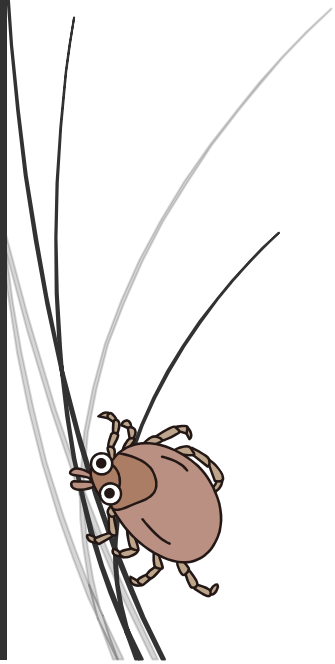
2016年9月25日

# はじめてのバグ票システム～導入実践ガイド

第1.0版



NaITE - バグ票システム検討 SIG

NaITE（長崎ＩＴ技術者会）

# 巻頭言

2016年9月25日，半年のNaITE-バグ票検討SIGの活動成果物として「はじめてのバグ票システム　～導入実践ガイド」の初版を公開することができ，嬉しく思っています。

ことの始まりは，本ガイドの著者の一人でありSIGメンバでもあるすずきしょうごさんとの何気ない会話でした。もともと本ガイドを一人でまとめたいと考えており，そのような話をとある飲み会で話題にしたところ，すずきさんに興味を持っていただき彼が主催しているバグ票ワーストプラクティス検討プロジェクト殿にも協力していただいたらどうかという話になりました。ただし飲み会での会話でもありますので，一度話を持ち帰っていただき内部でご検討いただきました。結果，本ガイドの作成に協力したいというお申し出をいただき，本SIGはバグ票ワーストプラクティス検討プロジェクト[1]殿とNaITE（長崎ＩＴ技術者会）[2]の共同SIGという形をとることとなりました。ベストエフォートかつスモールスタートアプローチでの活動でありますが，こうしてガイドの公開に至っています。

[1] バグ票ワーストプラクティス検討プロジェクトWebサイト
https://sites.google.com/site/swworstpracticesite/

[2] NaITE（長崎IT技術者会）公式サイト
http://naite.swquality.jp/

このバグ票システムに関するガイドをまとめる活動を始めたのは，世の中にバグ票システムを導入するにあたり必要となる企画から運用までの一連活動の概要がまとめられた書籍や文書といった情報が見つけられず，自身でまとめる必要が生じたことがきっかけです。

バグ票の書き方やOSSツール・システムのインストール・使い方に関する書籍や情報は多く見つけられるものの，先ほど挙げたバグ票システム導入にあたっての企画から設計やツールへの設定実装，テストといった稼働までの活動内容に焦点を当てた情報はあまり見当たりません。

「バグ票ツール・システムのインストール＝バグ票システムの検討」と捉えられていることが多いように感じています。このため導入にあたっての工数に理解が得られないなどということもたびたびありました。どう大変なのか，工数がかかるかを理解してもらうために「とりあえずこの本読んでみて」というようなこともできません。身の回りには軽くていた結果，実際にやってみると導入に苦労する人もたびたびみかけてきました。

そこでいつもどおり「ないなら作ればいいじゃない」精神で情報をまとめることとしました。当初，自身のみの利用を想定していただのですが，どうせ個人の余暇時間で作るのであればそれらを広く公開し同様な悩みを持つ技術者や必要としている技術者の方々にお役立ちできればと考えました。また，教育資料としてお役立ていただくのも良い事です。

そうしてNaITEにバグ票システム検討SIGを作り，バグ票ワーストプラクティス検討プロジェクト殿から岡内佑樹さんに識者として本ガ

イドの執筆にご協力いただけることになりました。とはいうものの，SIG メンバは３名での活動となるため，はじめから大きなことはできません。そこで小さく始めて徐々に内容や情報を付随する活動を充実させていくという方針としました。活動は各自の本業や他のコミュニティ活動の間をぬって実施しました。茨城県大洗にてのブレーンストーミング合宿をキックオフとし，集まれるメンバで一月に一度程度打ち合わせを実施しながら，ガイド作成を進めてきました。

以上を経て公開した本ガイドですが，改版を前提としています。しかし，まだ情報は少ないとはいえ，バグ票システム導入に必要な段取り・プロセス等の情報を必要としている方々に何かしらの情報をご提供できているのではと思います。

本ガイドは今後も改版をする他，ガイドを使った教育勉強会等も検討してまいりたいと考えておりますので，ガイドとあわせてご活用いただければと思います。直近では 2016 年 10 月 7 日に長崎市内で開催する「長崎 Software Quality and Development Gathering 2016[3]」にて本ガイドをベースとして発表を実施いたします。こちらも是非参加をご検討いただけますと幸いです。

[3] 長崎 Software Quality and Development Gathering 2016 詳細ページ
http://nagasaki-it-engineers.connpass.com/event/21303/

本ガイドには執筆を担当された岡内佑樹さんとすずきしょうごさんのご経験や知見がふんだんに盛り込まれています。お二人のお力があってこそ，このガイドは公開することができています。ここに感謝申し上げます。また，読者におかれましては是非本ガイドを，よりよいソフトウェア開発の実現ご活用いただければと存じます。

2016年9月25日

NaITE（長崎ＩＴ技術者会）　代表　池田 暁

# 本ガイドの狙い目的

本ガイドは，これまでなんらかのバグ票システムは使ったことがあるけれど自分では導入したことがないという方に，導入の企画から稼働までのガイドとなる情報を提供することを目的としています。

# 本ガイドのスコープ

本ガイドは「バグ票システムの導入検討から稼働準備までのプロセスおよび必要な活動について解説すること，および参考となる情報の紹介」をスコープとしています。バグ管理活動そのものや計測活動などについては今後の改版で情報を追加していく予定ですが，現在（第1.0版）では触れておりませんので，それらの情報が必要な方は他の専門書籍等をご参照ください。

# 本ガイドの対象読者

本ガイドは，以下の方を対象読者として作成しています。

- バグ票システムに関心のある方
- バグ票システムを稼働させたいと考えている方
- バグ票システムの企画から稼働までの基本的なプロセスを知りたい方

# NaITE とは

NaITE（長崎ＩＴ技術者会）とは，長崎になんらかの縁や関わり，興味がある IT 技術者および IT 技術利用者を中心とした技術コミュニティです。

2015 年 4 月にコミュニティとして設立し，ソフトウェア品質技術やソフトウェア開発技術の話題を中心に，長崎での技術イベントや全国各地場所を問わない勉強会の開催，SIG（研究会）など活動しています。

本グループは主に以下の方を想定していますが，長崎に関心のある方であればどなたでもご参加いただけます。

- 長崎に在住もしくは在住していた IT 技術者
- 長崎出身の IT 技術者
- 長崎に縁や興味がある方

NaITE では参加者の自主性を大切に活動しています。多数のご参加をお待ちしております。

詳しくは，公式サイトや関連ページをご参照ください。

NaITE（長崎ＩＴ技術者会公式） 公式サイト

NaITE（長崎ＩＴ技術者会公式） Facebook ページ

NaITE（長崎ＩＴ技術者会公式） Facebook 参加者交流グループ

# バグ票システム検討 SIG とは

バグ票システム検討 SIG は，バグ票システムを立ち上げるのにあたって必要な事を整理しガイドなどにまとめることを目的とした研究会活動です。バグ票システムを立ち上げるまでのプロセスやアクティビティ，コツやノウハウなどについて，知識や情報を両団体や参加者が持ち寄り，議論を通してまとめています。

http://naite.swquality.jp/?page_id=40

■活動概要

- バグ票システムの立ち上げにおける様々なことを議論しながらガイド等にまとめる

■活動期間，活動形態

- 2016 年 4 月～2017 年 3 月を予定
- ML および定例会や合宿

■想定する成果物

- ガイド等のドキュメント
- 長崎 QDG2016 での発表
- その他，セミナーなどの教育の実施

# SIG メンバ および 本ガイド執筆者

バグ票システム検討SIGおよび本ガイド執筆活動は以下のメンバで行ないました。

■ SIGリーダ

池田 暁 (NaITE)

■ SIGメンバ

岡内 佑樹 (バグ票ワーストプラクティス検討プロジェクト)

すずき しょうご (NaITE, バグ票ワーストプラクティス検討プロジェクト)

# 謝辞

本ガイドの執筆にあたり以下の方々に，多くのご協力をいただきました。この場を借りてお礼申し上げます。

- バグ票ワーストプラクティス検討プロジェクト殿
- NaITE運営スタッフ

# 本ガイドの利用およびご注意

本ガイドについては以下をご参照・ご理解のうえ，ご利用下さい。

- 自学自習や勉強会など非営利の利用に制限はありません。ご自由にお使いください
- 有料セミナーでのテキスト利用など，営利目的の利用はご遠慮願います（ただし，企業内勉強会等，直接の営利目的でなければご利用いただけます）
- 本資料に登場する会社名や製品名等は一般に各社の商標です
- 本資料の利用により発生したいかなる損害やトラブルについても，NaITE（長崎 IT 技術者会）は一切の責任を負いかねますのであらかじめご了承ください

