ソフトウェア製品の障害テスト実践事例

岩本裕子†

†日本アイ・ビー・エム（株）アドバンスト・テクニカル・サポート　〒261-8522 千葉市美浜区中瀬 1-1
E-mail:　†iyuhko@jp.ibm.com

あらまし　ソフトウェア製品の品質向上のためには，事前に十分なテストを行い，ソフトウェア・バグを叩き出し，修正することが必須であるが，とりわけ，障害テストの持つ意義は大きい．しかし，障害テストを効果的に行い，成果をあげるためには，その特性から，様々な困難を伴う．そのため，なかなか十分な障害テストを行えず，潜在的なソフトウェア・バグが見逃されている．本論文では，ソフトウェア製品における障害テストの重要性に続き，その難しさを述べる．そして，如何にして効果的な障害テストを実施するかについて，その方法論を，実践事例を元に紹介する．

キーワード　障害テスト，エラー・リカバリー，バグ，アプリケーション

Practice of Error/Recovery Test for software product

Yuhko IWAMOTO†

† Advanced Technical Support, IBM Japan, Ltd.　1-1, Nakase, Mihama-ku, Chiba-shi, Chiba 261-8522 Japan
E-mail:　†iyuhko@jp.ibm.com

Abstract　Substantial test prior to the shipment of software product through the detection and correction of software bugs is a key factor for the improvement of the quality of the products. Error/Recovery Test is the most significant factor above all kinds of test. However effective implementation of Error/Recovery Test for the achievement has some difficulty because of it's characteristics. As the result, insufficiency of Error/Recovery Test　results in miss of chance to discover potential software bugs. This report refer to the importance of Error/Recovery test and it's difficulty, and then shows you practice of　effective Error/Recovery test approach.

Keyword　Recovery Test,　Error/Recovery,　bug,　application

1. 障害テストの重要性と困難さ

ソフトウェア製品の品質向上を目的として，出荷前に，テストを実施したところ，その問題(ソフトウェア・バグ)の多くが，障害テスト実施時に，発生したことが分かった（図１参照）．

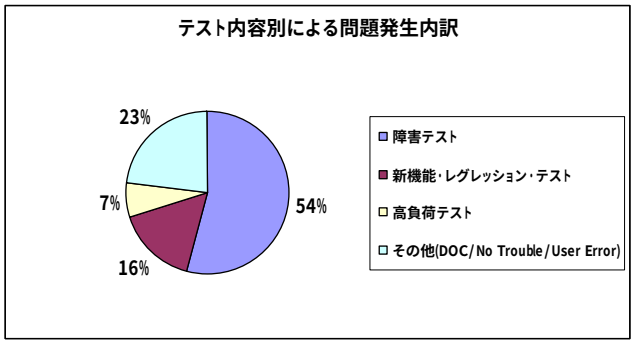

図１．テスト内容別による問題発生の内訳

ソフトウェアの新バージョンが出荷される場合，互換性保障の観点からの既存機能テスト（レグレッション・テスト），或いは，新バージョンの“売り”である，新機能テストは，開発部門によって，事前に，繰り返し実施される場合が多い．よって，その過程で発見されたバグは，事前に修正されている．

一方，障害テストの場合，事前のテストが不十分であり，潜在的なソフトウェア・バグが見逃されがちである．その結果，製品が出荷された後，ハードウェア障害等によるシステム･ダウンに見舞われて，初めて，連鎖反応としてソフトウェア・バグも顕在化し，システムが再始動できずに業務が長時間にわたって停止したり，データの整合性が失われ，その回復に多大な労力と時間がかかる，といった甚大な被害を招くことがある．情報システム障害原因の中で，ハードウェア障害(77%)が，アプリケーション・エラー(13%)，オペレーション・ミス(10%)を押さえて，最も多い日本の状況[1]を考えると，この様な深刻な被害を事前に防ぐためにも，製品出荷前に，十分な障害テストを実施すべきである．しかし，障害テストの場合，以下の特性から，その効果的な実施に，困難を伴っている．

第一の特性としては，障害テストは，アプリケーション稼動環境で実施しなければ，その成果を期待できない点である．障害テストは，対象ソフトウェア製品の上で動くアプリケーション，それも，出来るだけ，実環境に近いもの，そして，ピーク時間帯の稼働環境を用意してこそ，成果が期待できる．複雑に入り組んだ

