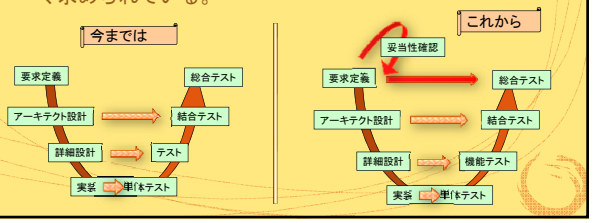
検証技法としてのモデル検証、その検証結果のフィードバックについて

財団法人福岡県産業・科学技術振興財団　片平典幸、孔維強
九州大学　福田晃、宮崎大学　片山徹郎、
キャッツ株式会社　渡辺政彦

なぜ検証（Verification)が必要か？

- 従来のソフトウェア開発における試験（Test)では設計仕様に基づいて作成されたロジックの正しさの確認になってしまい、“何がしたい”の正しさの確認と乖離している可能性がある。昨今、要求仕様に対する検証としてV&V（Verification&Validation=検証＆妥当性確認）が強く求められている。



検証技法としてのモデル検証

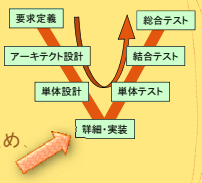
- まず振舞いモデル（オートマトン）として仕様を記述することにより、論理的な（数理的）検証を行うことができる。
- ソフトウェアの機能安全確保手法として、各分野においてその検証が重要視されている

モデル検証ツール一覧

ツール	モデル(形式)記述	特徴	出力検証結果	その他
Spin	Promela	オートマトン、LTL	標準出力(Text出力)	
nuSMV	状態遷移図(SMVコード)	CTL、BMC対応 BDDによる記号モデル	標準出力(Text出力)	
UPPAAL	状態遷移図(c++風)	時間付きCTL 連続時間オートマトン	GUI表示(シーケンス図)	
Vdm	vdm-SL vdm++	形式手法ツール	VdmTools(GUI)	Vdm-SLはISO標準(15048,61508)
Garakabu2	状態遷移表(ZIPC)	LTL、BMC対応 有限界オートマトン	GUI表示(状態遷移表と同期)	言語的モデル記述を行わない
Vdm	vdm-SL vdm++	形式手法ツール	VdmTools(GUI)	Vdm-SLはISO標準(15048,61508)

モデルベース開発（MBD)との親和性

- 粒度の大きい仕様での検証手法



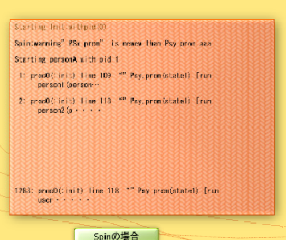
→上流工程での検証方法で
V字開発全体に貢献（妥当性確認）

→上流工程でモデル化（形式化）するため、
V字開発の左側が一貫して作業出来る
（要求開発から、
コード生成、シミュレーションまで）

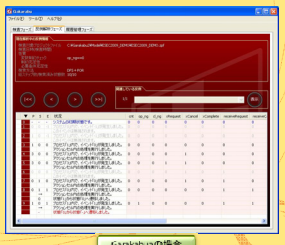
- 要求仕様に詳細なタイミングは書けない（ことが多い）
→逆に考えれば、“～のとき、その後～になる”などを
検証するには最適な作業フェーズである
→詳細なタイミングは別途下流で行うことで検証粒度を分ける

検証結果はどんなふうに見える？

- 不具合（反例）は、モデルに正しさとして修正しなければならない。
フィードバックの容易さは、検証技法自体の使用頻度に関わってくる。



Spinの場合



Garakabu2の場合

フィードバックの容易さ
＝検証者の敷居の低さ

- SpinやVdmなど自然言語的な検証（形式記述）では、記述・検証結果が“見える化”していないため、言語的フィードバックとなってしまい、下流設計では有効だが、振舞いが見えづらい、解りづらい。
- Garakabu2,UppaalなどのGUI化したモデル記述では、検証結果が視覚的に確認できるため、検証結果（反例）の修正が容易である。

まとめ

- 分かりやすいモデリング
  - ➔形式的に検証するためにはモデリングが必要
  - ➔自然言語的なものより“見える化”されているほうが仕様が分かりやすい
- 検証の重要性
  - ➔肥大化していくソフトウェアに対して、今後、上流工程での検証が重要性を増す
  - ➔ロジックの正しさではなく、要求仕様への正しさが重要
- 検証結果のフィードバック（検証とモデル修正）
  - ➔不具合（反例）を見つけても、モデル修正が難しくては現実的な開発手法として普及の障害になってしまう
  - ➔不具合の修正の容易さが検証・モデリングともに今後の課題

ご清聴ありがとうございました