# 目次

巻頭言 ..... i

本ガイドの狙い目的 ..... v

本ガイドのスコープ ..... v

本ガイドの対象読者 ..... v

NaITE とは ..... vi

バグ票システム検討 SIG とは ..... vii

SIG メンバ および 本ガイド執筆者 ..... viii

謝辞 ..... viii

本ガイドの利用およびご注意 ..... ix

1　バグ票およびバグ票システムとは ..... 2

1.1　バグとは ..... 2

1.2　バグ票とは ..... 3

1.3　バグ票システムとは ..... 4

1.4　バグ票およびバグ票システムを利用する価値 ..... 5

2　バグ票システムの導入検討から稼働準備までのプロセス ..... 8

3　バグ票システムの企画 . . . . . 10

3.1　バグ票システムの目的 . . . . . 11

3.2　誰が使うのか . . . . . 12

3.3　いつ使うのか . . . . . 13

3.4　どこで使うのか . . . . . 14

3.5　いつまで使うのか . . . . . 14

3.6　まとめ . . . . . 15

4　バグ票システムの設計 . . . . . 16

4.1　バグ票のワークフローの設計 . . . . . 17

4.1.1　ワークフローの設計に利用できる技法 . . . . . 20

4.2　バグ票項目の設計 . . . . . 21

4.2.1　バグ票で扱う情報の設計で利用できる技法 . . . . . 24

4.3　ロールおよび権限の設計 . . . . . 25

4.3.1　ロールおよび権限の設計で利用できる技法 . . . . . 26

4.4　必要な機能などの整理 . . . . . 27

4.5　動作環境や保守方法の設計（ネットワークやハードウェア等の稼働環境） . . . . . 28

4.6　まとめ . . . . . 29

5　バグ票システムの実装 . . . . . 30

5.1　ツール・システムの選定 . . . . . 31

5.1.1　代表的なツール・システムの紹介 . . . . . 33

5.2　選定したツール・システムに対応した実装仕様の作成 . . 39

5.3　実装仕様の実装（設定/反映） . . . . . 39

5.4　まとめ . . . . . 40

6　バグ票システムのテスト . . . . . 42

6.1　設定値が正しく設定されているかを確認する . . . . . 43

6.2　ワークフローが正しく動作するかを確認する . . . . . 44

6.3　実運用を想定したテストを行なう . . . . . 45

6.4　その他のテスト . . . . . 45

6.5　まとめ . . . . . 46

7　稼働準備 . . . . . 47

7.1　システムに関するドキュメントの整備 . . . . . 48

7.2　説明会の実施 . . . . . 49

7.3　利用者のトレーニング . . . . . 49

7.4　・まとめ . . . . . 50

8　バグ票に関してのワーストプラクティス . . . . . 52

8.1　バグピンポン . . . . . 54

8.2　バベルの塔 . . . . . 55

8.3　「このぐらいわかってくれよ」症候群 . . . . . 57

8.4　書き手理解不足 . . . . . 58

8.5　まとめ . . . . . 59

9　バグ票を活用した活動 . . . . . 61

9.1　バグ分析，原因分析 . . . . . 61

9.2　バグの予測 . . . . . 64

9.3　バグ票システム自体の改善 . . . . . 65

9.4　まとめ . . . . . 65

10　おわりに . . . . . 68

11　参考文献・URL . . . . . 69



# 第 I 部
# バグ票およびバグ票システムとは

第 I 部では，バグ票およびバグ票システムについて，簡単に解説します。

# 1 バグ票およびバグ票システムとは

本章は，第ＩＩ部でバグ票システムの企画から稼働準備を解説するにあたり，本ガイドにおけるバグ票およびバグ票システムを定義するものです。

## 1.1 バグとは

もし，あなたが「『バグ』とはなんでしょうか？」と聞かれた場合どのように回答しますか？ バグと似たような用語としては以下のようなものがあります。

- 欠陥
- 不具合
- 障害
- 不良
- 事故
- 故障

「バグ」という用語は，ソフトウェア開発においてよく聞く用語の1つであり，ソフトウェア開発にかかわる人で聞いたことがない人はほとんどいないと思います。しかし，ソフトウェア開発において「バグ」を一意に定義している規格や公式な定義はありませんし，先ほど挙げた用語を同じような意味で使っている企業やプロジェクトも多い

でしょう。また，同じ単語でも微妙に意味が異なる使われ方がされていることも多いようです。これらはバグの認識を違えたり誤解を生じたりといった問題を生じさせることになります。本ガイドの読者についても例外ではないため，一意に定義しておく必要があります。

以降，本ガイドではバグを「意図した，あるいは，期待した振る舞いで無いため調査が必要な事象」という意味で使用します。

## 1.2 バグ票とは

バグ票はバグに関する情報をまとめた帳票（情報の塊）です。バグ票は利用者の組織やプロジェクトでの利用方法を考慮したうえで必要な項目を設定します。たとえば，ISTQB[4] Foundation Level のシラバス（JSTQB[5]による日本語版[6]）には次の項目が挙げられています。

- 作成日付，作成した組織，作成者
- 期待した結果と実行結果

[4] ISTQB(International Software Testing Qualifications Board)
http://www.istqb.org/

[5] JSTQB（Japan Software Testing Qualifications Board)
http://jstqb.jp/

[6] JSTQBによるISTQB Foundation Level シラバス 日本語版の掲載ページ
http://jstqb.jp/syllabus.html

- インシデントログ，データベースのダンプ，スクリーンショットなどのインシデントの再現および解決を可能にする詳細な記述
- 修正の緊急度／優先順位
- インシデントの状態(例えば，オープン，次バージョンへ延期，重複，修正待ち，再テスト待ち，解決済み など)
- 結論，アドバイス，承認
- 問題を摘出したテストケース仕様の ID 番号など参照情報

バグ票を活用することで，バグそのものの情報を確認することは勿論，記録やエビデンスとすることができます。プロジェクト中やプロジェクト後に分析することでリリース判定の材料情報としたりプロセス改善に生かしたりすることもできます。

## 1.3 バグ票システムとは

バグ票はバグの情報を集めた帳票ですが，バグ票システムは管理するためのシステムです。システムは「BTS（バグ・トラッキング・システム）」と呼ばれ，よく知られている物として Redmine[7]や Trac[8]，Mantis[9]などがあります。

[7] Redmine http://www.redmine.org/

[8] Trac https://trac.edgewall.org/

[9] Mantis https://www.mantisbt.org/

バグ票システム（BTS）を使うことで，バグ票を集中管理できるようになるほか，システムの機能によりバグ票の情報の検索や加工，分析などが手軽に行えるようになります。これにより，バグ票を効率的に活用できるようになるほか，システムの力を借りてより高度な利用を実現することができるようになります。現在のソフトウェア開発ではバグ票システムを利用することは常識と言える状況となっています。

## 1.4 バグ票およびバグ票システムを利用する価値

これまで述べてきたように，ソフトウェア開発中に発見・検出したバグを管理することはいまや必須の活動です。これを実施しなければ，バグそのものの内容詳細を把握することができません。また，蓄積したバグ情報を分析して品質状況を把握することができません。

バグ票を使わずに，バグを管理することを考えてみましょう。

バグ報告者などからのバグの情報は，口頭やメール，あるいはメッセンジャなどで伝えられるとします。その場合，バグ報告者によっては情報の伝え忘れなどもあり断片的な（欠損した・不確かな）情報となってしまうことになります。複数の報告者がいる場合，それぞれからばらついた情報や異なった情報が開発者に伝えられることが起きます。そうすると，情報を受け取った開発者はバグの再現や調査に必要な情報を改めてメールや直接出向いて確認することになり，コミュニケーションのコストが大きくなってしまいます。また，そういった口頭やメールベースでやり取りした情報は記録されていませんから，関係者の記憶がなくなれば合わせてバグの情報も消えてしまうことになります。

バグ票は，こうした不効率を改善します。また，記録に残すことで当事者だけでなくステークホルダにバグ情報を共有することが可能となり，今後のためにプロセスやプロダクトの改善の情報として活用することも可能となります。

さらにバグ票システムを導入することで，バグの情報や状況を利用者でリアルタイムに共有でき，情報をスピーディに分析して，その情報を活用することで，プロジェクトを円滑に進めたり改善したりすることが可能になります。

そしてプロジェクトが終了したのちも，バグ票システムに蓄えられたバグの情報を活用することで，次のプロジェクトを改善することができます。バグはソフトウェア開発の弱点を表しているとみることもできることから，バグ票やバグ票システムを継続的に利用することで，組織の継続的改善にも役立てることができます。

以上が，バグ票とバグ票システムを利用する主な価値です。是非積極的に導入すると良いでしょう。



# 第ＩＩ部
# バグ票システムの
# 企画から稼働準備まで

第 II 部では，バグ票システムの導入について，その一つのモデルプロセスを示します。２章ではモデルプロセスを示し，３章以降でモデルプロセスに沿って企画から稼働準備についてそれぞれ必要となる活動や観点を解説します。

# 2 バグ票システムの導入検討から稼働準備までのプロセス

バグ票システムを導入しようと思い立つきっかけは沢山あると思います。何らかの理由によりバグ票システムの導入を検討する場合，様々なアプローチや進め方があります。

バグ票システムの導入でよく目にするのが「ひとまずなにかしらのバグ票システム（BTS）をインストールして，その場その場，その都度その都度で設定していく」というやり方です。現在よく利用されているOSSのバグ票システムなどではワンクリックやそれに近い手軽さでインストールができ，動かせる状態にするだけならば半日もあれば十分でしょう。システムが動いてしまえば，あとは使いながら随時使いにくいところを設定したり変更したりしていけば良さそうに思えます。しかしこの方法は，次のような問題をはらんでいます。

- インストールしたものの，使ってみると機能が足りなかったり目的にそぐわなかったりすることに気が付く
- 度重なるワークフローの改変でワークフローが複雑化しメンテナンスできなくなる
- 入力する情報項目の無計画な追加や削除により扱う情報項目が膨大になったり壊れたり喪失したりしてしまい，バグ票やバグ票が持つ情報一貫性がなくなってしまう
- システムにプラグインや拡張機能を追加しすぎ，それらがシステムを壊してしまう

- 利用者の拡大に伴い，システム利用者のためのサポートデスク対応の作業が増大，対応しきれなくなってしまい，利用者によって利用のばらつきが大きくなる

バグ票システムは一般に一度導入すると短くても半年，長いと数年は使い続けるものです。導入のための試行目的ならばいざ知らず，開発の標準インフラシステムとして構築する場合は，場当たりの導入ではなく，「企画・設計・実装・テスト・稼働準備」といったプロセス（図 2.1）を経て，実際の稼働につなげることが大切です。

第ＩＩ部の以降は図 2.1 に示した稼働までのプロセスにそって解説を進めます。

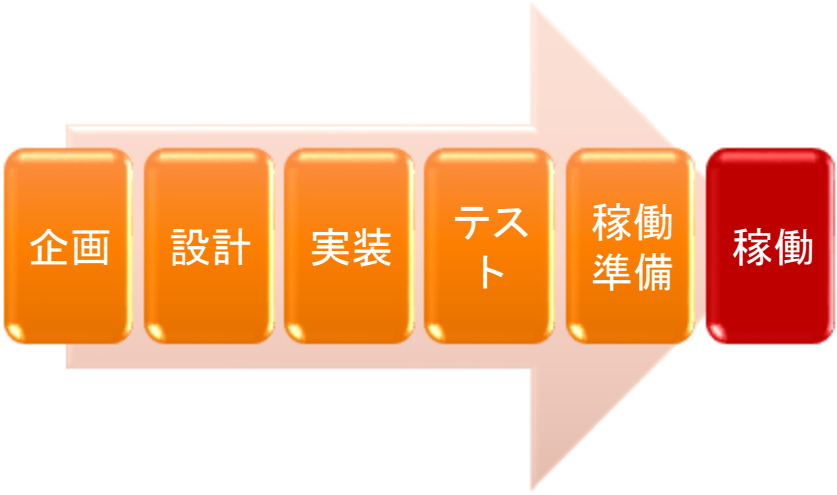



図 2.1 バグ票システムの稼働までのプロセス

# 3 バグ票システムの企画

3 章以降はバグ票システムを稼働させるためのプロセスについて解説します。3 章ではプロセスの最初である「企画」について解説します。（図 3.1）




図 3.1 バグ票システムの稼働までのプロセス（企画）

「企画」といっても何をすれば良いのでしょうか？ 本ガイドではバグ票システムの「企画」で取り扱う（決める）ものとして，以下を挙げています。

- バグ票システムの目的
- 誰が使うのか
- いつ使うのか
- どこで使うのか
- いつまで使うのか

これら以外にも挙げればきりがないのですが，最低限これくらいきめておけばバグ票システムの設計検討を進めることができます。逆に言えば，決めておかないと企画の後に実施する設計の作業がうまくいきません。

本章ではこの５項目について簡単に解説します。

## 3.1 バグ票システムの目的

あなたやチーム，組織ではこれから立ち上げようと考えているバグ票システムをどのような目的で使うのでしょうか。

紙ベースでの管理を止めたいのでしょうか？ Excel 帳票を止めたいのでしょうか？ バグの情報を一元管理したいのでしょうか？ 検索機能など便利な機能を利用したいのでしょうか？ バグの件数を管理して状況を把握したいのでしょうか？ バグの情報からバグ分析をしたいのでしょうか？ 情報から今後を予測したりプロセス以前のための情報に変換したりしたいのでしょうか?

