合に、デバッグ対象のプログラム本体をデバッギと呼ぶ。

2) デバッガの基本機能

一般的にデバッガは以下のような機能を持つ。

* 変数の表示
  登録した変数やオブジェクトのプロパティ、メモリ、レジスタ等が保持する値やデータ型などの情報を出力する機能。実行途中のデータの状態を確認することができる。
* ブレークポイント
  ソースコード中の任意の箇所で実行を中断する機能。ブレークポイントを設定してプログラムを実行すると、処理がブレークポイントの設定箇所に達した時点で一時停止される。プログラム実行の途中の位置での状況を確認することができる。
* ステップ実行
  プログラムを 1 ステップ(1 命令あるいはソースコードの 1 行)ずつ一時停止を繰り返しながら実行する機能。プログラムをステップを追いながら途中の位置での状況を確認することができる。
* ステップオーバー
  ステップ実行中に、呼び出される関数等の内部処理をスキップする機能
* ステップイン
  ステップ実行中に、呼び出される関数等の内部処理を実行する機能
* ステップアウト
  ステップインで中にはいった関数の処理が終了するまで、ステップ実行をキャンセルして一度に処理する機能。

3) デバッガを利用したプログラムのデバッグ作業

デバッガを利用したデバッグ作業の基本手順を以下に示す。

* 問題のある個所より手前でブレークポイントを設定してプログラムを中断させる。全く見当がつかない場合は、一行目に設定する。
* 変数の値を表示して監視しつつ、疑わしいステップは 1 ステップずつ、問題ない箇所はステップオーバーでスキップし、効率的にプログラムの挙動を観察する。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-7. アプリ、ミドルウェア等における様々なデバッグ手法	
対応する コースウェア	第 5 回（デバッガによるプログラムデバッグの環境）	

I-17-7. アプリ、ミドルウェア等における様々なデバッグ手法

プログラムの部分的な実行や条件を変えてのテスト、ウォッチポイントの設定など、デバッガの様々な機能を紹介する。また、アプリケーションやミドルウェアをデバッグする際に活用できる様々なデバッグ手法を説明する。

【学習の要点】

* デバッガの様々な機能を使いこなすことで、プログラムの効率的なデバッグができるようになる。
* 様々なデバッグ手法を試すことで、通常のデバッガでは発見が困難なバグも容易に発見できる場合がある。

```
[user@term02 c]$ gdb sample
GNU gdb Red Hat Linux (6.5-25.el5rh)
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db library
"/lib/i686/nosegneg/libthread_db.so.1".

(gdb) break 4
Breakpoint 1 at 0x80483d4: file main.c, line 4.
(gdb) r 4 5
Starting program: /home/user/work/c/sample 4 5

Breakpoint 1, main (argc=3, argv=0xbfa51a84) at main.c:4
4         if( argc != 3 ) {
(gdb) watch a
Hardware watchpoint 2: a
(gdb) cont
Continuing.
Hardware watchpoint 2: a

Old value = 11612148
New value = 4
main (argc=3, argv=0xbfa51a84) at main.c:9
9         int b = atoi(argv[2]);
(gdb)
```

図 I-17-7.　ウォッチポイントの利用例

【解説】

1) 便利なデバッガの拡張機能

デバッガの拡張機能を紹介する。これらをうまく使いこなすことによりデバッグ作業を大幅に効率化することができる。

＊ ウォッチポイント

ウォッチポイントは、監視対象の変数を登録し、その変数が変更された時点で実行を中断する機能である。ソースコードの特定行で中断させるブレークポイントとうまく使い分けることで、より柔軟にプログラムを操作できる。

＊ 変数の常時監視機能

GUI を持つデバッガによくみられるものとして、登録した変数を、デバッグ中に常時表示しておき、変数の値が変更されると強調表示される、などの機能がある。デバッガによっては、こちらの機能をウォッチポイントと呼ぶこともある。変数に意図しない値が混入するバグを追跡する際、この機能を使って変数を監視しながらステップ実行することで、問題の個所を素早く特定することができる。

