正　本

令和元年（ワ）第１０９４０号　損害賠償請求事件

原告　森　次　茂　廣

被告　株式会社[redacted]

第６準備書面

令和３年３月１日

大阪地方裁判所第２６民事部合議係　御中

原告訴訟代理人弁護士　[redacted]

同　[redacted]

同　[redacted]

（担当）同　[redacted]

第１　被告準備書面４に関する主張

１　ソースコードの創作性について

（１）　本件プログラム１について

本件プログラム１について，共通ルーチンを加えた完全なソースコードとして，甲２８号証を提出する。その上で，被告準備書面４について反論する。

ア　被告は，本件プログラム１に関し，第２準備書面別紙２の青色の吹き出しで原告がコメントした箇所は，機能の説明であり，原

告は機能に創作性があると主張するに過ぎないと述べる（被告準備書面４第２の１参照）。

この点，甲２８号証のソースコードにも関連するため反論するが，原告が青色吹き出しでコメントをしている箇所は，当該ソースコードがどのような処理をしているのかを説明しているものであり，機能の説明ではない。そして，原告が主張しているのは，甲２８号証の具体的なソースコード自体に創作性があるというものである。

イ　また，被告は，サブルーチンの最適化に関するファイル「mdlCtrlSub03.vb」について一般的なプログラミング言語からなるアルゴリズムにすぎず，その配置や順序に創作性が認められないと主張する（被告　準備書面４第２の３参照）。

この点，被告は，「mdlCtrlSub03.vb」のソースコードについて，「If vMode = eCtrlMode.ecmGet Then」のソースコードに対して，「vMode = ecmGet の場合」と処理内容を記し，「rStr = rCtrl.Text」のソースコードに対して，「[rCtrl]テキストボックスの値を rStr に取得」と記載しているが（乙１４），プログラム言語の基本命令を指して，その命令の動作を説明したものであり，組み合わせた指令に対する指摘ではないため，被告の主張は失当である。乙１４号証のその他の箇所についても同様であり，プログラム言語の基本命令についての動作の説明をしているにすぎない。

ウ　被告が例として挙げた「mdlCtrlSub03.vb」に記載されているサブルーチンは，コントロールに対して，データの設定（ＰＵＴ）と取得（ＧＥＴ）を行う処理を一元化するコードを各種コントロール毎及び変数型毎用に作成したものである。当該サブルーチンは，

ソースコードを一元化し，プログラム全体の最適化を目的として作成されている。

例えば，テキストボックスコントロールにデータを設定する際は「txtTitle.Text = mTitle」，テキストボックスコントロールからデータを取得するときは「mTitle = txtTitle.Text」のように，別々に右辺左辺が逆になったコードを記述する方法が選択肢として一般的に考えられる。

その場合のソースコードは以下の＜例１＞のとおりとなる。

＜例１＞

```
If vMode = ecmSet Then
    txtTitle01.Text = mTitle( 1)
    txtTitle02.Text = mTitle( 2)
    txtTitle03.Text = mTitle( 3)
    txtTitle04.Text = mTitle( 4)
    txtTitle05.Text = mTitle( 5)
    txtTitle06.Text = mTitle( 6)
    txtTitle07.Text = mTitle( 7)
    txtTitle08.Text = mTitle( 8)
    txtTitle09.Text = mTitle( 9)
    txtTitle10.Text = mTitle(10)
Else If vMode = ecmGet Then
    mTitle( 1) = txtTitle01.Text
    mTitle( 2) = txtTitle02.Text
    mTitle( 3) = txtTitle03.Text
    mTitle( 4) = txtTitle04.Text
    mTitle( 5) = txtTitle05.Text
    mTitle( 6) = txtTitle06.Text
    mTitle( 7) = txtTitle07.Text
    mTitle( 8) = txtTitle08.Text
    mTitle( 9) = txtTitle09.Text
    mTitle(10) = txtTitle10.Text
EndIf
```

もっとも，原告はそれを同じコードで記述可能にするサブルーチンを作成するという選択肢をとった（甲２８号証１０２頁１１行目乃至１７行目）。

＜原告が作成したソースコード①＞

Public Sub CtrlIO_TextBoxToStr(ByVal vMode As eCtrlMode, ByRef rCtrl As TextBox, ByRef rStr As String)

If vMode = eCtrlMode.ecmGet Then

```
rStr = rCtrl.Text
Else
rCtrl.Text = rStr
End If
  End Sub
```

上記のサブルーチンを利用すれば，動作モードにより処理内容を切り替えるため，「CtrlIO_TextBoxToStr(vMode,txtTitle,mTitle)」の同じコードで設定と取得が実行できるようになる。具体的には，以下の＜例２＞のようなコードとなる。

＜例２＞

```
CtrlIO_TextBoxToStr(vMode,txtTitle01,mTitle( 1))
CtrlIO_TextBoxToStr(vMode,txtTitle02,mTitle( 2))
CtrlIO_TextBoxToStr(vMode,txtTitle03,mTitle( 3))
CtrlIO_TextBoxToStr(vMode,txtTitle04,mTitle( 4))
CtrlIO_TextBoxToStr(vMode,txtTitle05,mTitle( 5))
CtrlIO_TextBoxToStr(vMode,txtTitle06,mTitle( 6))
CtrlIO_TextBoxToStr(vMode,txtTitle07,mTitle( 7))
CtrlIO_TextBoxToStr(vMode,txtTitle08,mTitle( 8))
CtrlIO_TextBoxToStr(vMode,txtTitle09,mTitle( 9))
CtrlIO_TextBoxToStr(vMode,txtTitle10,mTitle(10))
```