少し挙げただけでも様々に利用目的があることがわかるかと思います。システム利用者が多くなればなるほど，目的や思惑は異なってくることも理解することができると思います。

これから立ち上げようとしているバグ票システムの目的によってシステムに求める機能は変わるでしょう。また，入力情報として必要な情報やバグ情報のライフサイクルや運用方法など様々な影響を受けることになります。目的をはっきりさせておくことはとても大切です。

## 3.2 誰が使うのか

これから稼働させようとしているバグ票システムは誰が利用するのでしょうか。

システムをインストールした人のみが使うということはまれでしょう。ソフトウェア開発プロジェクトは様々な立場やロールのメンバで構成されますが，バグ票システムもまた様々な立場やロールのメンバが利用します。開発者の場合もありますし，テスト担当者の場合もあるかもしれません。プロジェクトマネージャやSEPG部門も利用するかもしれません。QA部門やお客様も利用するかもしれません。

利用する人が変わると利用法やシステムに求める物も変わります。例えば，開発者がメインに使うとなると，バグ票にはバグの情報だけでなくデバッグに関する情報も入力したいと考えるかもしれません。テスト担当者であればバグの情報に加えて，実行したテストの情報やその他のテストに関する情報も入力したいと考えるかもしれません。プロジェクトマネージャやSEPG部門の場合，管理のための情報を入力したいと考えるかもしれません。

誰が使うのかによって，バグ票に求める情報や量，種別が変わります。またバグ票のワークフロー（ライフサイクル）も変わることになるでしょうし，システムに求める機能も変わるかもしれません。

あらかじめ誰が使うか，何を期待しているか，主に誰が使うのかなどを明らかにしておくことが必要です。

## 3.3 いつ使うのか

バグ票システムはソフトウェア開発プロセスにおいて，いつ使うのでしょうか。

バグ票システムを使う典型的なタイミングは所謂テスト工程に入ってからですが，プロジェクトが始まった直後から利用を始める場合もあるでしょうし，プロジェクトの企画が終了し開発作業が始まってからかもしれません。また，特定のフェーズのみでしか利用しないかもしれません。テスト工程でのみでしか使わないかも知れないし，出荷試験でしか利用しないかもしれません。アジャイル開発やインクリメント開発では，最初のイテレーションから使うかもしれないし，そうでないかもしれません。

このように開発プロセスにおいて「いつ使うのか」を明らかにしておく必要があります。「いつ」が決まれば，それに合わせてバグ票システムが稼働状況になるようにシステム導入計画を立てる必要がありますし，ユーザのための事前のトレーニング計画などにも影響が出ます。また，バグ票の情報項目内容やワークフローも影響を受けることになります。

また，一日の中でいつ使うでしょうか。日中でしょうか，それとも一日の最後でしょうか。夜間も使うでしょうか。バグ票システムを稼働させなければならない時間によって，保守活動が大きな影響を受けます。バックアップはどのタイミングで実施するのか，システムメンテナンスや若化をいつ実施するかなどです。また，24時間365日連続稼働といったような事になると，バグ票システムはもちろんそれをインストールハードウェアに求められる信頼性が変わってきます。

## 3.4 どこで使うのか

バグ票システムはどこで使うのでしょうか。

開発の現場でしょうか？ テストルームでしょうか？ マネージャの自席でしょうか？ また，オフショアを利用する場合は海外からの利用もあるかもしれません。ネットワークを介して使う場合もあるでしょうし，スタンドアローンやローカル環境での利用であるかもしれません。システムを利用できるのは1台かもしれませんし，複数台かもしれません。

このように「どこで使うのか」によって，バグ票システムや利用ルールに制約や制限を設定する必要が生じます。また，当然ながら，他拠点アクセスのような使い方であれば，そのための環境を整備したりする必要が生じます。これにより，ネットワーク構築やファイアウォールの設定，社内申請や輸出管理の調査が必要になります。どちらかというとインフラ構築の面が強いですが，システムの諸設定にも影響を与えます。

## 3.5 いつまで使うのか

さて，最後になりますが，このシステムはいつまで使うのでしょうか。

このプロジェクト一回きりでしょうか。それともその後も使うのでしょうか。システムとしてのライフサイクルということになりますが，これにより保守の計画は勿論，どこまでの機能を盛り込みどこまでを運用ルールで対応するかなどが変わってきます。継続的に使うという

ことは保守のコストのあり方も想定しておく必要があるかもしれません。なぜならば，最初に立ち上げたチームとその後保守するチームは異なるかもしれないからです。

## 3.6 まとめ

本章ではバグ票システムを立ち上げるにあたってまず必要な作業となる「企画」を取り上げました。バグ票システムは無計画に立ち上げると失敗する可能性が高まります。どれだけきちんと企画し，それを関係者と合意できるかが立ち上げの成否に鍵となることでしょう。ただしこのとき注意すべきは「バグ票システムの企画」であって「バグ票システムを使った開発プロセスの企画」ではないということです。本ガイドでは前者を扱っていることに注意してください。

# 4 バグ票システムの設計

３章ではバグ票システムの企画を取り上げました。バグ票システムを立ち上げるためにまず目的や制約といったことを整理しておくことはとても大切です。その上で，バグ票システムに求める機能や設定について設計していきます。本章では，バグ票システムの「設計」を解説します。




図 4.1 バグ票システムの稼働までのプロセス（設計）

さて，設計といっても範囲を広げるとこれもまたきりがないため，本ガイドでは**バグ票システムへの設定や運用のために最低限設計しておくべきことのみを取り上げます。**

では「設計」ではどのようなことを取り扱えば良いのでしょうか。本ガイドではバグ票システムの「設計」で取り扱う（決める）ものとして，以下を挙げています。

- バグ票のワークフロー（ライフサイクル）の設定仕様
- バグ票で扱う情報の設定仕様
- 利用者（ロール）と権限の設定仕様
- 必要な機能やその仕様
- 動作環境，必要となるインフラの仕様

本章ではここに挙げた５項目について簡単に解説します。

## 4.1 バグ票のワークフローの設計

本ガイドでは**バグ票は単なる一枚の帳票では無く，状態を持ったデータの集合**と捉えています。バグ票はバグの発生（発見）時にシステムに入力されることで作成（オープン）され，システムにワークフローとして登録したいくつかの状態を遷移しながら，最終的にバグ票の終結（クローズ）まで辿り着きます。この一連の流れをワークフローと呼びます。

図 4.2 にワークフローの例としてシンプルなものを示します。




図 4.2 シンプルなワークフロー

ネット上で参照できるOSS開発のバグ票システムの設定を見ると，このようなワークフローになっていることが多いようです。バグを発見したらその情報を新規登録，開発者がバグを修正して，問題なければ修正完了状態となります。このシンプルなワークフローであっても，現在報告したバグの対応状況がどのような状態にあるのかがリアルタイムで共有でき，また見える化することができます。

しかし，ソフトウェア開発の現場ではこのようにシンプルなワークフローであることはあまりなく，もう少し複雑なものが多いのではないでしょうか。図4.2を少し複雑にした例を示します（図4.3）。

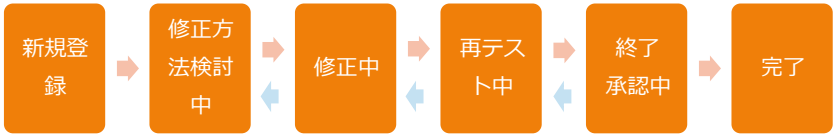



図4.3 少し複雑になったワークフロー

先ほどのワークフローへいくつかの状態を追加しています。これにより細かく対応状況を把握することが可能となります。また先ほどのワークフローでの状態遷移は一方向でしたが，このワークフローでは差し戻しや引き戻しの遷移が加えられています（水色の矢印）。

しかし，これもまた実際に開発の現場で使われているワークフローとしてはシンプルなほうで，実際は「終了承認中」から「修正方法検討中」へ遷移があったり，「修正方法検討中」から「終了承認中」に遷移することもあったりすることでしょう。3.2節や3.3節の内容に

よってはデバッグやメトリクス関連の状態が設けられる場合がありますし，審査承認を厳しくすると〇〇審査中や〇〇承認中といった状態が増えますし，却下の場合にどこに戻るかという遷移も複雑化します。３章で定義した情報によりこのワークフローを適切に設計する必要があります。

さらに状態を追加し，状態遷移を複雑にした例を図 4.4 に示します。

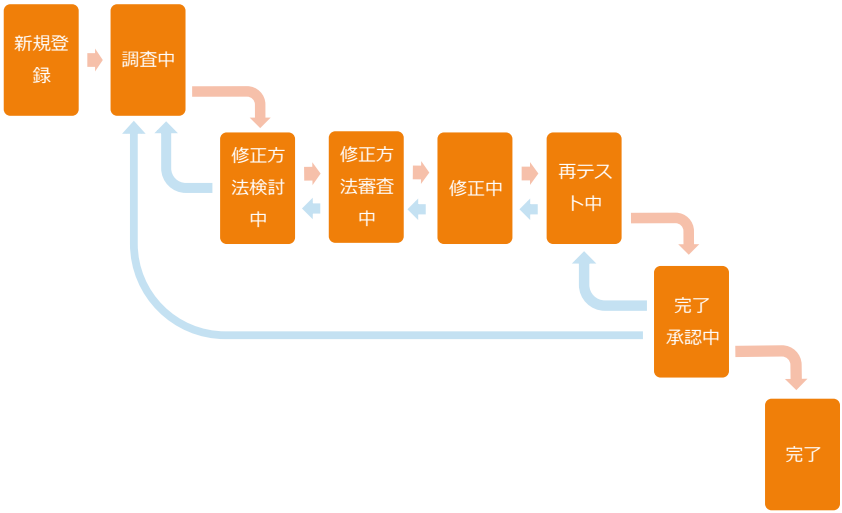



図 4.4　さらに複雑になったワークフロー

この例ではバグ情報が登録されたらそれを「調査中」にて調査し，その結果によって，修正するならば「修正方法検討中」に遷移させ，修正の必要が無い場合や単純な登録ミスの場合「完了承認中」に遷移

させます。また，「完了承認中」では「再テスト中」と「調査中」に差し戻します。これにより，承認プロセスが強化されたワークフローにすることができます。

ただし，あくまでもここでワークフローとして設計したいのはバグのライフサイクルであって，業務手順ではありません。バグ票システムに設定する（管理させる）ワークフローはバグ票に関するものに集中すべきで，決して業務を実装してはいけません。もし業務を実装したいという事であれば，バグ票システムという限定された使い方の設計は一度やめ，プロジェクト管理やタスク管理の範疇で検討するのがよいでしょう。