業務アプリケーションが，高負荷状態で稼動している際に起きるハードウェア障害こそ，ソフトウェア・バクを顕在化させる可能性が高いからである．しかし，ソフトウェア製品を開発する部門は，製品そのもののプロ集団であり，業務アプリケーションのスペシャリストではない．また，開発部門は，実際にその製品を使用されるお客様に日々接している営業部門からも，組織的に分離しており，アプリケーション情報を得にくく，製品テストに反映しにくい場合が多い．その結果，開発部門が，ソフトウェア出荷前に障害テストを行う場合，アプリケーションが稼動していない環境で行なうか，稼動させるとしても，単純なプロトタイプ的な処理を用意するに留まりがちである．そのため，非常に初歩的なソフトウェア・バグは発見できたとしても，製品が市場に出た後に，現実に起きえる深刻・複雑な問題にまでは，対応しにくい．

障害テストの第二の特性としては，テスト実施時に発見される問題(ソフトウェア・バグ)は，通常の機能テストと異なり，その顕在化がタイミングに依存する場合が多く，ユーザーが意図的に再現するのが困難であるという点があげられる．その為，問題が発生した時点で,その解析の為に必要なすべての資料を,確実に,取得できるよう，事前に，念の入った緻密な準備が必要となる．特に，実際の業務を想定したアプリケーション稼動環境でのテストの場合，ユニット・テストとは異なり,資料のために必要となる資源環境も巨大で,テスト手順も複雑となるが，製品開発部門には，それに対応できるノウハウが少ない．しかし，筆者は，障害テストの成否は，この準備段階に左右されると言っても過言でないと考えている．

以下，本論文では，障害テストの方法論を，その準備段階・実施段階・評価段階における実践事例を元に紹介する．

2. 障害テストの準備段階

2.1. テスト構成(ソフトウェア, ハードウェア)

ここでは，障害テスト実施のための事前準備として行ったことを，具体例を示しながら，紹介する．なお，ここでの障害テストの位置づけは,以下の(IBM メインフレーム[2]zServer 上で稼動する)ソフトウェア製品の中でも，特に，新しく出荷された DB2 for z/OS V8[3]の品質向上のための事前テストである．

- z/OS R4(オペレーティング・システム)
- DB2 for z/OS V8(リレーショナル・データベース)
- CICS TS 2.2(トランザクション・サーバー)[4]

ハードウェア構成としては，IBM zSeries z990(2084-B16)[5]を 4 つの区画に分け,うち 2 区画は,OS とミドルウェア(DB2 と CICS)用とし，残り 2 区画を内部結合装置として使用することで，並列シスプレックス・データ共用環境[6]を構築した．構成のイメージは，図 2 に示す通りである．