＜例１＞のように別々に右辺左辺が逆になったコードを記述する方法を取る場合には，２３行要するものが，原告のサブルーチンを使うことで１０行に収めることができる。加えて，設定と取得が同じコードであるため，別々で記述したときに起こりうるコードの不一致等がなくなり，ヒューマンエラーを防止することができ，効率化につながる。

もっとも，原告は，更に最適化を図るために，＜原告が作成したソースコード①＞と併せて，コントロール自体ではなくコント

ロールの名前を引き渡す以下のサブルーチンを作成するという選択肢を採用した（甲２８号証　１０２頁６行目乃至１０行目）。

＜原告が作成したソースコード②＞

```
Public Sub CtrlIO_TextBoxToStr(ByVal vMode As eCtrlMode, ByRef rForm As
Form,ByVal vCtrlName As String, ByRef rStr As String)
Dim mCtrl As TextBox = FindControl(rForm, vCtrlName)
CtrlIO_TextBoxToStr(vMode, mCtrl, rStr)
End Sub
```

コントロール自体を引き渡す場合は，コントロールの数だけサブルーチンを呼ばなければならないが，コントロールの名前を引き渡す場合は，ループで名前を作成して，サブルーチンを呼ぶように構成すれば，以下の＜例３＞のような少ない行でコードを記述でき，ヒューマンエラーを低減させることができる。

＜例３＞

```
For i=1 to 10
    mCtrlName="txtTitle" & i.ToString("00",i)
    CtrlIO_TextBoxToStr(vMode,rForm,mCtrlName,mTitle(i))
Next
```

＜原告が作成したソースコード②＞により，＜例２＞のようなナンバリングのみが違った同じようなコードを記述する必要がなくなるため，＜例２＞からより一層最適化を図ることができる。

また，動作モードについて，想定されている動作が設定と取得のみであったため，ソースコードに直接「０」と「１」を記述する方法を採ることも考えられたが，原告は拡張性を考慮した結果，設定を「ecmSet」，取得を「ecmGet」とする選択をしており，以下のようなソースコードを作成している。

＜原告が作成したソースコード③＞

```
Public Enum eCtrlMode
  ecmSet = 0 '情報をコントロールにセット
  ecmGet = 1 'コントロールから情報を取得
End Enum
```

以上より，原告は，「mdlCtrlSub03.vb」のソースコードにおいて，＜例１＞や＜例２＞のようなコードを記載する等多様な選択肢があった中から，最適化を最大限図るという観点から，独自に＜原告が作成したソースコード①＞及び＜原告が作成したソースコード②＞を作成し，ボリュームのある処理を＜例３＞のようにできるだけ少ないコードで実現するという選択肢を採用している。また，原告は多様な選択肢がある中，拡張性，生産性及び信頼性を考慮して，＜原告が作成したソースコード③＞のようなコードを独自に作成している。これらのソースコードは原告のこれまでの経験に基づくものであり，他のプログラマーが作成しても同様の記載になるものではない。

上記の＜例１＞乃至＜例３＞，＜原告が作成したソースコード①＞乃至＜原告が作成したソースコード③＞の説明は，「mdlCtrlSub03.vb」のソースコードの冒頭のテキストボックスコントロールと文字列変数に対するものであるが，それ以外のサブルーチンについても同様に最適化の観点等から原告はソースコードを独自に作成している。

よって，サブルーチン化する構成の観点から本件プログラム１のソースコードを検討しても，原告の個性が表現されているのは

明らかであり，創作性が認められる。

（２）　本件プログラム２について

被告は，サブルーチン化する構成は，プログラミングにおいて，ありきたりな構成であり，そこには創作性がないと主張する（被告準備書面４の２）。

しかしながら，上記（１）と同様に，どのようにサブルーチン化するかでソースコード自体の組み合わせや順序，呼び出し階層の表現方法等に多様性が生まれ，プログラマー毎の考え方等によって個性が生じる。

したがって，サブルーチン化されたソースコードの構造にも創作性が認められる。

第２　原告の主張

１　本件プログラム１の著作物性に関する主張（第４準備書面１）の補充

第４準備書面１で述べた点について，以下主張を補充し，甲２８号証に対応させる。

（１）　プログラムに選択の幅があること

ア　「clsAccessMdb.vb」のソースコードは，アクセスデータベースファイルの操作に関するものをクラス（オブジェクトを作る設計書）として定義して記載したものである。

マンロック内の気圧，二酸化炭素等の測定データの記録方法を検討するにあたって，被告から当初はテキストファイルで記録することを提案されていたが，総合的に考えてデータベースによる方法が

適していると考え，データベースによる方法を提案している（原告第２準備書面２（４）参照）。

そして，データベースによる方法の中にもデータベースソフトとしてどれを採用するかという選択肢があり，本件プログラム１については，Oracle（オラクル），MySQL（マイエスキューエル），ACCESS（アクセス）の選択肢があったが，原告はコストや操作性を考慮してACCESS（アクセス）を選択した。具体的には，Oracle（オラクル）は，別途ライセンスの費用がかかってしまうため，コスト面に問題があった。Oracle（オラクル）及びMySQL（マイエスキューエル）は，データベースを構築するスキルが必要であり，操作性に問題があると考えられた。一方，ACCESS（アクセス）は，OSに標準的に備わっている機能で対応でき，ライセンス費用が発生することなく，別途データベースを構築する必要もなかったが，機能に制約があるという状況であった。もっとも，機能面については，原告が独自のプログラムを創作することで技術的にカバーできると考えられたことから，原告はACCESS（アクセス）を選択した。