### 4.1.1 ワークフローの設計に利用できる技法

ワークフローの設計に利用できる技法として代表的には以下のような物があります。

- UML[10]のステートチャートおよびアクティビティ図
- PFD[11]（Process Flow Diagram）

[10] UML（Unified Modeling Language：統一モデリング言語）はOMG（Object Management Group）によって管理されているモデル記述言語である。主にオブジェクト指向開発で利用されている。
http://www.uml.org/

[11] PFD（Process Flow Diagram）は清水吉男氏によって提案されたプロセス表現のための言語。DFDをベースにプロセスと成果物の流れをわ

正直な話，バグ票システムのワークフローの設計程度くらいであれば，状態遷移図の類いあればなんでも対応は可能ですが，経験上これのワークフローは可読性や厳密性がある（つまり記法ルールが標準化されている）言語を使っておいた方が良いと考えています。その理由は「ワークフローのレビュー効率が向上する」「状態遷移テスト技法を適用するなど，テストしやすくなる」「保守の引き継ぎなど，後で誰もが読める仕様書としておくため」といったような理由があります。ワークフロー仕様は安易に作らず，きちんと技法を使って設計し，当然ながらレビューを実施する必要があるため，UML や PFD が書けない場合は合わせて技術導入するか，そのスキルを持っている人に設計を担当してもらうなどの策が必要となります。

## 4.2 バグ票項目の設計

バグ票はワークフローに従って状態が変遷していくデータの集合です。バグ票はその状態で情報を追加（入力）したり変更したりします。また，システムによっては入力した複数の情報からある情報を算出してくれたりもします。どの状態でどのような情報を追加するのかを全て洗い出し，システムに設定する上で型などを設計する必要があります。

---

かりやすく表現することが可能である。PFD の書き方については氏の Blog ページを参照されたい。
http://kohablog.cocolog-nifty.com/blog/pfd.html

バグ票で取り扱われる情報としては主に以下のようなものがあります。

- バグそのものの情報
- バグ票のライフサイクルに関する情報
- デバッグに関する情報
- ソフトウェアメトリクス
- 審査・承認や作業記録のための情報
- その他，参考情報など

先ほど示したワークフロー（図 4.2）を使ってシンプルな例を示します。

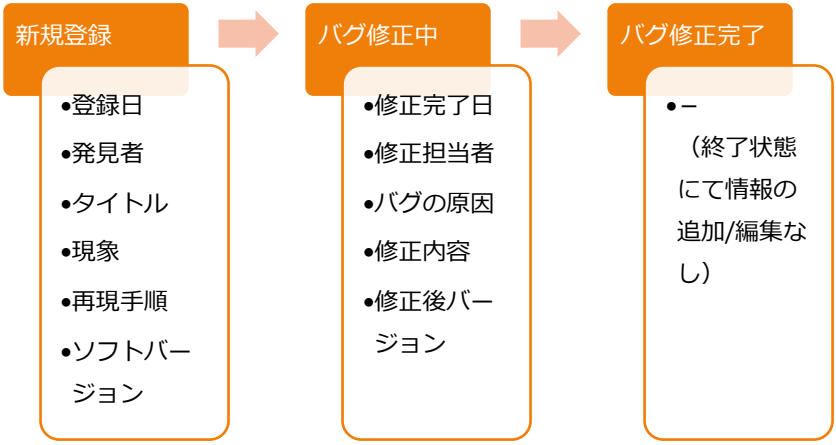



**図 4.4 シンプルなバグ票項目（状態ごとの入力項目）**

「新規登録」ではバグに関する情報を入力しています。「バグ修正中」ではデバッグに関する情報や担当者など作業に関する情報を入力しています。「バグ修正完了」はバグ票としては終了（クローズ）にありますから情報の追加や入力はありません。あくまでもこれはシンプルな例ですので，実際に業務で利用する場合は，情報はかなり多くなることが考えられます。ただし，なんでもかんでも情報を盛り込んでいくと，入力する手間もかかるようになります。ですので，情報を盛り込む際には本当にその情報が必要なのかという観点で考えることが必要です。よくある失敗として，バグ情報に関連が薄い業務情報であったり参考情報を盛り込みすぎて，入力フォームが長大になってしまったり，入力したはよいものの使われない情報が出てきたりとして，入力者の負荷を上げたり手間を増やしてしまうことになります。

この情報の設計については図4.4に示したような図を使ったり，ステートチャートを利用したり（Doに「以下の情報を入力」として整理するとわかりやすくなるでしょう），マトリクスを使うなどして整理するのが良いでしょう。

その際挙げた情報には，必須であるものとそうでないものをわかるように区別しておくと良いでしょう。この設計を設定するシステムによっては情報の入力時に必須項目として設定できるものがありますし，そういった機能を持たないシステムを利用する場合でも運用ルールとして決めておかねばならないケースがあります。

### 4.2.1　バグ票で扱う情報の設計で利用できる技法

バグ票で扱う情報の設計で利用できる技法として代表的には以下のような物があります。

- 情報の抽出/情報の構造化技法
- DFD[12]（Data Flow Diagram）

おそらくバグ票システムを導入しようとしているプロジェクトや現場はすでに何らかのバグ票管理の仕組みを持っていることがほとんどです。このため，バグ票で扱う項目を一から検討するのではなく，これまでに蓄積されている情報から必要な情報を抽出し構造化することがリーズナブルです。このあたりについて，NER[13]（Name Entity Recognition）など情報の抽出の技法や正規化構造化の技法を活用する

[12] DFD（Data Flow Diagram：データフロー図）は情報の流れを表現するための言語です。構造化分析設計においてよく利用されているモデル言語としても知られています。先ほど触れたプロセスセ家期のための言語であるPFDのベースとなった言語でもあります。
https://ja.wikipedia.org/wiki/%E3%83%87%E3%83%BC%E3%82%BF%E3%83%95%E3%83%AD%E3%83%BC%E5%9B%B3

[13] NER（Name Entity Recognition）は自然言語の形態素解析の一手法である。バグ票が大量にある場合，このような手法を利用することで情報抽出が手軽に実施できる。
https://ja.wikipedia.org/wiki/%E5%9B%BA%E6%9C%89%E8%A1%A8%E7%8F%BE%E6%8A%BD%E5%87%BA

のが良いでしょう。また，日頃設計書などで記載するDFDを活用するのも良いでしょう。PFDを使ってワークフロー設計した場合にはとても相性が良いです。

## 4.3 ロールおよび権限の設計

3.2項で誰が使うのかということに触れました。バグ票システムでは，機能利用可否や情報の入力可否，状態遷移の可否など，ロールや権限を設定してコントロールできることがほとんどです。利用者の種別をロールとして決め，それらのロールが，何ができて何ができないのかを定義する必要があります。仮に全員を何でもできるスーパーユーザーとしてしまうと，システムの設定を誰かが壊してしまうこともありますし，情報入力の際も気をつけないとそのロールが本来変更する必要がない情報を誤って変更してしまい，情報が壊れるということが起きたりします。また，場合によっては特定のロールにしか見えない情報が全員に見えてしまいセキュリティ面での問題が生じたり，ワークフローでの審査承認がマネージャやリーダ以外の誰でも操作が誰できてしまったりという作業管理上の問題が起きてしまいます。

小さなチームであれば運用ルールを設定することで乗り切れますが，人数が多くなってきたり，開発に複数のチームや会社が入ってきたりする場合には運用ルールだけでは統制が困難になります。従って，ロールおよび権限を適切に設定し，バグ票システムのデータや運用が壊れないようにする必要があります。

このロールおよび権限の設計は，ワークフローや情報，システムの機能を対象として実施しますが，その際はマトリクスを作成するなど

して，ロール間での不整合などが起きないように注意深く行なう必要があります。

### 4.3.1　ロールおよび権限の設計で利用できる技法

ロールおよび権限の設計で利用できる技法として代表的には以下のようなものがあります。

- ロール/権限マトリクス

ロールや権限の設計にはマトリクスを使うのが良いでしょう。ワークローとバグ票項目の仕様が出そろったのちに，それらに対してロールごとに権限を設定します。その際はワークフローやバグ票項目仕様に直接権限を記入するのではなく，マトリクスを使った方がわかりやすくなります。幸いワークフローは状態遷移図ベースの技法を利用しますので，それを状態遷移表に変換し，権限マトリクスにしてしまうのが実務的には楽です。

ワークフロー・バグ票項目と同様，ロールや権限についてもなんとなく，これは必要だろうあれも必要だろうそれも念のため入れておこうというような設計がされることが多いのですが，このような思いつき設計では，ロールが沢山増えたり，何でもできるスーパーユーザーだらけになったり，権限がコンフリクトを起こす状況になったりして収集がつかなくなってしまいます。ワークフロー仕様と同様，バグ票情報項目仕様も安易に作らず，きちんと技法を使って設計し，当然ながらレビューを実施する必要があります。もちろん技術を持たない場

合は合わせて技術導入するか，そのスキルを持っている人に設計を担当してもらうなどの策が必要となります。

## 4.4 必要な機能などの整理

その他，4.1節～4.3節といった基本であり必須の機能以外にバグ票システムで必要な機能などについても，システムに要求する機能要件・機能仕様としてまとめておく必要があります。例えば，バグ票へファイル添付機能といったものです。バグ票システムに対して求める機能としてよく挙げられるものには以下があります。

- バグ票へのファイル添付
- バグ票の入力項目の属性付加（入力必須や入力禁止）
- バグ票情報のインポート/エクスポート
- 更新のメール通知機能
- Wikiやフォーラムなどコミュニケーション機能
- プラグイン等を使ったシステムの拡張

これらの機能はツールやシステムの選定時に大きな影響を与えます。バグ票を管理するという本質的なところから外れ，使い勝手やバグ票システム運用の利便性などに属する物も含みますが，このあたりも検討しておかないと，実運用では使い物にならなかったということにもなりかねません。

## 4.5 動作環境や保守方法の設計（ネットワークやハードウェア等の稼働環境）

4.1節～4.4節はバグ票システムに対する設定や求める機能仕様という観点での設計ですが，それらを実際に設定するハアードウェア等の稼働環境も設計しておく必要があります。

「企画」で定義した内容によって利用人数や利用場所は変わりますので，ハードウェアやネットワークに求める物も変わります。お試しで使うのか24時間体制で使うのかでシステムへの信頼性も変わります。さらに，利用し続けるにあたり保守についても予めその方針や方法を決めておく必要があります。バグ票システムはとかくシステムの設定ばかりに意識がいきがちですが，バグ票に関するインフラストラクチャを構築するということですから，当然ながらシステム設計が必要であることは言うまでもありません。

インフラ設計については情報システム部門など社内の調整が必要な場合があります。可能であればそういった部門に予め関与してもらうように働きかけておくことが良いでしょう。なお，ハードウェアはいち度導入すると交換することが難しいことが多いので，可能な範囲で最もスペックが良い物を選択するのが良いでしょう。

## 4.6 まとめ

本章ではバグ票システムの「設計」を取り上げました。企画内容に基づいてそれを設計に落とし込むのは通常のソフトウェア開発のそれと変わりません。バグ票システムの設計とは本質的にはプロセス設計です。いきあたりばったりの設計ではなく，ステートチャートや PFD，情報抽出/構造化技法や DFD，マトリクスといった設計技法を駆使する必要があります。