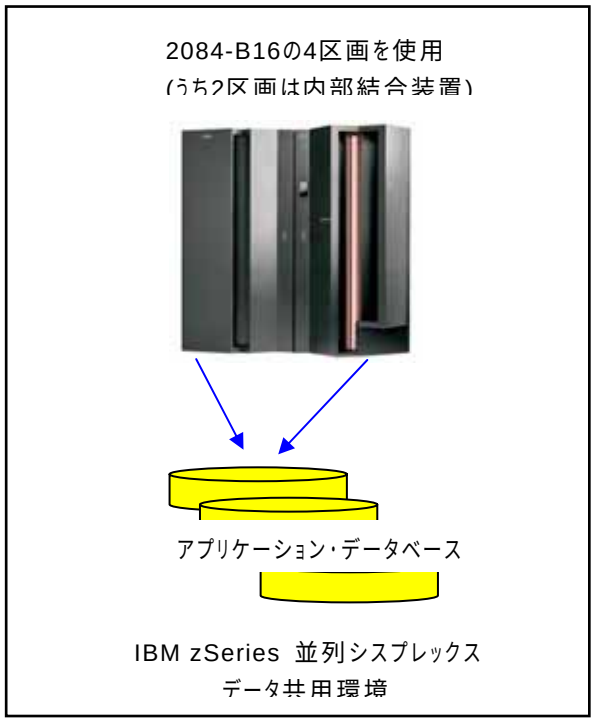

図 2．構成イメージ

テスト環境は，出来るだけ実際のお客様環境に近いものが望ましいが，リソースの制約で縮退構成にならざるを得ない．但し，テストするための障害コンポーネント(ハードウェア)を事前に決めて，それを満たす構成を組むことが必要である．ここでは，各区画，或いは，区画と内部結合装置が両方同時に，電源障害などでダウンした場合に，そこで稼動していたソフトウェアが，正常区画で自動的に立ち上がり，アプリケーションの継続処理が正常に行なえることを確認するため,それを満たすための構成を組んだ．

2.2. アプリケーションの準備

障害テストの第一の特性で前述したとおり，テスト実施に際しては，ソフトウェア・バグが顕在化しやすいように，高負荷のアプリケーション環境を用意する事が重要である．

ここでのアプリケーションは,CICS TS 2.2 の上で動くトランザクションが,DB2 for z/OS V8 上のデータベースに対して，参照(Fetch)，更新(Update)，挿入(Insert)する処理から構成されている．

表 1 は，実際に当該製品を使用する予定のお客様に対してリサーチし，それを元に作成したアプリケーション・プロファイルである．

ここでは,テスト環境として,5 種類のテーブル(Table1～5)と，6 タイプのトランザクション(Trx1～6)を用意

した．なお，テーブルとは，データベース表のことであり，各テーブルの行長，行数は，表 1 に示すとおりである．また，トランザクションをドライブするためのツールとして TPNS[7]を使用し，ピーク時を想定した，約 110 件/秒のトランザクション・レートが出るように準備した．

表 1. トランザクション・プロファイル

	Table1	Table2	Table3	Table4	Table5
行長 (ﾊﾞｲﾄ)	7514	1888	81	106	116
行数	110 万	550 万	33	294 万	108 万
Trx1	Fetch	Fetch	Fetch	-	-
Trx2	Fetch Update	Fetch	-	-	-
Trx3	Fetch Update	Fetch	-	-	Insert
Trx4	Fetch Update	Fetch	Fetch	-	Insert
Trx5	Fetch Update	Fetch	Fetch	Insert	-
Trx6	Fetch Update	Fetch	Fetch	Insert	Insert

上記アプリケーションの稼動中に，区画や内部結合装置を，ハードウェア・コンソールから停止させることで，障害状況を意図的に発生させ，ソフトウェアの動きが設計通りであること，及び，バグによる二次障害が起きないことを確認した．

2.3. テスト･プロジージャ雛形の事前準備

テスト・プロジージャ雛形とは，システムの立上げ(IPL)から，障害テストの実施，そして，テスト終了後のシステム立ち下げ(シャットダウン)に至る手順を，時間経過に沿って記録するためのものである．

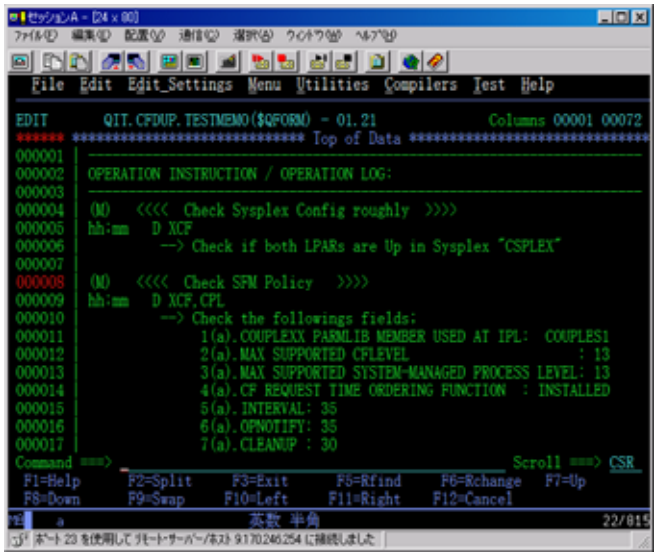

図 3．テスト・プロシージャ雛形(共通部分)