＊ 分散デバッグ機能

プログラムの実行環境によっては、OS の支援を得て実行中のプロセスへデバッガを接続したり、デバッグ専用のプロトコルを用いて、デバッガをネットワーク経由でプログラムへ接続させ、遠隔操作したりすることができる。これを分散デバッグと呼ぶ。特定の環境でのみ発生するバグを追跡する際、実環境で問題が再現したときにその場でデバッガを接続し、状態を解析するなどの用途が考えられる。

＊ コアファイル解析機能

プログラムが致命的なエラーで終了した場合に、core と呼ばれるファイルに、終了時点のメモリ内容等が出力される。コアファイルはそのままでは読むことができないが、これをデバッガに読み込ませることにより、エラーが発生した箇所やその際のメモリ内容等を参照することができ、エラー発生の要因を見つけることに役立てることができる。

2) その他のデバッグ作業に役立つツール

デバッグ対象のプログラムは様々であり、必ずしもデバッガを使用してデバッグすることが最善であるとは限らない。ここでは、デバッガでは追跡しづらいデバッグ作業に役立つツールをいくつか紹介する。

＊ tcpdump, ethereal(wireshark)

tcpdump、ethereal は Linux などで使用できるパケットキャプチャツールである。ネットワークプログラムのデバッグを行う場合、これらのパケットキャプチャツールを用いてパケットの中身を解析し、意図したデータが送受信できているかを調べることによって、デバッグ作業を効率化できる場合がある。

＊ strace, ltrace

strace は、特定のプログラムが呼び出すシステムコールを監視し、ログ出力させることができ、ltrace は、特定のプログラムの共有ライブラリ関数呼び出しを監視し、ログ出力させることができる。どちらも、既に稼働中のプロセスにプロセス ID を指定して接続することもできる。大規模なプログラムをデバッグする場合に、これらのツールで予め大まかな処理内容を把握することで、デバッグ作業を効率化できる場合がある。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-8. OS の動作に対する追跡手法	
対応する コースウェア	第 6 回（カーネルデバッガを使用したデバッグ）	

I-17-8. OS の動作に対する追跡手法

OS の動作を追跡する方法について説明する。通常のアプリケーションソフトウェアのデバッグとの違いについて解説し、カーネルデバッガ、デバッグライト、仮想マシン、リモートデバッグなど、OS をデバッグする様々な方法について説明する。

【学習の要点】

* デバッグライト（デバッグメッセージをコードに埋め込む手法）のみで OS のデバッグを行うと、OS の再起動が都度必要となり、非常に効率が悪い。
* カーネルデバッガや仮想マシン、リモートデバッグを活用することで、OS のデバッグをより効率的に行うことができる。
* OS の動作追跡手法を理解することで、OS とアプリケーションとの障害切り分けができるようになる。