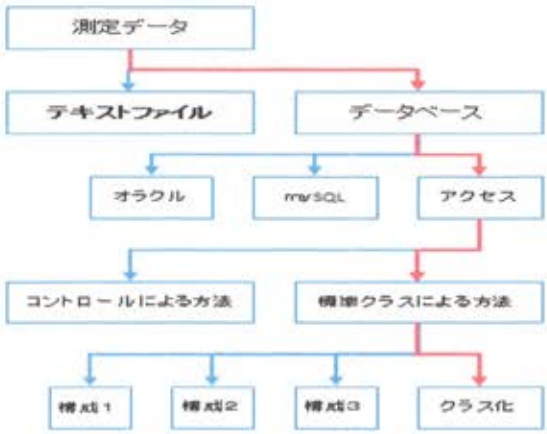
選択したデータベースを操作する方法として，コントロールによる方法と標準クラスによる方法という選択肢があり，原告はWebシステム開発の経験から，標準クラスによる方法が適していると考え，標準クラスを使用する方法を選択している。

したがって，そもそも収録方法についてテキストファイルを選択するかデータベースを選択するのかによって構成するソースコードが異なり，データベースについても，どのソフトを使用するかで構成するソースコードは異なる。そして，ソフトが決まった後も，実際にデータベースを操作するソースコードを組み立てるにあたっ

て，コントロールによる方法と標準クラスによる方法のどちらを採用するかで具体的なソースコードは異なることになる。

上記の選択肢を図に表すと以下の＜図１＞のとおりとなる。

＜図１＞



標準クラスを採用した後も，ソースコードの記載方法は多岐にわたっている。具体的には，本件プログラム１の最初の処理として，タイトルバーに表示する工事名称の取得処理が挙げられる。工事名称の取得処理の解法（アルゴリズム）としては，

①データベースをオープン

②検索命令を発する。

③検索結果からデータを取る。

④データベースをクローズ

が挙げられる。このアルゴリズムをプログラム言語及び規約で示すための選択肢の１つとして以下の＜構成１＞が考えられる。

＜構成１＞

```
メインルーチン
1: _con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & FileName
2: _con.Open()
3: _sqlCommand = New OleDbCommand("select * from `ManlockInfo` WHERE `ID`=1", _con, _trn)
4: _adapter = New OleDbDataAdapter(_sqlCommand)
5: _adapter.Fill(_Table)
6: gManlockInfo.Title = _Table.Rows(0).Item("Title")
7: _con.Close()
```

処理解説

- 1～2のコードで、アルゴリズム①の「データベースをオープン」を行います。
- 3～5のコードで、アルゴリズム②の「検索命令を発する」を行います。
- 6のコードで、アルゴリズム③の「検索結果からデータを取る」を行います。
- 7のコードで、アルゴリズム④の「データベースをクローズ」を行います。

また，コードの解りやすさを重視して，以下の＜構成２＞のようなソースコードにする選択肢もあり得る。以下のソースコードは，オブジェクト化したもので，命令文で並んだコードを処理毎（開く→取得→閉じる）に分かりやすく配列している。

＜構成２＞

```
メインルーチン
   1: dbOpen
   2: dbGetTitle()
   3: dbClose
サブルーチン
   4: Public Function dbOpen()
   5:   _con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & FileName
   6:   _con.Open()
   7: End Function
   8: Public Function dbGetTitle()
   9:   _sqlCommand = New OleDbCommand("select * from `ManlockInfo` WHERE `ID`=1", _con, _trn)
  10:   _adapter = New OleDbDataAdapter(_sqlCommand)
  11:   _adapter.Fill(_Table)
  12:   gManlockInfo.Title = _Table.Rows(0).Item("Title")
  13: End Function
  14: Public Function dbClose()
  15:   _con.Close()
  16: End Function
```

処理解説

処理を内容ごとに細分化し、動作を解り易く書しています。
メインルーチンの1～3で、動作の流れを表記し、サブルーチンの4～16で、それぞれの動作をコードで記述しています。
メインルーチンで、開く → 取得 → 閉じる と処理の流れが明確に表現されています。
サブルーチンでは、構成1の処理解説と同様に詳細が記されています。

さらに，上記＜構成２＞をさらに解りやすくするために，以下の

＜構成３＞のようなソースコードとする選択肢もある。＜構成３＞は，内部の処理に関わらず結果が正しければ良いという考えのもと処理をカプセル化し，メインルーチンでは、目的のみを明示したものである。大きなプログラムでは，細部の処理にとらわれると全体を把握することができないことから，個々の処理を完結して，大きな流れを把握するために＜構成３＞のような記載をするという方法がある。

＜構成３＞

```
メインルーチン
   1: GetTitle()
サブルーチン
   2: Public Function GetTitle()
   3:   dbOpen
   4:   gManlockInfo.Title =dbGetTitle()
   5:   dbClose
   6: End Function
   7: Public Function dbOpen()
   8:   _con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & FileName
   9:   _con.Open()
  10: End Function
  11: Public Function dbGetTitle()
  12:   _sqlCommand = New OleDbCommand("select * from `ManlockInfo` WHERE `ID`=1", _con, _trn)
  13:   _adapter = New OleDbDataAdapter(_sqlCommand)
  14:   _adapter.Fill(_Table)
  15:   gManlockInfo.Title = _Table.Rows(0).Item("Title")
  16: End Function
  17: Public Function dbClose()
  18:   _con.Close()
  19: End Function
```