図 3 は，全体で 700 行余りあるテスト・プロシージャ雛形(共通部分)の数行を抜粋したものである．ここでは，IPL 後のシステム稼働環境をチェックする為のコマンドと，入力後に期待される出力結果を記述しており，テストを実施する際，時刻を入れる仕様となっている(図中の hh:mm)．この雛形は，特定の障害テストに依存しない共通の手順を記述しており，これ以外に，複数の障害テストケースに対応して，個別の雛形を用意し，テスト実施時に，それを共通雛形に挿入する形で使用している．

共通雛形の意義は，次の通りである．第一に，障害テスト実施時の構成や環境を，過不足ない決められた手順で，確認･変更できることがあげられる．IBM zSeries のようなメインフレーム環境を，社内テストとして使う場合，コストの観点から，通常，複数のグループで，日割りや曜日毎にスケジュール調整をして占有使用することが多い．そのため，あるグループが障害テストを実施した翌日に，他のグループが，別の目的を持って，同じ zSeries を占有することが，しばしばある．テストの目的が違えば，(CPU 数，ストレージ等の)必要なリソースも異なる為，構成が，その目的に合わせて変更される場合も多々ある．その結果，前回の障害テスト時と，(数日後の)次回の障害テスト時の構成が，気づかぬうちに，変わっていることもあり，変更の有無は，テスト時にコマンドによって表示結果を確認しない限り，知ることができない．ソフトウェア・バグの顕在化は，特定の構成に依存する事もあるため，バグ発生時の解析をスムーズに進めるためにも，環境を確認し，必要の応じて変更するための手順の徹底は重要である．このことは，メインフレーム環境に限らず，特定のリソースを複数の目的を持って共有してテスト使用する場合には，大切なポイントとなる．また，障害時のソフトウェア・バグ顕在化に備えて，事前にトレースをかけておくなど，必要なコマンドは，すべて，この雛形に組み込む必要がある．第二に，この雛形は，障害テスト終了後の参照用としての意義がある．障害テストの場合，区画停止等によって，意図的に，システム・ダウン状況に陥らせるわけであるから，ダンプが書かれたり，コンソール上に膨大な量のメッセージが表示される．その場ですべてのメッセージを確認するのは，到底不可能であるから，コマンド入力や，区画停止の時刻を，プロシージャに記録しておくことで，後から，それを手がかりに資料を見直すことが出来，効率的な分析が可能となる．

特定の障害ケースに対応するプロシージャ雛形も，図４に数行を抜粋して，紹介する．

障害ケース用の雛形の中で，重要なのが，障害を起こす(区画を停止させる)直前の手順である(図４中の <<Check the status >>と，<< get SYSLOG >>の手順)．通常，システム・ダウンが起きた場合には，それまで

のログが，すべて保存されずに消えてしまうため，たとえソフトウェア・バグが見つかったとしても，後で解析しようにも，殆ど資料がない．資料がない場合，第二の特性で前述した通り，バグの顕在化がタイミングに依存し，ユーザーが意図的に再現するのが困難であるため，バグが再現するのを，繰り返しテストしながら，気長に待つしかない．業務に対する影響が，多大かつ深刻なものであればある程，再現待ちという状況は，お客様にとって，大変なストレスとなる．
よって，障害テストの際には，区画を停止する(図中の<< deactivate CEC1 & ICF2 >>以下の手順)直前に，その時のシステム状況をコマンドにより表示させ(図中の<<Check the status >>手順)，その結果ログを，DISK に保存する(図の<< get SYSLOG >>手順)等，バグ発生時の資料取りの準備を，怠ってはならない．この点については，「2.5 資料取得のための環境整備」でも，詳述する．

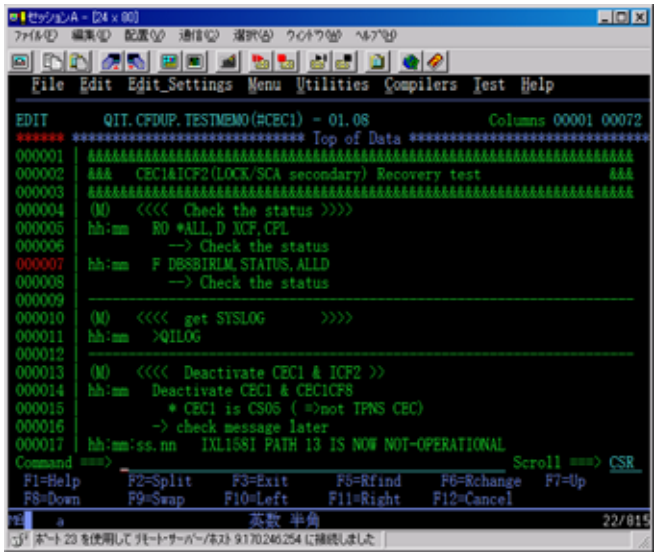

図４．テスト・プロシージャ雛形(障害ケース)