なお，本章で解説した設計時にはすでにツール・システムが決まっている場合があります。この場合は，決まっているツール・システムに依存した設計をするのではなく，メタな設計情報としてまとめ，その後，そのツールのための実装仕様としてまとめることをおすすめします。メタな情報でまとめられていると，ツールやシステムを変更したとしても，その設計情報は流用が可能になります。この考えから，本ガイドではツールに依存した仕様については「実装仕様」として 5 章で解説する「実装」にてまとめることとしています。

# 5 バグ票システムの実装

４章ではバグ票システムの「設計」を取り上げました。バグ票システムに設定するワークフローやバグ票項目，ロールや権限等を実際にツールやシステムに設定する必要があります。５章では設計内容に基づいて実施するバグ票システムの「実装」を解説します。




図 5.1 バグ票システムの稼働までのプロセス（実装）

なお，「実装」については，ツールによってその実装方法や手順の具体的内容が異なるため，代表的なバグ票ツールやシステムを紹介することにとどめます。それぞれについては書籍などがすでに多く出版されているため，実際の実装手順（設定の操作手順等）はそちらを参照していただくようお願いいたします。また，それらツールの動作環境であるインフラストラクチャの構築に関しても，本ガイドの直接的なスコープから外れますので解説いたしません。それぞれの関連書籍などをご参照ください。

さて，「実装」ではどのようなことを取り扱えば良いのでしょうか。本ガイドではバグ票システムの「実装」で取り扱う（決める）ものとして，以下を挙げています。

- ツール・システムの選定
- 選定したツール・システムに対応した実装仕様の作成
- 実装仕様の実装（設定/反映）

本章ではここに挙げた３項目について簡単に解説します。

## 5.1 ツール・システムの選定

バグ票システムに設計した仕様を設定するためにはツール・システムを選定する必要があります。バグ票システムには，Microsoft Excel のようなスプレッドシートや Redmine や Trac といった Web システムがよく利用されているほか，独自システムを構築しているケースもあるようです。

バグ票システムは無数にありますが，選定においては以下のような観点で比較評価して決定するのが良いでしょう。

- 動作プラットフォームやインストールの容易さ
- ワークフローを実装/カスタマイズできるか
- バグ票項目を新たに実装/カスタマイズできるか

- ロール/権限を新たに実装/カスタマイズできるか
- ライセンスの形態
- 費用
- 必要な機能が備わっているか
- カスタマイズや保守は容易か
- 機能拡張性
- スケーラビリティ
- システムについての情報の充実度（書籍やヘルプ等）
- その他ユーザの要求事項への対応

これ以外についても様々な比較項目があるでしょう。例えばOSSであればコミュニティが活発かどうかも気にする必要があるでしょうし，日本語の情報とか書籍の充実度なども重要かもしれません。また，困ったときにベンダやサービスでのサポートが受けられるかとか，自社内にそのツールやシステムに明るい人がいるかなんてことも評価のポイントになるかもしれません。インフラ的なところでもスタンドアローン前提なのかネットワーク前提なのかもあります。海外の人達と一緒に使う場合は輸出管理の問題もあります。これら比較項目を洗い出し，場合によってはツール・システムの評価を実施して，最終的に選定する必要があります。なんとなく「流行っているから」という理由では後々こんなはずでは無かったということにもなりますので，注意深く検討することが必要です。

### 5.1.1　代表的なツール・システムの紹介

本項ではバグ票システムとして利用できる代表的なツール・システムのいくつかを簡単に紹介します。バグ票システムにおいて利用するツールやシステムの候補検討にご活用下さい。

#### 5.1.1.1　*Excel*

Microsoft Excel はスプレッドシートのソフトですが，日本国内においてはバグ票の記入フォームや一覧表の作成ツールとして多くの現場で利用されています。業務用の Windows PC であればたいていの場合インストールされており，また，多くの人が基本的な使い方については習得しているためトレーニングをあまり必要としません。ただし，ワークフローの機能を持たず，変更履歴の機能も他のツールに比較すると弱いため，バグ票システムとしては向いていません。

マクロを使うことである程度の処理などは可能となりますが，コードレベルでの作りこみが必要となります。また，ツールのバージョンアップによって機能が動かなくなってしまうという危険性も持っています。

バグ票のレイアウトを作るには Excel は便利ではありますが，結果としてメンテナンスのコストが大きくなってしまったり，同時に複数のメンバで情報が更新できなかったり，チームでの利用には不向きであるため，その危険性を理解のうえ利用することが必要です。

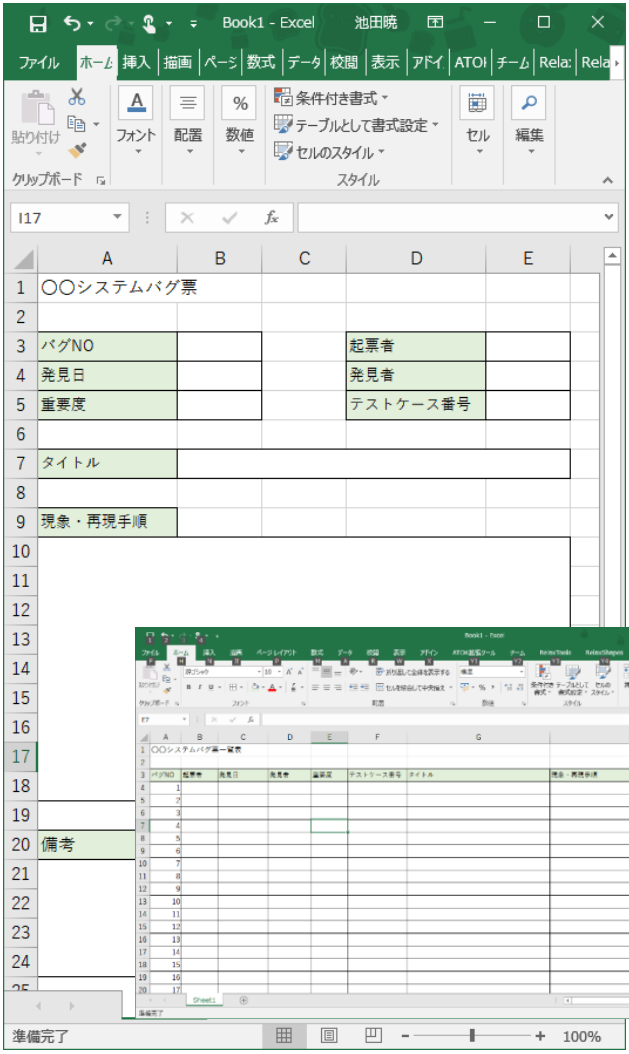



図 5.2 Excel の画面例

#### 5.1.1.2 *Redmine*

Redmine は，OSS のプロジェクトマネジメントシステムです。

タスクを管理することが主なシステムですが，バグに関する作業をタスクと見なし，バグ票システムとして利用することが可能です。

バグ票はチケットという単位で管理します。チケット項目を様々な型で設定できるほか，ワークフローやロール・権限の設定など，バグ票システムに必要な一通りの機能を持っているほか，ガントチャートやカレンダー，文書やファイル，Wiki 等の機能も持っています。

Ruby on Rails で動作しますが，Bitnami Redmine Stack などのインストーラを利用すると簡単にインストールができることも魅力です。

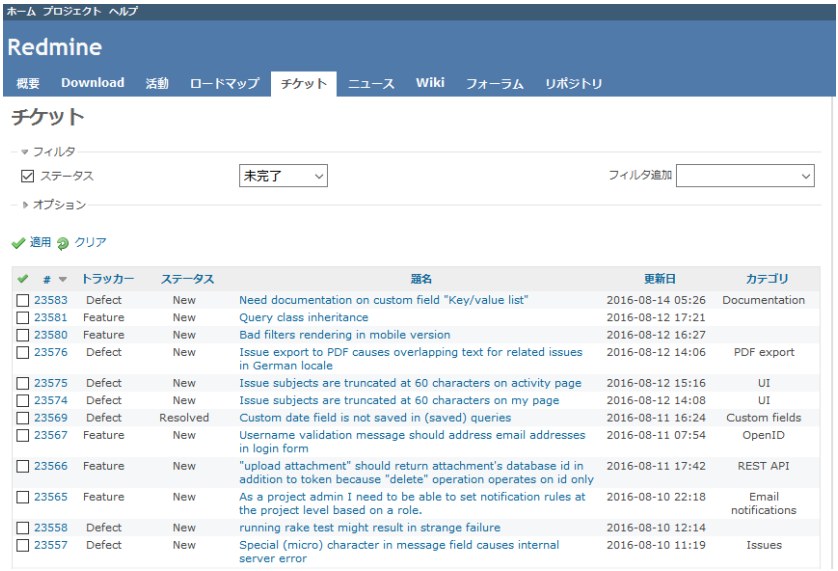



図 5.3 Redmine の画面例（Redmine 公式サイトより）

#### 5.1.1.3 *Trac*

Trac は，OSS のバグ・トラッキング・システムです。

バグ・トラッキング・システムなので，バグ票システムとして必要な機能が一通り揃っており，国内では Redmine と並んでよく利用されているツールです。Trac ではチケットの状態変更時にフックスクリプトを入れ込めるなどの柔軟性がありますが，一方でチケットレポートのクエリ等は SQL を打つ必要があるため，慣れるまでは検索に時間がかかるかもしれません。またワークフローなどを環境設定ファイルにテキスト記述する必要があるため，プログラムが得意で無い人にとっては少々のハードルがあります。

また，最近は更新頻度も落ちてきているため，今後中長期間システムを稼働させる場合には，未来のバージョンについて注意しておく必要があります。

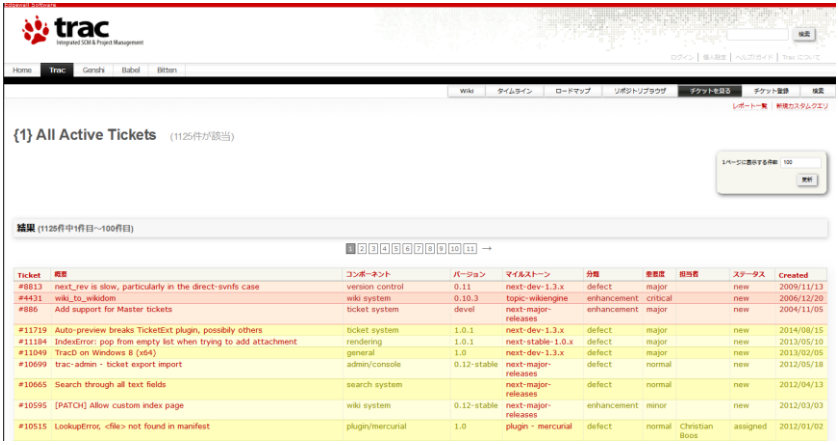



図 5.4 Trac の画面例（Trac 公式サイトより）

#### 5.1.1.4 *Mantis*

Mantisは，OSSのバグ・トラッキング・システムです。

チケットの検索を様々なフィルターを通して誰でも行えるのが他のシステムと異なっています。特定の週に報告されたチケットで重要度の高いもの，特定のOSやバージョンのある機能に限定した絞込みなどチェックボックスのオン/オフや日付の入力だけで行える為，デフォルト設定だけで情報収集が容易です。