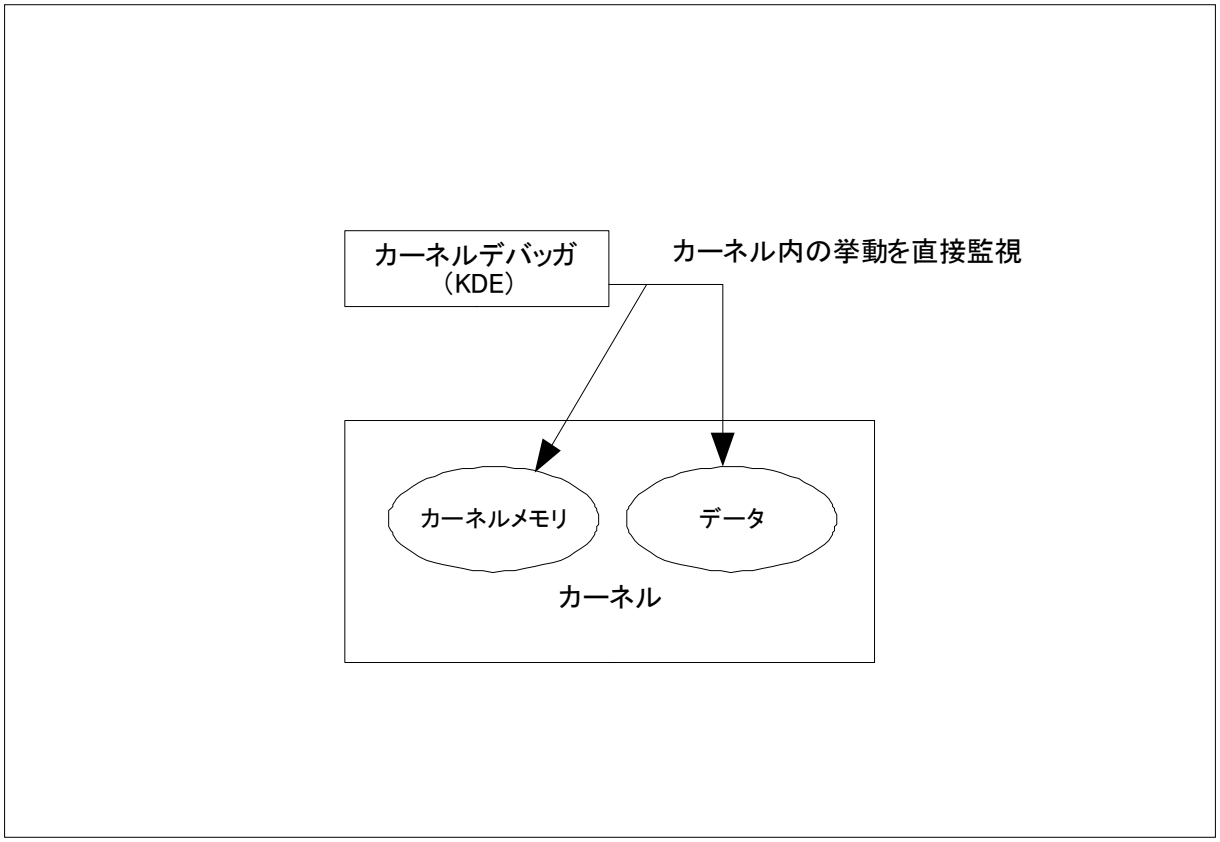

図 I-17-8.　カーネルデバッガ概念図

【解説】

1) OS のデバッグについて

通常のアプリケーションをデバッグする場合、実際に実行してみて、エラーの内容を確認しながらソースコードを読み込み、コードを書き換えてコンパイルし、再度実行してみる、といったことを繰り返して行い、デバッグ作業を進めていく。しかし、OS のデバッグを行う場合、OS がエラー終了してしまうと、その後そのままエラー内容を確認したり、コードを書き換えてコンパイルすることはできなくなってしまう。デバッグ作業の続きを行うためには、マシンを再起動し、OS を起動しなおさなければならない。しかし、再起動してしまえば、エラー終了した際のメッセージも画面から消えてしまうため、他のマシンでメモをとったり、紙に書き留めたりする必要があり、非常に手間がかかる。またコードを書き換えて再度実行する場合にも、OS を再起動しなければならず、やはり非常に手間がかかる。
これらの状況を改善する方法として、カーネルデバッガ、仮想マシン、リモートデバッグを用いる方法がある。

2) カーネルデバッガ

カーネルデバッガとは、OS カーネルのデバッグ作業を支援するツールである。カーネルプログラム上に埋め込むことで、実行ステップや関数値の参照が可能である。
カーネルデバッグには、デバッグライトと呼ばれる比較的簡単な方法があるが、デバッグライトでは、OS の再起動が都度必要となり、手間がかかる。Linux では、カーネルデバッガ(KDB)が開発されており、KDB を利用することで、現在実行中のカーネルのカーネル・メモリやデータ構造に直接アクセスすることができる。
KDB では主に、メモリを扱うコマンド、レジスタを扱うコマンド、例外を扱うコマンド、ブレークポイントを扱うコマンド群を利用してデバッグを行う。

3) 仮想マシン

仮想化とは、1 台のマシンで動作させるソフトウェア上で、複数の仮想的なマシンを構築する技術であり、この仮想的に構築されたマシンを仮想マシンと呼ぶ。デバッグ対象の OS を仮想マシンとして動作させることで、デバッグ環境を確保しつつ、一台の物理マシンで OS のデバッグを行うことができる。