また，障害発生時に，データ整合性が失われなかったことを確認するための方法も，事前に検討し，その手順を雛形に組み込む必要がある．ここでは，アプリケーション・プログラムに，カウンターを取る等の仕掛けを組み込み，それとは別に，カウンターを計算して表示させる為のプログラムを用意し，障害発生の前後で，実行させることで，整合性の確認を行なった．
この様に，テスト・プロシージャ雛形は，問題発生時には，(テスト担当者とは別の)解析者が参照することからも，必要項目を漏らすことのないよう，事前に，十分に検討し，出来るだけ，分かりやすく，具体的に，かつ，正確に，準備することが重要である．一見，当たり前の事の様に思えるが，雛形に，項目の抜けがないよう，緻密に準備するのは，意外と難しい．完璧に準備したつもりでも，実は漏れがあり，テストを実施する段になって，慌てることもしばしばである．

2.4. 簡略化されたコマンド(コマンド･リスト)の準備

テスト実施中に，複数コマンドを１つずつ手入力するよりも，コマンド・リストを事前準備し，起動する方が，効率的である．特に，桁数の多いコマンドの場合，入力ミスを防ぐためにも，コマンド・リストを準備するべきである．コマンド・リストについては，障害テストに限らず，通常の単体テストでも，準備することが多いため，障害テストに特化したノウハウではないと考える．ちなみに，前項で紹介したテスト・プロシージャ雛形内では，約 20 強のコマンド・リストを起動する仕組みになっている．また，図５は，図４(前掲)で紹介した，障害発生直前にシステム・ログを保存するためのコマンド・リスト(QILOG)である．

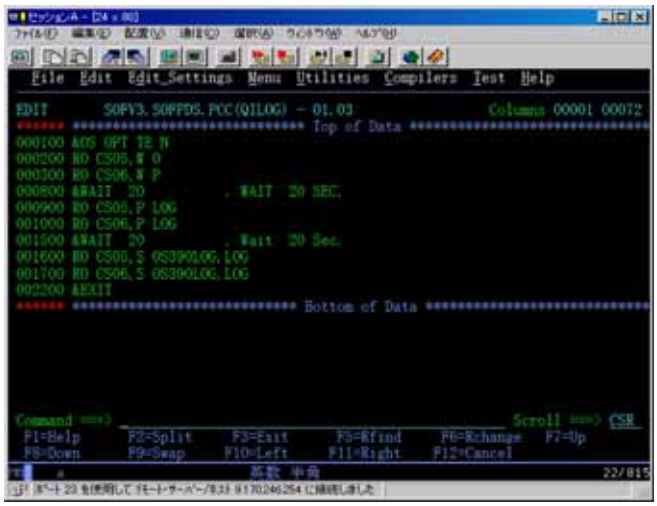

図５．コマンド・リスト(システム・ログの保存)

2.5. 資料取得のための環境整備

障害テスト実施中に，ソフトウェア・バグが見つかった場合，(解析の為の)取得資料に不足があってはならない．簡単には，問題を再現させることができないという現実を理解し，万全の準備を期すべきである．具体的には，ダンプ，トレース書き出し用の資源，アーカイブ・ログ用のディスク・スペース等は，開始前に不足なく準備しておくべきであるが，このことは，意外と忘れてしまうことが多く，いざ，障害が起きた時に，中途半端な資料しか取れず，後の祭りになることがある．
例えば，DB2 for z/OS を例にすると，ダンプ用の資源として，次の考慮点がある．この製品は，データベース処理を行なうソフトウェアであるから，処理中に，ストレージ(メモリ)上に，データを展開する．64 ビット・メモリ・アドレッシングにより，メモリ空間が拡大し，障害テスト用に準備したアプリケーション環境で，数十ギガ・バイドのデータを処理する場合，障害