処理解説
処理をカプセル化し、メインルーチンからは目的のみが解る様にした。
処理を階層化し、詳細はサブルーチンを探って見れるようにしています。

この他にも，処理スピード，使用メモリ，例外処理等の事情に合わせて，上記＜構成１＞ないし＜構成３＞とは別の記載方法でソースコードを表現することもできる。また個々のプログラマーによって，記載の方法は異なることから，「clsAccessMdb.vb」のソースコ

ードだけでも膨大な選択肢があることは明らかである。

イ　このような膨大な選択肢の中，原告が選択したソースコードが甲２８号証１頁乃至６頁のソースコード及び２１頁１６行目～３９行目の部分である。

（２）　原告の表現上の創作性があふれていること

上記（１）アで述べたアルゴリズムのうち，①のアルゴリズムを実現するためには，＜構成２＞の４乃至７のソースコードを記載するだけでも対応が可能である。もっとも，原告は，より精度の高い結果を実現するために，以下の＜原告が選択したソースコード＞を作成した。

＜原告が作成したソースコード＞

甲２８号証　１頁１３行目～３６行目

```
Public Function Connect(Optional ByVal dsn As String = "db1.mdb", _
                Optional ByVal tot As Integer = -1) As Boolean
  Try
    If _con Is Nothing Then
      _con = New OleDbConnection
    End If
    Dim cst As String = ""
    cst = cst & "Provider=Microsoft.Jet.OLEDB.4.0"
    cst = cst & ";Data Source=" & dsn
    ' データベースパスワードが設定されている場合
    ' cst = cst & ";Jet OLEDB:Database Password=xxxxx"
    If tot > -1 Then
      '_con.ConnectionTimeout = tot
      cst = cst & ";Connect Timeout=" & tot.ToString
    End If
    _con.ConnectionString = cst
    _con.Open()
    Connect = True
  Catch ex As Exception
    '     Throw New Exception("Connect Error", ex)
    Beep()
    Connect = False
  End Try
End Function
```

甲２８号証　２１頁１６行目～３９行目

```
Private Function dbOpen(ByVal vRecNo As Integer) As Boolean
  Dim mFileName As String
  mFileName = FileName
  If vRecNo >= 1 And vRecNo <= 10 Then
    mFileName = FileName & (vRecNo - 1).ToString("0")
  Else
    mFileName = FileName
  End If
  Return dbFileOpen(mFileName)
End Function
Private Function dbFileOpen(ByVal vFileName As String) As Boolean
  Dim mOpenFlag As Boolean, mFileName As String
  Dim libFF As New clsFolderFile
  If libFF.ExistsFile(vFileName) = False Then Return False
  Try
    mFileName = vFileName
    pvDB = New clsAccessMdb
    mOpenFlag = pvDB.Connect(mFileName)
    Return mOpenFlag
  Catch
    pvDB = Nothing
    Return False
  End Try
End Function
```

処理解説

データベースの基本的な命令に、例外処理を加えています。
巨大なデータを扱うため、ファイルの切り替えに対応しています。

上記の４０行を超える大量のソースコードで，原告は，①のアルゴリズムを実現するために，ファイルの正当性のチェックや，様々なアクセスに対応するためのファイルの切り替え処理を行えるようにソースコードを組み合わせている。

加えて，＜原告が選択したソースコード＞に複数記載のある「Ｔｒｙ～」，「Ｃａｔｃｈ～」の箇所で，原告は，例外処理を行えるようにソースコードを組み合わせている。具体的には，低レベルのファイル操作を行うにあたって，エラーが発生した場合は，それが致命的なエラーとなり，プログラムを停止することを余儀なくされることから，このような事態を回避するために，原告は上記「Ｔｒ

ｙ～」，「Ｃａｔｃｈ～」の箇所でエラー処理を行っている。エラー処理は，システムに任せるという方法も取れるところ，原告は，独自の考えから，プログラムでエラー処理ができるようにソースコードを組み合わせている。

以上より，原告は，①を実現するために，ファイルの正当性チェック，ファイルの切り替え，エラー処理等の複雑な機能を実現するソースコードを組み合わせており，これらについて原告の独自の考えによる組み合わせを織り交ぜていることから，原告の個性が発揮されていることは明らかである。

よって，創作性が認められる。

（３）　小括

甲２８号証の１頁から始まる「clsAccessMdb.vb」のソースコードに関するものだけでも，上記のように多様な選択肢の中から，原告の個性が発揮されたソースコードが構成されている。原告は，その他のクラスについても，同様の方法でソースコードを作成していることから，本件プログラム１全体に選択肢の幅があり，原告の表現上の創作性が表れているといえる。

よって，本件プログラム１に著作物性が認められる。

（４）　被告準備書面５第１に関する反論

ア　被告は，オブジェクト指向プログラミング手法でのクラスの仕組みを示しているにすぎず，創作性が認められないと主張する。

しかし，プログラム上では多様な処理があり，どの処理をクラス化するかという点についても多様な選択肢があり，クラスの設計をどうするかによってソースコードの組み合わせは異なってくるため，ありきたりな表現にはならない。