4) リモートデバッグ

リモートデバッグとは、デバッグ対象のプログラムを別マシンで動作させ、ネットワーク越しにデバッガを接続して行うデバッグのことである。組み込みシステムにおいて、デバッグ対象のプログラムが動作する環境で直接デバッガを起動するのが難しい場合に有用である。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-9. バグ追跡システムの目的、機能と利用方法	
対応する コースウェア	第 7 回（バグ追跡システムを使用したデバッグ）	

I-17-9. バグ追跡システムの目的、機能と利用方法

バグ追跡システムとは何か、その目的、機能、役割りと位置づけを解説する。継続的なデバッグの実現やバグ対応の生産性向上に寄与することを示し、ソフトウェア製品の品質向上に不可欠なツールであることを説明する。

【学習の要点】

＊ ソフトウェア開発プロジェクトにバグ追跡システムを利用することで、ソフトウェアの品質を向上させることができる。

＊ バグ追跡システムを正しく運用することで、開発効率が向上し、品質もさらに上げることができる。

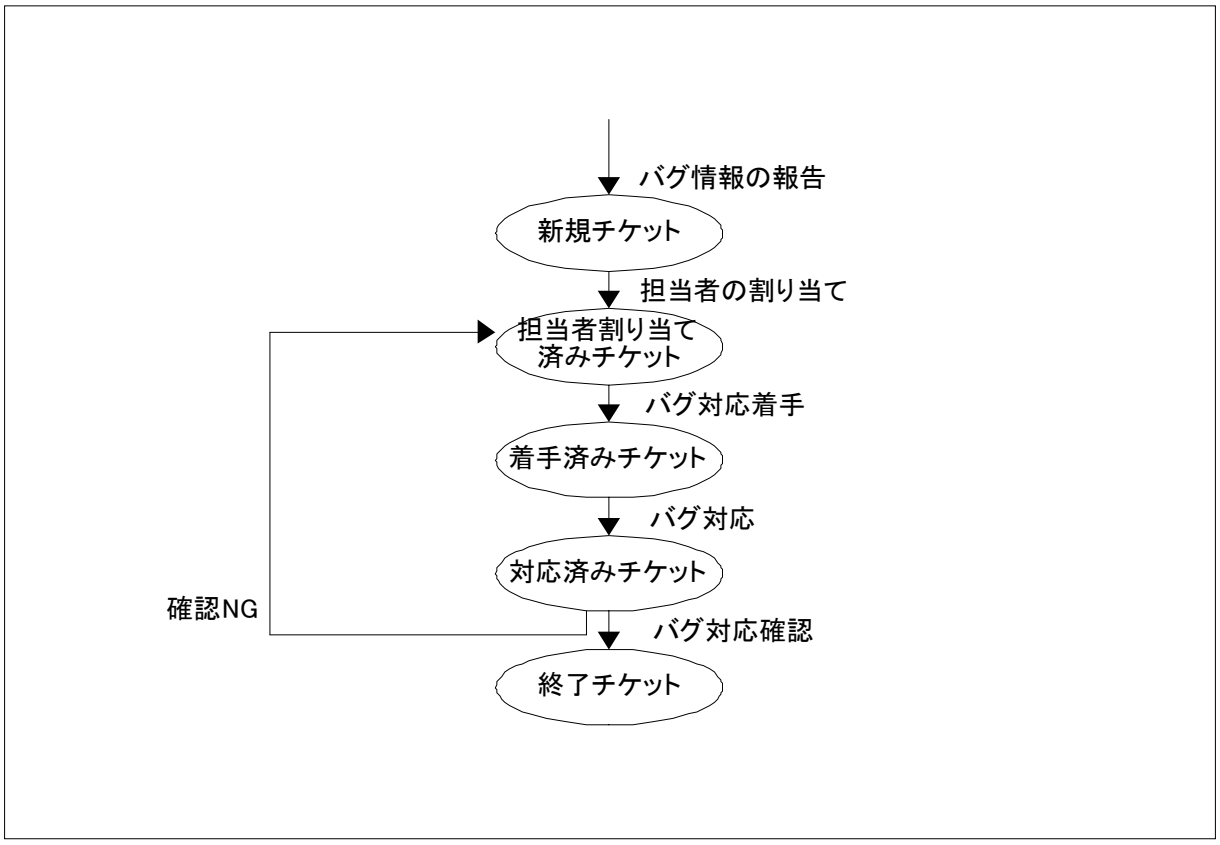

図 I-17-9.　バグ追跡システムのワークフロー概念図

【解説】

1) バグ追跡システムとは

バグ追跡システム(Bug Tracking System, BTS)とは、システムのバグ情報を登録し、その情報に随時報告を追加していくことで、バグ情報やバグの対応状況を管理するシステムである。

2) バグ追跡システム登場の背景

ソフトウェア開発の大規模化、複雑化や、アジャイル開発が進むにつれ、品質確保がますます困難になってきている。特に世界中の多数の開発者が参加するOSSのプロジェクトにおいては、各開発者が遠く離れた場所にいるため、従来型のバグ管理が適用できない。OSS プロジェクトを成功に導くためには、ネットワーク越しでのバグ情報の共有が必須である。このような背景に基づき、バグ追跡システムが登場した。

3) バグ追跡システムのワークフロー