が発生すると，巨大なストレージ上の内容が，自動的に一気に，ダンプされることになる．事前に，十分なサイズのダンプ書き出し先のディスクを用意しておかないと，いざ，障害が発生した場合，部分的なダンプしか取得できず，その結果，原因を究明することが困難となる．

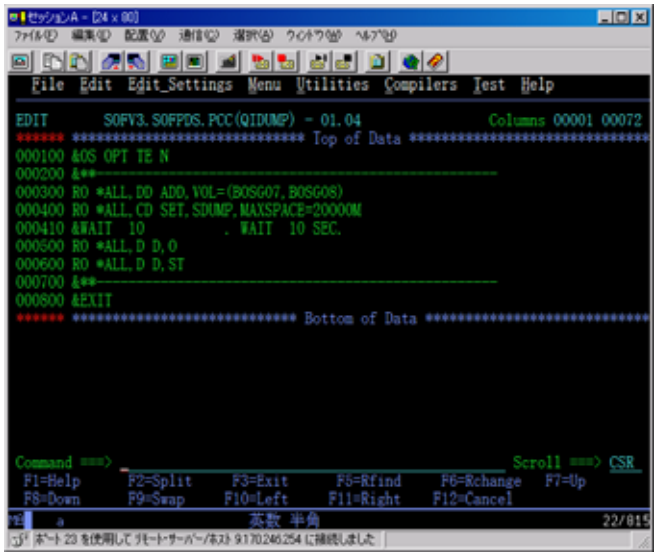

図６．コマンド・リスト(ダンプの為の環境準備)

図６は，そのような事態を防止するために，十分なダンプ用環境を準備するためのコマンド･リストであり，テスト・プロシージャ雛形に組み込まれている．
また，データベース・ログを例にとると，次の通りとなる．データベース製品は，更新処理の際にログを書き出す．ソフトウェアが異常終了した場合に，再始動に際して，このログを参照しながら，データベースの回復を行い，データ整合性を保ちつつ起動完了する．データ整合性が保たれてこそ，データベース製品としての意味があることから，たとえシステム・ダウン等の障害に見舞われても，データ整合性が失われることは許されない．万が一，ソフトウェア･バグによって，データ整合性が失われた場合は，その解析・修正のために，ログが重要な意味を持つ．ピーク時を想定した高負荷なアプリケーション環境でテストを行う場合，書き出されるログ量も膨大となり，テスト中に何度も，自動アーカイブされるが，アーカイブされたログは，障害テスト後のデータ整合性の確認が完了するまで，すべて，保存しておくだけの環境を準備する必要がある．
この様に，障害テストに際しては，事前に，必要な環境を漏れなく検討し，確保し，必要であれば，テスト・プロシージャ雛形に手順を組み込むことが重要である．

3. 障害テストの実施段階

テストの実施は，率直に言って，前述の準備に比べれば，非常に楽である．事前にプロシージャ雛形を完璧に準備さえしておけば，それにそって，進めていけばいいからである．但し，障害テストの場合は，一名でなく，二名で実施するのが望ましい．障害を起こす瞬間には，ハードウェア・コンソールから区画を停止(ディアクティベート)する担当者と，その時刻をプロシージャ雛形に記録する担当者が必要となる．また，全体の手順を通じて，OS コンソールからコマンド(或いはコマンド・リスト)を入力する担当者と，それを，プロシージャ雛形に記録する担当者が必要である．また，障害を起こした直後には，大量のメッセージやダンプが書き出されるため，二名で，結果を把握した方が見落としが少ない．特に，バグの顕在化によって，システムが，予想した結果と異なる動きを取った場合には，すべての取得資料の保存と記録を，手分けして，速やかに行なう必要がある．
実施する障害ケースの優先順位については，障害の発生可能性とその重大性を考慮して，実施スケジュールを組むべきである．ここで紹介した事例では，①OS 区画障害，②筐体全体障害，③内部結合装置(CF ; Coupling Facility)障害の順で実施した．①は発生頻度が他と比べて高い点から，②は障害発生時の影響の重大性の観点から決定し，③は発生頻度が低い上に，業務への影響が小さいことから，優先度を下げている．