またBTSだけあって，デフォルトのチケットはバグ票のイメージに近いです。項目として，報告する現象の再現率，優先度，重要度，概要や再現方法，追加手順などがデフォルト設定されており，バグ票フォーマットを設計する際の参考にもなるでしょう。

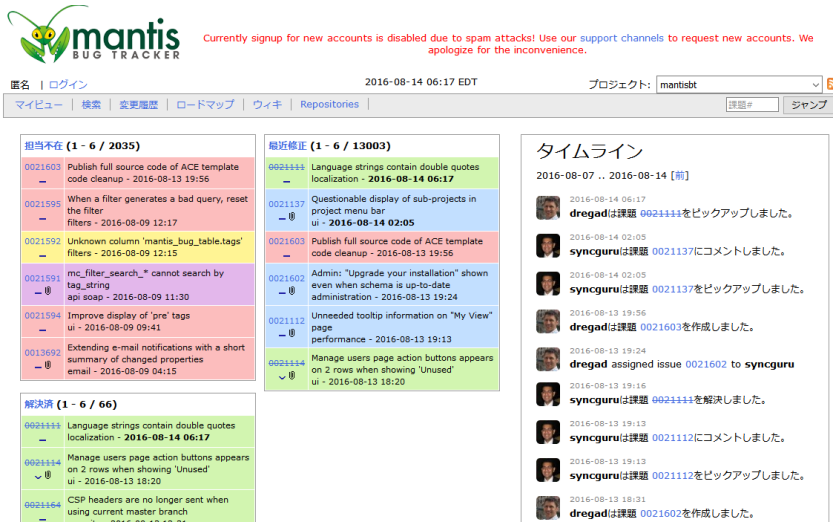



図 5.5 Mantisの画面例（Mantis公式サイトより）

#### 5.1.1.5 *その他，ツールベンダによる商用ツール*

その他，ツールベンダによる商用ツールやシステムがあります。

最近では JIRA[14]（アトラシアン社）が注目を集めているようです。また TFS[15]（マイクロソフト社）等のテスト管理ツール・システムがバグ票システムを内包している場合もあります。その場合，要件やコード・テストケースなどとのトレーサビリティを確保できたりします。Excel や OSS ツールは手軽に始められますが，ベンダのツールはそれらに比較してコストが大きいもののユーザサポートやベンダトレーニングが充実しているため，保守まで考えて導入検討するのも良いでしょう。

[14] JIRA　https://ja.atlassian.com/software/jira

[15] TFS（Team Foundation Server）　https://www.visualstudio.com/tfs/

## 5.2 選定したツール・システムに対応した実装仕様の作成

ツールやシステムの選定が終了したのち，それらに対して４章で作成した設定仕様を具体的にどのように実装するか仕様化します。

採用するシステムによってワークフローやバグ票項目，ロールや権限の設定のやり方は変わります。例えばTracであればワークフローはiniファイルに内容を記述しますが，Redmineだと Webブラウザでの管理画面から状態遷移マトリクスに内容を設定していきます。また，バグ票項目もシステムによって準備されている「型」が異なります。このため，そのシステムに具体的にどのように実装するかを定義しておく必要があります。また，当然ながらシステム全体の設定も差違があります。これに関してはツール・システムが決まらないと内容が定義できない部分も多いため，実装仕様としてきちんと定義しておく必要があります。

この作業を行なわない場合，いきあたりばったりの実装となってしまい，設定ミスが多発，リリース後の保守も困難になります。

## 5.3 実装仕様の実装（設定/反映）

実装仕様書が完成したらツール・システムの手順に従って設定していきます。この実装手順については各ツール・システムのヘルプや関連書籍をご参照下さい。

## 5.4 まとめ

本章ではバグ票システムの「実装」を取り上げました。

企画内容に基づいて作成された設計仕様に基づいて，ツール・システムを選定する必要があります。導入するバグ票システムを長期間にわたって利用するなどの場合，選定におけるベンチマークや評価にはそれなりのパワーをかけることをおすすめします。評価においては実際にシステムをインストールし動作させてみることも考えた方が良いでしょう。一度動き始めたシステムは利用者が増えれば増えるほど，標準化が進めば進むほど，途中で変更することが困難になります。ですから，評価ではこれで最低１年以上は稼働し続ける事が可能であることの確信を得られるくらいまでの深さで取り組むことが必要です。このあたりはシステム導入に関する書籍を参考にするなどして取り組むと良いでしょう。

ツール・システムが決まったら，それに対応した具体的な実装仕様をまとめておくことも重要で避けられない作業です。事前に実装仕様（書）をまとめておくことで，設定作業での抜け漏れや誤り防止の効果を期待できます。また，システム稼働後の設定変更や，他部門などに保守を引き継ぐ場合など，その時点での最新の実装仕様が残っていないと設定ミスが発生したり，システムになれていない人には設定変更自体ができなかったりします。つまり，保守が困難になってしまい，結局稼働中のシステムからリバースで実装仕様書を作成するはめになります。また，その際のコストは事前に作成している場合よりも高くなります。

実装仕様の実装はヘルプや書籍などを参考に手順に従って実施して下さい。

以上，仕様の実装といっても，業務で使う以上，いきあたりばったりにならないようにする必要があります。はやく設定して動かしたくなる気持ちも理解できますが，そこはぐっと抑え確実な実装ができるように一手間かけることが重要です。

# 6 バグ票システムのテスト

５章ではバグ票システムの実装を取り上げました。これでバグ票システムは動く状況になりましたが，プロジェクトにリリースする前に仕様が正しく設定されているか確認する必要があります。本章では見過ごされがちであるバグ票システムの「テスト」を解説します。




図 6.1 バグ票システムの稼働までのプロセス（テスト）

ただしバグ票システムのテストといっても本ガイドにおいては**選定したツール・システムそのものをテストするわけではなく，定義した実装仕様が正しく設定されているか，実装仕様が正しく設定されたシステムが実運用に耐えるかどうかを主眼に置いたテストを実施**します（ツールやシステムに依存する非機能などは対象外です）。具体的には以下のような観点でテストします。

- 定義したワークフローが正しく設定されているか
- バグ票項目および型が正しく設定されているか

- 利用者（ロール）と権限が正しく設定されているか
- その他の設定が反映されているか
- 実際にプロジェクトで稼働できるシステムとなっているか

上述したように，あくまで設定仕様が正しく設定されているかどうかをテストすることに注意して下さい。

具体的には「設定値が正しく設定されているか」「仕様通りにワークフローが動作するか」「実運用を想定したテスト」の３つが最低限のテストとなります。本章ではこれら３つについて簡単に解説します。

## 6.1 設定値が正しく設定されているかを確認する

実装仕様で定義した具体的な設定値がシステムに正しく設定されているかを確認します。

例えば，バグ票項目がその項目名・型で設定されているかどうかなどを１つずつ確認します。決めた設定値が反映されているかどうかをチェックするため，特段テスト技法などは使う必要は無く，基本的には実装仕様を塗りつぶしながら確認していくことで対応が可能ですが，可能であれば別途テストケースリストを作成して確認しておくのも良いでしょう。その結果をリリースするにあたってのエビデンスとすることが可能になります。

## 6.2 ワークフローが正しく動作するかを確認する

ワークフロー仕様がバグ票システムに正しく設定されており，ワークフロー通りにバグ票オープンからクローズまで流れるかをテストします。

ワークフロー仕様に対して状態遷移テスト技法やシナリオテスト技法を適用するなどして，仕様として定義した状態や遷移を網羅する必要があります。このとき，ロールによって遷移や情報入力，遷移のトリガやガード条件，入力できるバグ票項目が異なる場合があるため，それを考慮してテストケースを作成する必要があります。その他，権限設定によってバグ票項目の表示非表示が変わるシステムもあるため，ロールや権限に加え，システムの特性も考慮しておく必要があります。

設定値チェックに比較すると一筋縄ではいきませんが，バグ票システムを使ったバグ管理はこのワークフローが肝であるため，可能な限り厳密にテストを実施手おくことが重要です。プロジェクトにリリース後でのワークフローの変更は即ちバグ管理プロセスの変更であるため，影響がとても大きいことを認識しておく必要があります。

なお，バグ票システムに Web システムベースの物を採用する場合は，のちのちの保守も考えて Selenium[16]などのツールを使ってテストを自動化しておくことを推奨します。

[16] Selenium　http://www.seleniumhq.org/

## 6.3 実運用を想定したテストを行なう

実際にユーザを交えて捜査してみるなど，実運用を想定したテストを行います。

前節で解説したワークフローのテストはあくまでワークフローとして設定が反映されているかを主眼に行なうテストでした。ただし，実際に設定をシステムに組み込んでみると，事前にレビューをしていたにも関わらず，実運用としては使いにくかったり，そもそもワークフローがあわなかったり，ネットワークやハードウェアの性能に動作が影響されたりということが起こったりします。このため，バグ票システムが稼働する環境もしくは限りなくそれに近い環境にて実運用を想定したテストを実施する必要があります。所謂システムテストに相当するようなテストが必要となります。各種テスト技法やテストツールを利用してテストする必要があります。

## 6.4 その他のテスト

ツールやシステムそのものに対してテストは行なわないとしましたが，機能の重要度に応じて，ツール・システムの機能に対するテストを追加しておくことは良いことでしょう。環境によってはファイルのエンコードや設定のコンフリクトやツール・システムのバグによって機能が動作しないこともあります。特にOSSのバージョン違いは気を付ける必要があります。評価時のバージョンとテスト時のバージョンが異なっている場合，評価時とは機能の振る舞いが変わっていたり，バグが発生していたりということが起こりえます。

## 6.5 まとめ

本章ではバグ票システムの「テスト」を取り上げました。設定の反映が終わると動いているように見えますが，なにせ人間が設定するものですからほぼ確実にバグを埋め込みます。バグ票システムはバグ管理プロセスをツールのワークフローに置き換えた物とも言えるため，バグがプロジェクトに与える影響は大きいものとなりがちです。このため，テスト技法を使うなどしてきちんとテストをする必要があります。

# 7 稼働準備

第６章ではバグ票システムの「テスト」を取り上げました。第７章では「稼働準備」について解説します。




図 7.1 バグ票システムの稼働までのプロセス（稼働準備）

バグ票システムだけに限りませんが，システムの円滑な導入には事前の稼働準備が重要です。稼働のためには，システムに関するドキュメントのほか説明会など開催する必要があります。

バグ票システムの「稼働準備」として代表的なものには以下のようなものがあります。

- システムに関するドキュメントの整備
- 説明会の実施
- 利用者のトレーニング

本章ではここに挙げた３項目について簡単に解説します。

## 7.1 システムに関するドキュメントの整備

バグ票システムを稼働するにあたっては，主に以下の２種類のドキュメントを整備しておく必要があります。

- システム管理者用マニュアル・ドキュメント
- ユーザ用マニュアル・ドキュメント

管理者向けには，システムを円滑に稼働・メンテナンスするために，システムの設定の変更に関する情報をまとめたドキュメントやシステム利用者の管理に関する情報をまとめたドキュメント，バックアップなど定期メンテナンスに関する情報をまとめたドキュメントなどを整備する必要があります。