したがって，本件プログラム１に創作性が認められる。

イ　また，被告は，ファイルの正当性のチェックや切り替えの処理は，構造化プログラミングとしてありきたりな処理であると主張する。

この点，構造化プログラミングとは，命令の構成の仕方に関する計画のことを指し，ファイルの正当化チェックや切り替えは，計画とは別次元の処理の問題である。そのため，そもそも構造化プログラミングの問題とファイルの正当性のチェックや切り替えの処理については関係しない。

そして，被告は，プログラムの実行中に発生したエラーをプログラム内で処理するようプログラミングすることは常識的なことをも主張するが，プログラムの中でエラーチェックをするかしないかもプログラマーの裁量で決めることができ，かつエラーチェックの方法自体もプログラマーによって個性が現れる。具体的には，上記の本件プログラム１のエラーチェックは，ファイルへアクセスする直前で行うことも考えられるが，基本的にエラーチェックはプログラム起動時に行われているためファイルにアクセスする段階でエラーになることは考えられない。また，仮にファイルへアクセスするタイミングでエラーが発生したとしても，その段階だとパソコンの前に人がいない可能性が想定されたことから，エラー回復が望めないという状況であった。

したがって，原告はエラーが発生することが想定され，かつ人がエラーに対処することが出来るプログラム起動時にエラーチェックを行うようソースコードを組み合わせた。

以上より，エラーチェックの処理１つをとっても，プログラムの特性やプログラムを利用する者の状況を踏まえて個別に対応する必

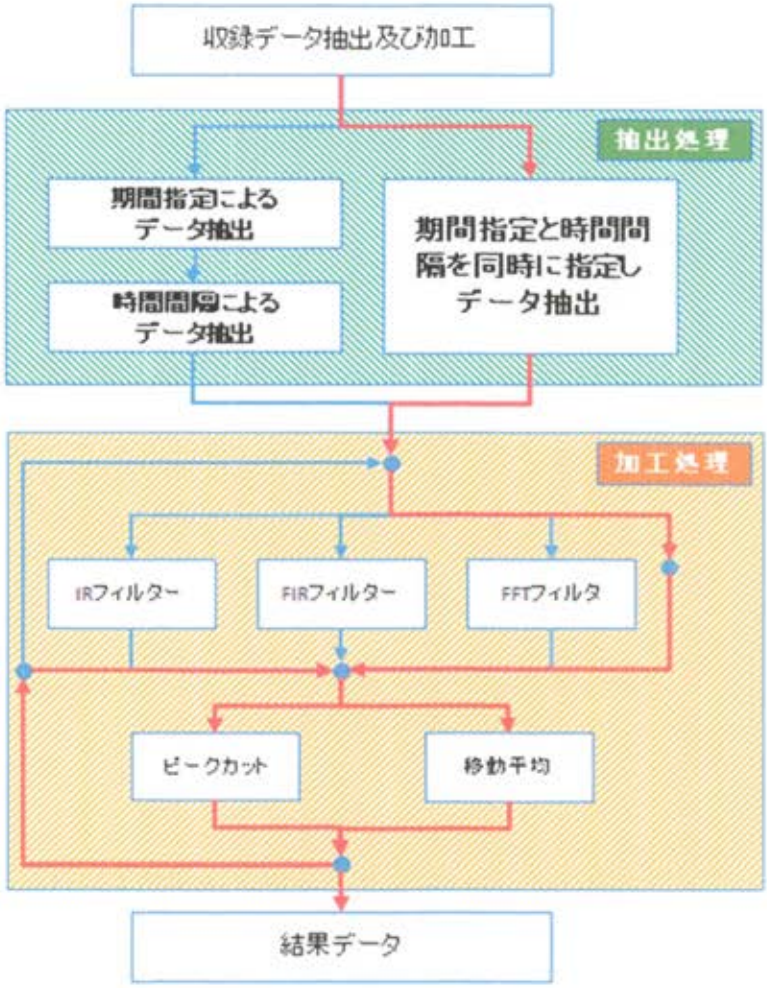
要があったことから，ありきたりな表現にはならない。

2　本件プログラム２の著作物性に関する主張の補充

本件プログラム２について，完全なソースコードとして甲２９号証を提出し，本件プログラム２について主張を補充する。

本件プログラム２の代表的な処理としては，①本件プログラム１で収録されたデータを元に，高圧室作業の開始時刻，終了時刻，作業時間及び最高圧力を探索する処理（第３準備書面第２の３（２）），②①で探索された作業時間及び最高圧力を元に減圧計画を選出する処理（第３準備書面第２の３（３）），③選出された減圧計画を元にグラフを表示させてリアルタイム画面で減圧工程を確認（第３準備書面第２の３（４）），④上記の内容を報告書にまとめてエクセルで出力する（第３準備書面第２の３（５）），⑤各作業員の高圧室内作業時間の管理（第３準備書面第２の３（１））等がある。以下，②（第３準備書面第２の３（３）に関するソースコード）について特に述べる。以下で述べるソースコードの選択肢を＜図２＞で図示する。

＜図２＞



（１）　原告は，本件プログラム１で測定されたデータを読み込む方法として，データベースを操作するＳＱＬ文（データベースに指示を出すための命令文）を作成している。ＳＱＬ文には，条件の指定方法に多様な選択肢があるところ，選択肢のうちの一部として以下のように処理を分けて，１段階目では期間内のデータを全て読み込み，２段階目で測定間隔の対象となるデータを抜き出すようにソースコードを構成する方法がある。