4. 障害テストの評価段階

評価段階においては，明らかにバグと思われる状況が発生した場合のプロセスは，比較的単純明快である．その問題の報告・資料送付・修正待ち・修正確認という定型的なプロセスを通るからである．一方，一見，問題がなさそうな結果となった場合，何をもってして，“問題なし”と結論付けるかは，重要なポイントとなる．以下，具体的な評価確認項目を述べる．

4.1. 障害発生時の状況確認

前項で述べた，障害発生頻度が，他と比べて高いと考えられる OS 区画障害のケースでは，ハードウェア障害によって，その上で稼動していたソフトウェアも異常終了(アベンド)するが，その後の動きについては，個々の製品仕様によって異なる．
筆者が使用した並列シスプレックス，データ共用環境の場合は，2 台の OS 用区画上に，(DB2，CICS といった)ミドルウェア製品が，それぞれ全く同じ構成で稼動している(図７の CEC7 と CEC8)，所謂，クローン環境である．そして，区画の一方が障害でダウンした場合，正常系の区画で稼動していたミドルウェアと，その上で動くアプリケーションは，そのまま処理を継続する

が，障害系の区画で稼動していたミドルウェアは，アベンドした後，反対側の正常区画上で，自動的に再始動する仕組みとなっている．実際には，再始動の有無や各ミドルウェア製品の再始動の順番は,あらかじめ,定義しておくわけだが，それぞれのミドルウェア製品が，期待通りに，正常に再始動し，アプリケーション処理が継続できることを確認する．

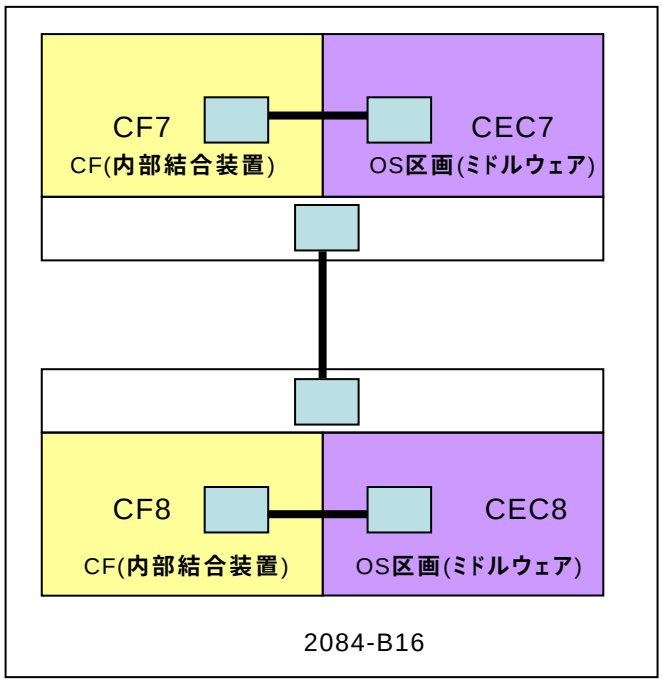

図７．区画イメージ図

前項で二番目の優先順位とした筐体全体障害は，電源障害等によって，筐体ごとダウンするケースであり，(図７中の)CF7 と CEC7，或いは，CF8 と CEC8 の同時二重障害を意味する．内部結合装置(CF)用の区画(図７の CF7 と CF8) は，並列シスプレックス環境における各 OS 区画間の情報のやり取りに使われており，その重要性から，情報は二重書きされている(CF Structure Duplexing[8])．よって，筐体障害(例；CF7 と CEC7 の同時障害)が起きた場合には，①CF 二重化が自動的に停止すること，②正常 CF(CF8)と，正常 OS 区画(CEC8)によって，アプリケーションの継続処理が可能であること，③障害 OS 区画(CEC7)のミドルウェアが，期待通りに，CEC8 上で再始動すること，以上の 3 点が，確認項目となる．