ユーザ向けには，システムを円滑に利用するために，システムの全体像や機能を解説するドキュメントや操作マニュアル，FAQ などを整備する必要があります。

## 7.2 説明会の実施

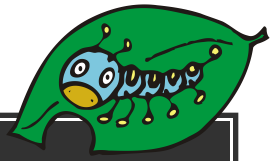
ドキュメントが整備できたら，システム利用者（管理者・ユーザ）を集めて説明会を実施することを推奨します。全員にもれなくシステムそのものや導入の目的，今度の稼働のスケジュールなどを説明するほか，参加者から質問を受けます。システム稼働前に質問を受けておくことで参加者の疑問を払拭するほか，重要なことについては関連ドキュメントにフィードバックします。バグ票システムやバグ管理に利用者全員が詳しいわけではありません。また，それぞれに個別に説明や質問を受け付けるのは大変です。説明会を行なうことで一度に全員の理解してもらうことが可能です。

## 7.3 利用者のトレーニング

これまでバグ管理システムを使ったことがないシステム利用者に対しては，事前にトレーニングが効果的です。また，機能豊富なシステムやワークフローが複雑なシステムであった場合も事前にトレーニングを実施しておくと良いでしょう。またトレーニングを実施しておくことで，利用者の本稼働に向けての不安感を払拭できるほか，稼働初期の問い合わせを減らすこともできます。当然ながらシステム管理者向けトレーニングとユーザ向けトレーニングそれぞれについて実施を検討すると良いでしょう。

## 7.4 ・まとめ

バグ票システムの円滑な稼働のためには，利用者のための情報や説明会，トレーニングの実施が効果的です。特に，大規模導入の場合，この稼働準備にしっかりと取り組めるかどうかで，稼働直後の混乱具合が大きく変わります。また，理解が不十分である状況では折角作ったシステムを効果的に利用してもらうことができません。



# 第ＩＩＩ部
# バグ票システムを
# よりよく利用するために

第ＩＩＩ部ではバグ票システムを有効に使うための参考情報を紹介します。

# 8 バグ票に関してのワーストプラクティス

第 II 部までは，バグ票システムの検討・導入などについて説明してきました。第 III 部では，バグ票システム稼働後に起こりそうな問題や有効に利用するために考えることを紹介します。

本章では，バグ票システム運用時に，バグ票が有効に使われていない例(ワーストプラクティス)のなかからいくつかを紹介します。

ソフトウェア開発現場において，バグ票システムを導入したにもかかわらず，バグ票が有効に利用されていないあるいは有効に利用できていない，場合によっては，利用することが目的になってしまい，バグ管理が形骸化してしまうことがあります。

「バグ票ワーストプラクティス検討プロジェクト」は，バグ票が有効に利用されていない事例を JaSST[17](ソフトウェアテストシンポジウム)や SQiP シンポジウム[18](ソフトウェア品質シンポジウム)，

[17] JaSST（Japan Symposium on Software Testing：ソフトウェアテストシンポジウム）は日本におけるソフトウェアテスト技術に関する最大のシンポジウムです。バグ票に関連する発表も過去実施されています。
http://www.jasst.jp/

[18] SQiP シンポジウム（ソフトウェア品質シンポジウム）は日本におけるソフトウェア品質技術に関する最大のシンポジウムです。こちらも過去バグ票に関する発表が実施されています。
https://www.juse.jp/sqip/symposium/

WACATE[19](ソフトウェアテストワークショップ)など，ソフトウェアテストやソフトウェア品質に興味・関心がある方々を中心に議論や調査などを行なってきました。議論や調査した事例からバグ票がうまく使われないパターン(ワーストプラクティス・アンチパターン)のなかからいくつか紹介します。

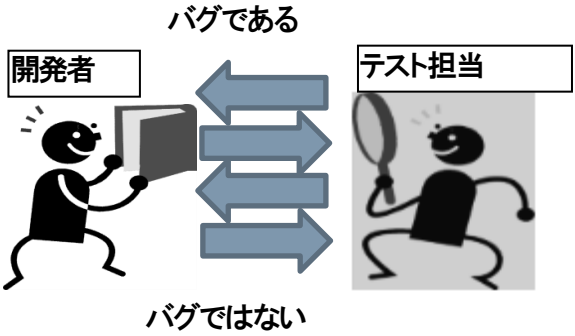
バグ票ワーストプラクティス検討プロジェクトやワーストプラクティスについての詳細は次のURLからご参照下さい。

★バグ票ワーストプラクティス検討プロジェクト
https://sites.google.com/site/swworstpracticesite/

[19] WACATE（Workshop for Accelerating CApable Testing Engineers：ソフトウェアテストワークショップ）は日本におけるソフトウェアテストに関する最大の勉強会合宿です。過去にバグ票に関する発表や演習が実施されています。
http://wacate.jp/

## 8.1 バグピンポン

ある現象やふるまいに対して起票したバグ票が，テスト担当者と開発者の間で，「バグである」「バグではない(または，仕様である等)」というやり取りが収束しない状態を示します。




この現象は，テスト担当者と開発者の間で心理的あるいは物理的離れていることでコミュニケーションが円滑でない，過剰な責任分担意識から自分の立場を守る(作業を増やさないことを含む)ことや，テスト担当者の「品質番人意識」が要因と考えられます。

この現象を防ぐには，テスト担当者と開発担当者の間に調整役となる担当者を置くことや，テスト担当者と開発担当者などでテスト目的の共有化を図ることで，「何を確認するためのテストなのか?」「プロダクトで必要な特性は過不足なくすべてテストされているか?」を合意しながらプロセスを進めることが考えられます。テスト担当者と開発担当者の間で同等のスキルを持ち，お互いの立場や責任を理解したうえで，バグ票の内容を共有することが求められます。関係者が，お互いが経緯と尊敬を持ちながら仕事ができる環境であれば発生しにくいでしょう。

## 8.2 バベルの塔

バグ票に記載する用語や記載ルールが不統一であることから，バグの問題解決管理(たとえば，品質担当者やマネージャなどによる品質分析など)に支障をきたす現象です。例としては，ある機能名の略称が同じ機能であるのに複数の呼び名がある，再現性記載が自由記述になっており人によって再現頻度の表現が異なる(再現率の表現を「100%」と記載されたり，「5回中5回」と記載されたりする)，ということがあげられます。この現象により，バグ票の担当者割り当てや，優先して対応するべきバグ票が抽出しにくくなり，バグ票の運用やバグ票から判断すべき対応に支障が発生し，結果として，優先して対応が必要なバグ等が抽出されにくくなることでプロダクトのリリースに影響が出る，という恐れがあります。

この現象は，プロジェクト内でバグ票に関する記載の自由度必要以上に大きいこと，あるいは記載のための教育がないために発生します。また，プロジェクトのリーダやマネージャが「バグ票くらいなにも指導せずとも書けるだろう」と楽観的に考えてしまい，メンバのドキュメント作成能力(ここでは，バグ票の記載内容や粒度の理解)を過信しすぎた場合にも発生しやすいと考えています。さらに，この現象が，特に表れるのは，プロジェクトの進捗が遅れ気味になり，プロジェクト要員を追加投入したときです。特にパートナー企業(協力会社)等，異なる組織からの要員が，プロジェクトにおいて適したバグ票を起票できないこともあります。

対策として2つの点が考えられます。マネジメント的な観点とシステム的な観点です。

マネジメント観点では，プロジェクト開始時，あるいは要因追加時に必ずバグ票の書き方の教育を行なうことです。どのように記載すればよいか，プロジェクトで起票された手本となるバグ票を例に説明するとよいでしょう。多くの方と議論してきた中感じたことの１つに，「バグ票の書き方の教育がない」ことがあげられます。自分とは異なるメンバがどのようなバグ票を書いているか知らないことが思ったより多いように思います。「自分たちにとって良いバグ票とはなにか?」という基準を作ることが重要です。そうしてさらに，バグ票システムのワークフローによりリーダやマネージャ等がバグ票のチェック行ない，問題があるは場合など都度フィードバックすることで効果が見込めます。

システム的な観点では，テキストボックスによるフリーフォーマットによる項目はなくし，選択式にし，起票者・入力者の属人的な回答を減らすことで条件による抽出をしやすくすることなどが考えられます。機能名など名称やIDが一意に決まる物は選択式にすることで効果が見込めます。



## 8.3 「このぐらいわかってくれよ」症候群

この現象は，プロジェクトの進捗が遅れこれ以上の遅延が許されないような強いプレッシャーがかかることや，リソースが足りないことで，「バグ票を丁寧に記載したり，確認を行なったりしているような余裕がない」などと思ってしまい，バグ票の記載がおざなりになってしまうことから，読み手に負担をかけてしまう現象です。具体的には，空欄が増えたり，定義のないような用語や表現が断りなく使われたり，再現条件や手順の一部が報告者の都合や思い込みにより省略されてしまうことです。

この現象は，リソース(特に人や工数)が足りなくなることで発生します。特に，上述したように，時間的心理的余裕がなくなることで，バグ票の読み手を意識できず，いわば周りが見えなくなることにより起票してしまうことに起因していると考えます。

この状況では，マネージャがプロジェクトにおけるバグ票起票の重要度をきめ，必要であればリソースの確保に努めるとよいでしょう。リソースの確保が難しい場合，バグ票起票後などにチェックする担当者を置くことが考えられます。また，起票するバグ票の対象を制限するということが考えられます。バグ票で行ないたい業務をよく考え，後続プロセスやプロセス改善に利用するに値する内容のみを起票対象にする，などです。例としては，誤字脱字などは起票対象外とし，深刻な不具合や，影響の大きいバグの場合にバグ票を記載する，ということが考えられます。そのうえで，バグ票の内容のチェックを行ない，不足な点，不十分な点をフィードバックし，適切なバグ票になるように支援することが効果的です。

## 8.4 書き手理解不足

バグ票の項目に関する記載が不明確不可解な部分があり，または空欄があることにより，バグ票の読み手が，バグ票に対してどう処理してよいか判断に迷う現象です。バグ票の状態が変更できないことから，調査が停滞したまま放置されたり，調査が不充分なまま却下されたりといった，不適切な状態変更への引き金になります。また，確認のために余計なコストが増える(負担が増える)ことになります。

この現象は，プロジェクトメンバの中に，テストの経験が少ないメンバがいることや，対象プロダクトの経験が少ないために，読み手がどのような情報がほしいか把握していないままバグ票を起票してしまうことで発生します。また，プロジェクトのリーダやマネージャが，バグ票は教育がなくても作成できるという思い込みがあることでも起こりやすくなります。

バグ票により，バグの情報は過不足なく伝えられることが求められます。このために，チーム内で「良いバグ票」の認識を共有することが重要となります。その1つとしてケーススタディバグ票による教育が必要になります。自分たちの組織やプロジェクトのバグ票から，バグの内容とバグ票のセットで事例として提示し，良いバグレポート，悪いバグレポートの認識を合わせることが重要となります。