＜選択肢１＞

" WHERE `日時`>=#" & vStaet.ToString("yyyy/MM/dd HH:mm:ss") & "# and " &

"`日時`<=#" & vEnd.ToString("yyyy/MM/dd HH:mm:ss") & "#"

具体的には，１段階目で，期間内データを読み込むプログラムソースコード及び上記のソースコードを条件とするＳＱＬ文を実行し，対象期間内の全データをテーブルとして取得し，２段階目に１段階目で取得したデータをループですべてスキャンして測定間隔に見合うデータかを判定し，対象であれば採用する処理を行うことになる。

これらは，データベース処理であるＳＱＬ文を最小の記述にして，ＶＢ言語での記述を主体にするものである。

また，ＳＱＬ文には，期間指定の条件文としてBETWEEN命令があり，処理内容としては＜選択肢１＞と同じではあるが，視覚を重視して，以下のようなＳＱＬ文を作成するという選択肢を取ることもできる。

＜選択肢１´＞

" WHERE `日時` BETWEEN #" & vStaet.ToString("yyyy/MM/dd HH:mm:ss") & "#, #" & vEnd.ToString("yyyy/MM/dd HH:mm:ss") & "#"

上記＜選択肢１＞は，柔軟性があるプログラミングであり，以下の＜原告が作成したソースコード＞は，柔軟性に欠けてはいるものの，効率化及び高速化が望めるプログラミングであった。原告としては，プロセスを固定化し高速化を実現できるようなプログラミングを行いたい考えていたため，複雑な指令文になってしまうという点はあるが，一層の効率化及び高速化を図ることを重視して以下のＳＱＬ文を作成した（甲２９号証　５８頁２６行目乃至５９頁５９行目）。

＜原告が作成したソースコード＞

```
Public Function MeasureDataReadDB(ByVal vRecorderNo As Integer, ByVal
vStact As Date, ByVal vEnd As Date, Optional ByVal vPitch As eTimePitch =
eTimePitch.etpAll, _
                                        Optional ByVal vFlag As Integer = 0) As
clsMeasureData()
    Dim mTable As DataTable, mSql As String, mTableName As String
    Dim mRecorderInfo As clsRecorderInfo
    Dim mSql_WHERE As String
    Dim mMeasData() As clsMeasureData
    dbOpen(vRecorderNo)
    If vRecorderNo < 1 Or vRecorderNo > 10 Or IsNothing(pvDB) = True Then
      ReDim mMeasData(0)
      Return mMeasData
    End If
    mRecorderInfo = Me.pvRecorderInfo(vRecorderNo)
    mTableName = "MeasureData" & vRecorderNo.ToString("00")
    mSql = "select * from `" & mTableName & "`"
    mSql_WHERE = " WHERE `日時`>=#" & vStaet.ToString("yyyy/MM/dd
HH:mm:ss") & "# and " & "`日時`<=#" & vEnd.ToString("yyyy/MM/dd
HH:mm:ss") & "#"
    Select Case vPitch
      Case eTimePitch.etpMinute01     '1分
        mSql_WHERE &= " and SECOND(日時)=0"
      Case eTimePitch.etpMinute02     '2分
```

```
        mSql_WHERE &= " and ("
        mSql_WHERE &= "MINUTE(日時)=0 or MINUTE(日時)=2 or
MINUTE(日時)=4 or MINUTE(日時)=6 or MINUTE(日時)=8"
        mSql_WHERE &= " or MINUTE(日時)=10 or MINUTE(日時)=12 or
MINUTE(日時)=14 or MINUTE(日時)=16 or MINUTE(日時)=18"
        mSql_WHERE &= " or MINUTE(日時)=20 or MINUTE(日時)=22 or
MINUTE(日時)=24 or MINUTE(日時)=26 or MINUTE(日時)=28"
        mSql_WHERE &= " or MINUTE(日時)=30 or MINUTE(日時)=32 or
MINUTE(日時)=34 or MINUTE(日時)=36 or MINUTE(日時)=38"
        mSql_WHERE &= " or MINUTE(日時)=40 or MINUTE(日時)=42 or
MINUTE(日時)=44 or MINUTE(日時)=46 or MINUTE(日時)=48"
        mSql_WHERE &= " or MINUTE(日時)=50 or MINUTE(日時)=52 or
MINUTE(日時)=54 or MINUTE(日時)=56 or MINUTE(日時)=58"
        mSql_WHERE &= ") and SECOND(日時)=0"
    Case eTimePitch.etpMinute05     '5分
        mSql_WHERE &= " and ("
        mSql_WHERE &= "MINUTE(日時)=0 or MINUTE(日時)=5 or
MINUTE(日時)=10 or MINUTE(日時)=15"
        mSql_WHERE &= " or MINUTE(日時)=20 or MINUTE(日時)=25 or
MINUTE(日時)=30 or MINUTE(日時)=35"
        mSql_WHERE &= " or MINUTE(日時)=40 or MINUTE(日時)=45 or
MINUTE(日時)=50 or MINUTE(日時)=55"
        mSql_WHERE &= ") and SECOND(日時)=0"
    Case eTimePitch.etpMinute10     '10分
        mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=10 or
```

```
MINUTE(日時)=20 or MINUTE(日時)=30 or MINUTE(日時)=40 or MINUTE(日
時)=50) and SECOND(日時)=0"
        Case eTimePitch.etpMinute20     '20分
          mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=20 or
MINUTE(日時)=40) and SECOND(日時)=0"
        Case eTimePitch.etpMinute30     '30分
          mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=30) and
SECOND(日時)=0"
        Case eTimePitch.etpHour01       '1時間
          mSql_WHERE &= " and MINUTE(日時)=0   and SECOND(日時)=0"
        Case eTimePitch.etpHour02       '2時間
          mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=2 or HOUR(日
時)=4 or HOUR(日時)=6 or HOUR(日時)=8 or HOUR(日時)=10 or HOUR(日
時)=12 or HOUR(日時)=14 or HOUR(日時)=16 or HOUR(日時)=18 or HOUR(日
時)=20 or HOUR(日時)=22) and MINUTE(日時)=0 and SECOND(日時)=0"
        Case eTimePitch.etpHour03       '3時間
          mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=3 or HOUR(日
時)=6 or HOUR(日時)=9 or HOUR(日時)=12 or HOUR(日時)=15 or HOUR(日
時)=18 or HOUR(日時)=21) and MINUTE(日時)=0 and SECOND(日時)=0"
        Case eTimePitch.etpHour06       '6時間
          mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=6 or HOUR(日
時)=12 or HOUR(日時)=18) and MINUTE(日時)=0 and SECOND(日時)=0"
        Case eTimePitch.etpHour12       '12時間
          mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=12) and
MINUTE(日時)=0 and SECOND(日時)=0"
```

```
        Case eTimePitch.etpDay01        '1 日
          mSql_WHERE &= " and HOUR(日時)=0 and MINUTE(日時)=0 and
SECOND(日時)=0"
      End Select
      If vFlag = 1 Then
        mSql_WHERE &= " and Flag>0"
      End If
      mSql &= mSql_WHERE
      mSql &= " ORDER BY `日時` ASC"
      pvDB.BeginTransaction()
      mTable = pvDB.ExecuteSql(mSql)
      pvDB.CommitTransaction()
      Dim mCount As Integer = mTable.Rows.Count
      If mCount > 0 Then
        ReDim mMeasData(mCount)
        For i = 1 To mCount
          mMeasData(i) = New clsMeasureData
          With mMeasData(i)
            pvDB.GetTableData(mTable, mMeasData(i).MeasDate, i - 1, "日時")
            For mChNo = 1 To 10
              pvDB.GetTableData(mTable, mMeasData(i).MeasData(mChNo), i -
1, "Data" & mChNo.ToString("00"))
            Next
            pvDB.GetTableData(mTable, mMeasData(i).Alarm, i - 1, "Alarm")
            pvDB.GetTableData(mTable, mMeasData(i).Flag, i - 1, "Flag")
```

```
        End With
      Next
    Else
      ReDim mMeasData(0)
    End If
    dbClose()
    Return mMeasData
  End Function
```