以下に、1つのバグが報告されて対応が完了するまでのフローを示す。バグ追跡システムの種類によって、この流れには若干の違いがあるが、大まかには多くのバグ追跡システムは以下のようなフローになっている。

* バグ情報の報告
  テストメンバーがバグを発見した場合、その現象、出現条件、再現手順などを細かく記載したチケットを登録する。
* 担当者の割り当て
  テスト管理者は登録されたチケットを確認し、チケット上で、その内容に応じて実装メンバーを割り当てる。
* バグ対応着手
  実装メンバーは登録されたチケットを確認し、作業着手の旨を報告する。
* バグ対応
  バグが確認できた場合は、修正のうえその旨報告する。バグが確認できなかった場合は、「修正不要」「再現せず」などといった状況を報告し、チケット担当をテストメンバーに割り当てる。
* バグ対応確認
  テストメンバーは再テストを行い、バグ修正が確認できた場合、バグ修正不要と判断できた場合は、チケットをクローズ状態にする。

4) OSS の開発プロジェクトにおけるバグ報告

OSS は、世界中に分散した個人の開発者がボランティアで開発しているものである。利用者がそのソフトウェアを利用して何らかの不具合を発見した場合、プロジェクトで運営されているバグ追跡システムへ不具合の内容を報告し、バグ修正に役立てることによって、そのソフトウェアの品質向上のために貢献することができる。このような開発者のみならず、利用者個人の一つ一つの貢献の積み重ねが OSS の品質向上に寄与している。

スキル区分	OSS モデルカリキュラムの科目	レベル
開発体系分野	17　開発ツールに関する知識 I	基本
習得ポイント	I-17-10. OSS バグ追跡システム「Bugzilla」の利用	
対応する コースウェア	第 7 回（バグ追跡システムを使用したデバッグ）	

I-17-10. OSS バグ追跡システム「Bugzilla」の利用

オープンソースソフトウェアとして提供されているバグ追跡システムの代表的なシステムである Bugzilla について解説する。Bugzilla の構成、機能を示し、基本的な使い方を説明する。