また，起票されたバグ票の内容をチェックし，できるだけはやくフィードバックをすることが重要です。経験が浅い担当者や，プロジェクトに参加したばかりの担当者に，あなたのレポートで足りない情報や修正が必要な情報はこういうことです，という説明が都度行われることが必要であると考えます。



## 8.5 まとめ

ここでは，4つのアンチパターンについて説明しました。いずれにも共通していると考えられることは，「バグ票を自分が見える範囲でしかとらえていない」ということです。「ソフトウェア開発工程のなかでバグ票がどのように使われ，だれに対してなにを伝えたい内容なのか？」「自分のバグ報告内容あるいは回答内容を受けた人が，報告内容からどのような作業をしなければならないか？」という考慮が足りないことです。

「バグ票」は名前の通り，バグ「レポート」です。レポートということは，必ず相手がいます。自分の主張があり，主張に伴って相手にしてほしいことがあるはずです。また，バグ票の特徴の1つに読み手

と書き手が頻繁に代わることがあげられます。わかりにくいバグ票を書くということ，まわりまわって自分の作業を増やすことになります。

品質を考えるうえで重要であることの１つに「次工程はお客様」というフレーズがあります。バグ票を受け取る人をお客様と考え，自分の主張を明瞭で分かり易く表現し対応を依頼するような気持ちでバグ票を書くことを続けることでで，今まで以上に，開発組織にとって生きたバグ票になります。

バグ票の起票者は，プロジェクトやプロダクトに有益な適切なバグ票を書き続けることで，バグ票の管理は勿論，プロジェクトそのものの改善に貢献することができます。

# 9 バグ票を活用した活動

バグ票は，関係者，主にテスト担当者と開発者の間のバグに関する情報共有と，報告内容に対する対応要否などを決定する根拠として利用されます。しかし，それだけではバグ票の効果は十分利用できていません。バグ票は，現在進行中のプロジェクト/プロダクトに対するバグ情報を共有するだけではなく，自分たちのこれから少し未来のソフトウェア開発をよくするためにも使うことができるからです。

本章では，現在進行中のソフトウェア開発対象に対するバグ票の使い方と，これからのソフトウェア開発を今より少しよく行うためのバグ票使い方の例を紹介します。

## 9.1 バグ分析，原因分析

バグ分析や原因分析を実施する目的として，次のようなことが考えられます。

- 現在の作業状況や品質状況の確認
  - どのようなバグがでているか？ 追加でテストをする必要があるか否かの判定などを確認します
- 後工程や最終品質の予測
  - このままプロジェクトが進行した場合，目標とする品質基準(バグ密度やテスト密度等)，納期やコストなどの条件を満たせるか？などを確認します
- 次回の開発作業で，バグを予防するための対策検討

- 次回の改修作業では，バグを予防するために，仕様検討・設計時や設計レビューなどでどのような仕組みやプロセスを直すことで予防できるか？などを確認する

あるプロジェクト，あるいはあるプロダクトといった切り口でバグ票を分析することで，自分たちのソフトウェア開発でどのようなバグがでているかわかります。バグ票からわかるバグ分析は，バグ票にどのような項目を入れるかで変わってきますが，以下のような点に着目してみるとよいでしょう。

- 欠陥の種類
  どのような欠陥が検出されているかという情報。たとえば，初期化忘れ，インタフェース不正，タイミング誤りなど
- バグの深刻度
  例えば，深刻，中程度，軽微，といったプロダクトに対する影響度合い。深刻なものが多い場合，なぜ見逃しているか？やまだ検出できていない深刻なバグがあることを確認する必要があります
- 欠陥混入工程と，欠陥検出工程
  欠陥が混入した工程と欠陥検出した工程の件数などを調べます。また。混入した工程と検出した工程が同じであることが望ましいが，いくつもの工程を経て検出されているバグ，例えば，要求に関する欠陥がシステムテスト

で検出される，という場合は，なぜ，混入行程から時間がかかってしまっているか確認する必要があります。

- 欠陥検出箇所(プログラム名やモジュール名，機能名など)
  ソフトウェア開発では，全体に均等な分布でバグが検出されることは多くなく，むしろ特定の部分やモジュールなどに偏って多く検出されることが多い(JSTQB テストの 7 原則「原則 4 欠陥の偏在」)。特定のモジュールやコードに修正が多い場合，次回以降の開発にも大きな影響を与える恐れがあるため，なぜ修正が多いか確認する必要があります。必要であれば，レビューやテストを再度行ない，品質やその他に問題がないこと，問題がある場合はその原因などを確認する必要があります。
- バグ票が起票(オープン)されてから，クローズされるまでの時間が大きいバグ票

バグ票だけに存在する項目だけでなく，プロダクトに関するメトリクス（たとえば，開発対象規模，コードの複雑度）やプロジェクトに関するメトリクス(メンバの人数やプロジェクトに参画してからの経験年数など)もあわせ分析することで，より効果的な分析が可能となります。こうした情報をどこかに持っておく必要があります。



## 9.2 バグの予測

前節で，自分たちの開発する対象にとって作りこみやすいバグが分かります。「どこにどこのような種類のバグを作りこみやすいか?」「どの部分にバグが多いか?」などが分かります。これはプロダクトやプロジェクトメンバが大きく変わらない限り，次回の作業でも同様の結果になることが予想されます。

そこで，バグ票からの分析結果を，テストの前の工程，たとえば，仕様検討時や設計レビューで適用することが必要です。この時分析やテスト担当者が，設計開発者に伝え，後工程（テスト工程）で検出されないようにすることが大切になります。

ライフサイクルが長いプロダクトの場合，過去の多くのバグ情報から「予測モデル」を作成することができます。「予測モデル」を作ることで，バグの多そうな箇所やバグ数などが予測することが可能となります。予測モデルについては，多くの論文や書籍を参照してください。書籍では，例えば「データ指向のソフトウェア品質マネジメント―メトリクス分析による「事実にもとづく管理」の実践」(日科技連出版社)などがあります



## 9.3 バグ票システム自体の改善

「バグ票の項目にどれくらい変更要望が入るか?」ということも，バグ票がどれだけ関係者に利用されているかのバロメータになります。

上述した「バグの分析」，「バグの予測」を行なえるようになると，現在利用しているバグ票システムの項目だけでなく，目を追加たな項加したり，あるいは設定してみたが使用されていない項目(記入のコストが高いわりには効果がない項目など)を削除したりすることが発生します。また，前述したようにフリー入力を選択式にするなど変更もあり，そうするほうがバグの内容の集計や分析を行なうこと容易になるかもしれません。自分たちの目的に合わせて使いやすい(バグの情報がしやすく，問題が発見しやすい(バグ分析やプロセスの問題が見えるような))改善を行なうことが重要です。もしバグ票の項目を決めたのち，長期間経っても項目に変更が入らない場合，関係者に対してヒアリングを行ない，「利用者それぞれの立場でバグ票の内容から自分の作業が過不足なく行えているか?」「情報共有ができているか?」等を確認する必要があります。

## 9.4 まとめ

本章では，バグ票をさらに有効に利用するための使い方の例を紹介しました。バグの記録というと，すでに作成したコードに対する対応，過去に埋め込んでしまった不具合対応という「後始末」的なイメージをもつ方が多いようです。しかし，本章で記載したように，自分たちのこれから先のソフトウェア開発において，「すでに報告されているバグ票をどのように利用するか?」という視点，つまり，「自分たち

はどのようなバグを作りこみやすいのか？」「これらを(テストより前で)予防するためにはどのようにすればよいか?」を考えることが大切です。例えば，「設計ドキュメントに記載するようにする」「設計レビューの観点をレビューワ全員で確認する」「テストケースレビューで同様の観点が盛り込まれていることを確認する」など，という視点で「バグ票の記録から欠陥を予防する」ためにバグ票を使えるようにすることです。

最後になりましたが，本章に記載した内容の前提として重要なことがあります。それは，「１件１件のバグ票が適切な内容になっているか？」ということです。開発プロジェクトや組織にもよりますが，すべてのバグ票が分析に利用可能であるとは限りません。程度の差はあれ，分析に利用できない（対象外となってしまう）バグ票が発生してしまいます。分析できないバグ票が多くなると，精度の高い適切な分析や対策を検討することが難しくなります。つまり，バグの情報は何もしなくても集まるわけではなく，バグ票記載時から，適切な内容であるか確認する必要があります。プロジェクトが収束した時点で大量のバグ票をあとから修正することは難しいです。バグ票起票時や，状態とクローズとする前に，１件１件のバグ票の内容が適切であるか確認することが，バグ票を有効利用するうえで重要になります。

以上のようにバグ票は開発プロジェクトや組織を改善することにも有効利用することができます。単なるバグの管理から脱却して，未来の自分たちが良い製品開発をするために有効利用することを検討しましょう。



# 第Ⅰ Ⅴ部
# おわりに・参考文献

第Ⅰ Ⅴ部では参考文献を紹介します。

# 10 おわりに

本ガイドは全４部で構成しました。

第Ｉ部ではバグ票システムの導入プロセスを解説する前に，バグやバグ票，バグ票システムについて簡単に解説しました。

第ＩＩ部ではバグ票を導入するために必要な，企画～稼働準備という導入プロセスについて１つずつ解説しました。なにかとツール先行で導入が始まりやすいバグ票システムが，実はきちんと検討をしなければならない，そんなに簡単話では無いことをご理解いただけたのでは無いでしょうか。

第ＩＩＩ部ではバグ票についてのワーストプラクティスをなど，バグ票システムをより有効に利用するために参考となる情報を紹介しました。

最後に第ＩＶ部では，これまでを振り返り，また参考文献・URL をご紹介します。

冒頭にも述べた通り，本ガイドは情報量が少なくてもまず情報を伝えることが重要であると考え，スモールスタートで公開しております。今後は全体の情報の充実や，新たな章の追加など，継続的に改版を実施していく予定でおります。

是非，ご活用いただければ幸いです。

# 11 参考文献・URL

本ガイドを作成するにあたり参考にした文献やURL一覧を以下に示します。


[1] あかさた(著), 『実践バグ管理』, ソシム, 2009年

[2] 小川明彦(著), 阪井誠(著), 『Redmineによるタスクマネジメント実践技法』, 2010年

[3] IPA/SEC(発行), 『組込みソフトウェア開発における品質向上の勧め バグ管理手法編』, 2013年

[4] 株式会社アジャイルソフトウェア(著), 『Redmine実践ガイド 理論と実践 事例で学ぶ新しいプロジェクトマネジメント』, ソシム, 2015年

[5] 小川明彦(著), 阪井誠(著), 『チケット駆動開発―プロジェクトを成功に導くための「現場からの改善」提案を完全網羅』, 翔泳社, 2012年

[6] Redmine, http://www.redmine.org/

[7] Trac, https://trac.edgewall.org/

[8] Mantis, https://www.mantisbt.org/

[9] JIRA, https://ja.atlassian.com/software/jira

[10] 鈴木ら, 「バグレポートの改善に向けた問題事例の調査とアンチパターン作成」, SQiPシンポジウム2014, 2014年

[11] Cem kanerら, 「Bug Advocacy」, Context-Driven Press, 2015年

NaITE（長崎ＩＴ技術者会）

http://naite.swquality.jp/