（２）　原告が作成した上記ソースコードは，データベースの読み込みを一度の処理で済ませるために，大量のソースコードとなっている。その中でも特に，測定間隔の指定について，原告は以下のような条件文を作成している。

＜上記原告が作成したソースコードのうち測定間隔の指定についての条件文＞

```
    Select Case vPitch
      Case eTimePitch.etpMinute01     '1分
        mSql_WHERE &= " and SECOND(日時)=0"
      Case eTimePitch.etpMinute02     '2分
        mSql_WHERE &= " and ("
        mSql_WHERE &= "MINUTE(日時)=0 or MINUTE(日時)=2 or
MINUTE(日時)=4 or MINUTE(日時)=6 or MINUTE(日時)=8"
        mSql_WHERE &= " or MINUTE(日時)=10 or MINUTE(日時)=12 or
MINUTE(日時)=14 or MINUTE(日時)=16 or MINUTE(日時)=18"
        mSql_WHERE &= " or MINUTE(日時)=20 or MINUTE(日時)=22 or
```

```
MINUTE(日時)=24 or MINUTE(日時)=26 or MINUTE(日時)=28"
        mSql_WHERE &= " or MINUTE(日時)=30 or MINUTE(日時)=32 or
MINUTE(日時)=34 or MINUTE(日時)=36 or MINUTE(日時)=38"
        mSql_WHERE &= " or MINUTE(日時)=40 or MINUTE(日時)=42 or
MINUTE(日時)=44 or MINUTE(日時)=46 or MINUTE(日時)=48"
        mSql_WHERE &= " or MINUTE(日時)=50 or MINUTE(日時)=52 or
MINUTE(日時)=54 or MINUTE(日時)=56 or MINUTE(日時)=58"
        mSql_WHERE &= ") and SECOND(日時)=0"
    Case eTimePitch.etpMinute05     '5分
        mSql_WHERE &= " and ("
        mSql_WHERE &= "MINUTE(日時)=0 or MINUTE(日時)=5 or
MINUTE(日時)=10 or MINUTE(日時)=15"
        mSql_WHERE &= " or MINUTE(日時)=20 or MINUTE(日時)=25 or
MINUTE(日時)=30 or MINUTE(日時)=35"
        mSql_WHERE &= " or MINUTE(日時)=40 or MINUTE(日時)=45 or
MINUTE(日時)=50 or MINUTE(日時)=55"
        mSql_WHERE &= ") and SECOND(日時)=0"
    Case eTimePitch.etpMinute10     '10分
        mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=10 or
MINUTE(日時)=20 or MINUTE(日時)=30 or MINUTE(日時)=40 or MINUTE(日
時)=50) and SECOND(日時)=0"
    Case eTimePitch.etpMinute20     '20分
        mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=20 or
MINUTE(日時)=40) and SECOND(日時)=0"
    Case eTimePitch.etpMinute30     '30分
```

```
        mSql_WHERE &= " and (MINUTE(日時)=0 or MINUTE(日時)=30) and
SECOND(日時)=0"
    Case eTimePitch.etpHour01       '1時間
      mSql_WHERE &= " and MINUTE(日時)=0   and SECOND(日時)=0"
    Case eTimePitch.etpHour02       '2時間
      mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=2 or HOUR(日
時)=4 or HOUR(日時)=6 or HOUR(日時)=8 or HOUR(日時)=10 or HOUR(日
時)=12 or HOUR(日時)=14 or HOUR(日時)=16 or HOUR(日時)=18 or HOUR(日
時)=20 or HOUR(日時)=22) and MINUTE(日時)=0 and SECOND(日時)=0"
    Case eTimePitch.etpHour03       '3時間
      mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=3 or HOUR(日
時)=6 or HOUR(日時)=9 or HOUR(日時)=12 or HOUR(日時)=15 or HOUR(日
時)=18 or HOUR(日時)=21) and MINUTE(日時)=0 and SECOND(日時)=0"
    Case eTimePitch.etpHour06       '6時間
      mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=6 or HOUR(日
時)=12 or HOUR(日時)=18) and MINUTE(日時)=0 and SECOND(日時)=0"
    Case eTimePitch.etpHour12       '12時間
      mSql_WHERE &= " and (HOUR(日時)=0 or HOUR(日時)=12) and
MINUTE(日時)=0 and SECOND(日時)=0"
    Case eTimePitch.etpDay01        '1日
      mSql_WHERE &= " and HOUR(日時)=0 and MINUTE(日時)=0 and
SECOND(日時)=0"
  End Select
```