【学習の要点】

＊ Bugzilla は Web ブラウザでアクセス可能なバグ追跡システムの OSS 実装であり、広く利用されている。

＊ Bugzilla では詳細な条件に対応した検索機能が用意されており、多くの項目を指定して検索することが可能である。

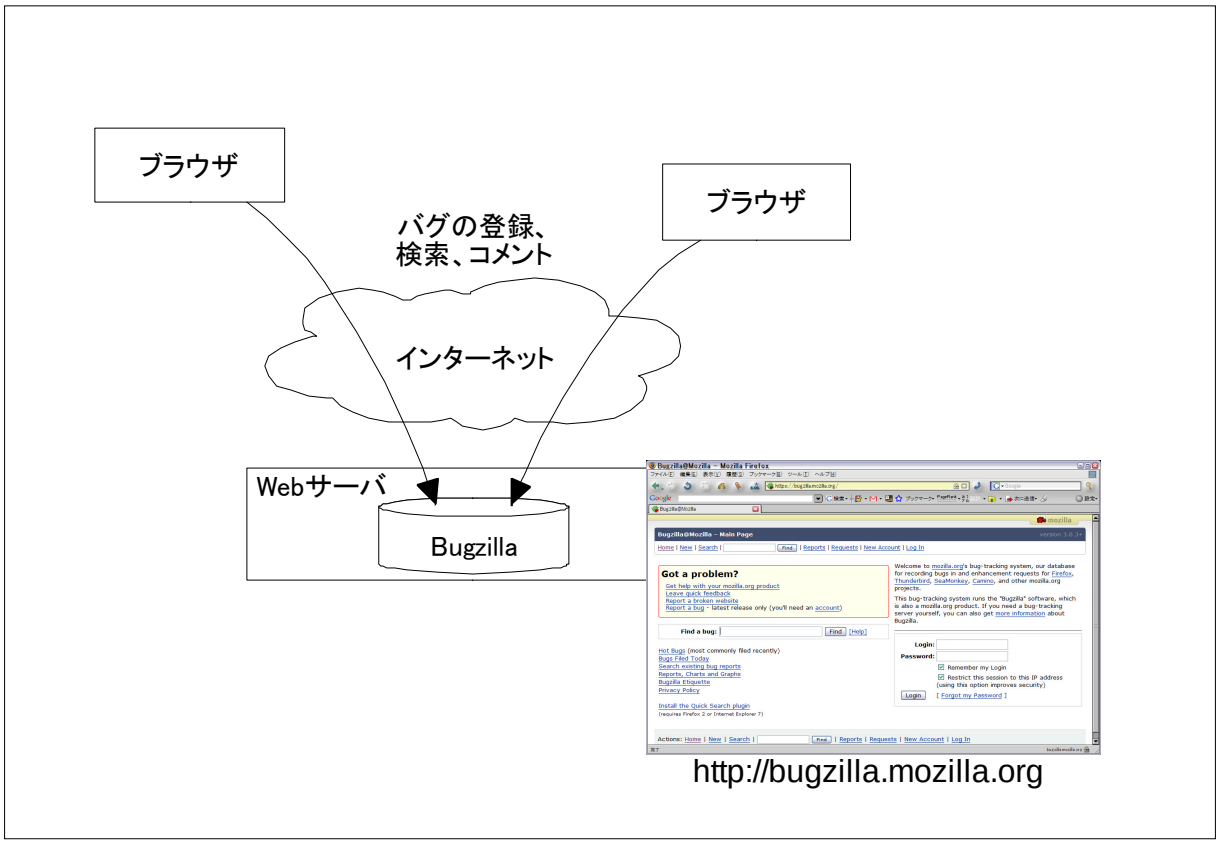

図 I-17-10.　Bugzilla 概念図

【解説】

1) Bugzilla とは

Bugzilla とは、mozilla.org によって開発された、Web ベースのバグ追跡システムであり、数多くのプロジェクトで採用されている。

参考：http://www.bugzilla.org/

http://bugzilla.mozilla.gr.jp/

2) Bugzila の利用方法

Bugzilla では、バグを参照するのみの場合を除き、アカウントを必要とする。アカウント登録申請のためのフォームが用意されており、フォームへメールアドレスを入力して送信することで、アカウント作成のアナウンスメールが届くと同時に、Bugzilla の利用が可能となる。

3) Bugzilla におけるバグ

Bugzilla では、バグ情報を掲示板のスレッドのような形式で管理している。バグに関連付けることができる情報として以下のようなものがある。

(http://developer.mozilla.org/ja/docs/Bugzilla-jp:Guide:About:BugDetails より一部抜粋)

* プロダクト
* コンポーネント
* ステータス
* 処理方法
* 担当者
* URL
* キーワード
* プラットフォーム
* OS
* バージョン
* 優先順位
* 深刻度

4) バグ検索機能

Bugzilla では、上記のバグに関連付けられた多数の情報のうちの一部を指定し、バグを検索することができるようになっている。検索条件は非常に柔軟な指定が可能である。