4.2. 障害後の回復確認

障害テストにおいては，単にハードウェア障害発生時のソフトウェアの動きを確認するだけでなく，回復後，構成を戻す場合の動きも確認すべきである．なぜなら，現実の運用では，障害が起きたハードウェアが回復すれば，それを起動して，従前の構成に戻すわけで，この際に，バグが顕在化する可能性も否定できないからである．障害発生時の評価に比べて，戻し時の評価については，しばしば軽視されがちであるが，実際の運用環境を視野にいれれば，念入りに評価すべきと考える．OS 区画障害の場合は，区画回復後，IPL と(他区画で再始動させた)ミドルウェアの戻しが発生し，CF 区画障害の場合は，回復後，二重化の再開となる．24 時間システム連続稼動が要請される昨今，障害発生時の業務継続の要請はもとより，構成の戻し，即ち，回復処理時の業務継続も必須要件となる．よって，アプリケーションが稼動した環境で,構成の戻しを行い，期待通りの結果が得られることを確認する必要がある．稼動中アプリケーションへの影響については，次項で詳述する．

4.3. 稼動中アプリケーションへの影響確認

区画障害時のミドルウェア自動再始動や CF 二重化は，システム連続稼動実現，可用性向上のために搭載された機能と言える．よって，製品の品質という観点では，稼動中アプリケーションへの影響の有無・程度を，検証し，評価するべきである．たとえ，障害発生時や回復時の製品の動きが，予想通りとしても，その最中に，長時間にわたってアプリケーション処理が中断せざるをえない場合には，(可用性向上としての)製品の目的が達せられたとは言えない．ここで見落としてはならないのが，回復時(構成戻し時)の，アプリケーションへの影響である．前述の通り，通常，可用性向上の機能は，障害発生時のアプリケーションへの影響度合いという観点では十分な検討がなされるが，その後，障害コンポーネントが回復し，構成を従前に戻す際の影響については，軽視されがちだからである．最大レスポンス・タイムの突出が見られたり，トランザクション処理量が落ち込んだ場合には，その原因を究明するため，テスト実施中に取得したログ等の資料を使って，調査し，それが，仕様なのか，ソフトウェア・バグによるものなのか，環境特性からくるものなのか，オペレーション・ミスが原因なのかを，明らかにする必要がある．

5. まとめ

当論文では，まず，ソフトウェア品質を高めるための，障害テストの重要性について述べた．同時に，効果的な障害テストの実施が，いかに困難であるかを，その特性を紹介しつつ，言及した．そして，最後に，障害テストのもつ困難な課題を解くための方法論を，筆者の実践事例を元に，提示した．中でも，障害テストは，その準備期間中の作業の精度と質が，テストの成否に最も影響することを述べた．そして，テスト結果の評価としては，単にソフトウェアの動きが，期待通りであるか否かの評価だけでなく，現実のお客様環

境での，業務と運用を踏まえた評価が重要であることを述べ，稼動中アプリケーションへの影響といった具体的な評価例を示した．この事例が，ソフトウェアテストを検討され，実践される方々の一助になれば，幸いである．

文　献

[1] ガートナー・リサーチ,“情報システムの障害原因：日本と世界を比較する”
http://w3-6.ibm.com/jp/domino56/ebiz/ebiznews.nsf/doc/FC8F2590AD1827D949256C8600236B65?OpenDocument 2002 年 10 月 18 日

[2] 千田淳, 矢口竜太郎, ″メインフレーム大解剖”，日経コンピュータ，583 号，p50－p.69，2003 年 9 月 22 日号.

[3] DB2 UDB for z/OS Version 8,
http://www-6.ibm.com/jp/software/zseries/prod/db2v8/

[4] CICS,http://www-6.ibm.com/jp/software/websphere/ft/cics/

[5] 千田淳，矢口竜太郎,“z990 にみる大規模サーバーの将来像”，日経コンピュータ，583 号，p70－p.75，2003 年 9 月 22 日号.

[6] IBM 並列シスプレックス機能
http://www-6.ibm.com/jp/servers/eserver/zseries/os/zos/sysplex.shtml

[7] TPNS Overview,
http://www.salemsoftware.com/html/body_tpns.html

[8] IBM「システム管理による CF ストラクチャーの二重化」
http://www-6.ibm.com/jp/software/support/techflash/49256F09000DDA32_jp_servers_eserver_support_zseries_all.html