また，本件プログラム２についても，コストパフォーマンスのために Access（アクセス）のデータベースを使用しているところ

もあり，一定以上複雑なＳＱＬ文には制限があった。そのため，制約がある中，原告は測定間隔の指定について上記のような工夫した条件文を作成し，一度の処理でデータの読み込みを実現していることから，他のプログラマーが作成しても同様の記載になるものではない。

（３）　さらに，本件プログラム２では，本件プログラム１から読み込んだデータの中から，高圧室内の気圧データである圧力データを抜き出す処理を行っている。当該処理については，本件プログラム１からデータを読み込むのと同時に行うという処理も考えられるが，原告はデバッグ（プログラム内のバグを見つけて改修する作業）の利便性を考え，データの読み取りと別処理にするよう以下のとおりソースコードを工夫している（甲２９号証１０９頁５３行目乃至５５行目）。

＜圧力データ抜き出しのためのソースコード＞

```
For i = 0 To mCount - 1
mData2.DataAdd(New clsDataSet(mMeasData(i + 1).MeasDate, mMeasData(i +1).MeasData(mPressChNo)))
Next
```

そして，上記圧力データを抜き出した後，作業員の作業時間を求める処理が必要となるところ，生の測定データには，欠陥やノイズが生じるため（第３準備書面別紙１参照），解析しやすいデータに加工する必要があった。そして，加工の処理については，＜図２＞の「加工処理」の箇所に記載があるとおり，「ＩＲフィルター」，「ＦＩＲフィルター」，「ＦＦＴフィルター」，「ピークカット」「移動平均」等の選択肢が存在する。加工処理を構

成するためにはこれらの選択肢から様々な組み合わせが選べるところ，「ＩＲフィルター」，「ＦＩＲフィルター」，「ＦＦＴフィルター」は，複雑な係数を与える必要がある一方で，思うような効果が得られないことから原告は採用できないと判断した。そこで，原告は測定データを加工するために，プログラマーとしての経験から以下のソースコードを作成し，「ピークカット」と「移動平均」の処理を組み合わせた（甲２９号証　１０９頁５６行目乃至１１０頁１４行目）。

＜加工のためのソースコード＞

```
Dim t As Date = mDataStart
Dim tw As TimeSpan
Dim mPressData As Double
tw = mDataEnd - mDataStart
Dim mAveTime As Integer = numAveTime.Value
Dim mTimeSpan As TimeSpan = New TimeSpan(0, 0, mAveTime)
mCount = (tw.Ticks / mTimeSpan.Ticks)
HiPressInfo.計測Data(WorkNo).Clear()
While t <= mDataEnd
mPressData = mData2.GetAveData(t, -mTimeSpan.TotalSeconds / 2,
mTimeSpan.TotalSeconds / 2)
If IsNullData(mPressData) = False Then
HiPressInfo.計測Data(WorkNo).DataAdd(t, mPressData)
End If
t += mTimeSpan
End While
```

HiPressInfo.ReadStart(WorkNo) = HiPressInfo.計測Data(WorkNo).TimeTop

HiPressInfo.ReadEnd(WorkNo)=HiPressInfo.計測 Data(WorkNo).TimeBottom

以上のとおり，本件プログラム２は，測定データの解析の処理だけでも，ソースコードに多様な選択肢があることから，本件プログラム２全体に選択の幅があることは明らかである。

加えて，原告が作成した上記ソースコードは，大量かつ複雑で，随所に原告の独自の考えに基づく構成が取られており，ありふれたものではなく，作成者によって相違が生じるものといえる。

よって，本件プログラム２に，著作物性が認められる。

以　上