Assurance Based Development

Patrick John Graydon
Assurance Based Development

Patrick John Graydon
This work begins with an observation:

It is not enough for critical software to be fit for use; we must know\(^1\) that it is fit for use

Assurance arguments can be used to provide the necessary assurance of fitness

---

\(^1\) To the extent possible.
Introduction (2)

That observation is followed by a second:

Producing an assurance argument after building software might be difficult: key evidence may be difficult to obtain

Key idea: Let the need to produce an assurance argument drive software development
Introduction (3)

• **Assurance Based Development** is a novel approach to constructing critical software

• Key ideas:
  - An assurance argument delivered with software should demonstrate that the software is fit for use
  - The need for evidence for this argument should drive development decisions
Uses of Argument in Engineering

• **Assurance arguments** can be used to show:
  
  – Safety (e.g. *safety arguments*)
    
    • Part of a complete *safety case*
  
  – Security (e.g. *security arguments*)
  
  – Dependability
  
  – Compliance with a standard
  
  – Other goals
    
    • For example, service continuity during upgrade

The Role of Argument

• **Prescriptive standards** demand evidence:
  — Test results, analysis reports, etc.
  — Independent reviews of test plans, configuration management procedures, etc.

• **Arguments** explain evidence:

  “Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained – it can be unclear that (or how) safety objectives have been satisfied.”

Can We Exploit Argument?

Argument tells us what is important

Is there a mechanism?

Artifacts provide services and evidence

Mechanism goal: generate the necessary artifacts (including software) and evidence
The ABD Concept

Software Requirements

Software Development

Required Evidence

Supplied Evidence

Software

Argument

Software Dependability Goals

Argument Development
Turning the Knobs to Maximum

- Extreme Programming “turned up the knobs”
- ABD turns up the knobs on rational development

<table>
<thead>
<tr>
<th>Good</th>
<th>➔</th>
<th>Better</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety arguments</td>
<td>➔</td>
<td>Broader argument</td>
</tr>
<tr>
<td>Early and often</td>
<td>➔</td>
<td>Continuous argument update</td>
</tr>
<tr>
<td>Rational choices</td>
<td>➔</td>
<td>A completely rational process</td>
</tr>
</tbody>
</table>
Comprehensive Argument

• Problem:
  – Adequate safety/security is not enough
  – Must address all goals of all stakeholders

• There will be tradeoffs
  – Must appear explicitly in the argument
    • Must be addressed
    • Must be seen to be addressed
    • Impact of each must be clear

• As a driver, safety argument does not do this
Software That Is Fit For Use

- Not just safety, but **fitness for use**:
  - Adequate safety, *and*
  - Adequate security, *and*
  - Desired functionality, *and*
  - *Everything else*

**Fitness for use:**
A system is fit for use if and only if it adequately addresses a balance of stakeholder concerns and that balance is itself acceptable to all stakeholders.
“Darn, the wings did fall off.”
Successful Development

- The *fitness argument* captures data about the *product*
- The *success argument* captures data about the *process*

- Development schedule, *and*
- Budget and other resource constraints, *and*
- Pragmatic development constraints
Main Argument Claims

- **Success Argument**
  
  **G_Success**
  The effort will lead to *acceptable software in acceptable time* and at *acceptable cost*.

- **Fitness Argument**
  
  **G_Fitness**
  The software is adequately *fit for use* in the context(s) in which it will be operated.

- **Notice Wording**

- **Notice Wording**

- **These claims encompass all possible stakeholder goals:**
  - If system *kills people* or *damages the environment*...
  - Or is safe but *doesn’t work*...
  - Or *violates regulations*, or engineering ethics...
  - Or is delivered *late*, or *costs too much*...

  The **Effort Is Not A Success**
Process Synthesis in ABD

Development

Success Argument

Fitness Argument

Obligation

Support

Synthesized Development Process

Obligation

Support

The list of development steps that will be performed, each with any relevant constraints
Process Synthesis: Select Goals to Address

• First step: select a goal or goals to address

  Pattern library and other literature

  Options

  Option 1
  Option 2
  Option n

  Experience (both personal and that of colleagues)

  Assurance obligations

  Assurance obligations

  Fitness argument

  Success argument

• Area of expertise
• Perceived risk of infeasibility
• Minimize interdependency
Process Synthesis: Gather Options

- Second step: gather options that meet the selected obligations

- Gathering options takes time
  - Must balance effort spent versus perceived risk of making a poor choice
Example Options

• Consider a hard-real-time system
• Obligations:
  – **Success**: Timing goals can be met demonstrably
  – **Fitness**: Real-time requirements met
• Options considered:
  – Analyze WCET using a tool to be chosen later
  – Utilize a watchdog timer to re-issue last frame’s control outputs if deadline would be missed
Process Synthesis: Evaluating Options

Consider:
- Functionality
- Dependability
- Restrictions on later choices

- Cost
- Schedule
- Feasibility
- Applicable standards
- Non-functional requirements
Evaluation of Example Option

• In the context of this effort, and given the particulars of each choice:
  – **WCET analysis**: Would supply *strong evidence* that the hard real-time deadlines would be met
  – **Re-issuing the last control outputs**: Unacceptable. How would we demonstrate that:
    • re-issuing the last frame’s outputs would be rare
    • doing so rarely would keep the system safe
If a choice is expected to produce evidence, the evidence added to the argument is marked as forthcoming (e.g. using a diamond in Goal Structuring Notation)
Process Execution and Repair

• The *process execution mechanism*:  
  – Guides when process steps are executed  
  – Guides responses to exceptions  
  – Handles incorporation of evidence produced  

• The *ABD repair mechanism*:  
  – Guides developers to repair argument and process whenever a problem arises
An Ideal Evaluation of ABD

• Evaluate *efficacy*
• A controlled experiment:
  – A team using ABD
  – Control team(s) using alternative(s)
• Replicated in order to cover:
  – Different team sizes and skill levels
  – Multiple application domains and effort sizes
• *This would be far too costly*
A Practical Evaluation of ABD

• Case study development of specimen software
  – Assess *feasibility*, not efficacy
    • Looked for evidence of specific ways in which ABD might be infeasible
  – Two different specimen systems
    • One safety-critical, one security-critical
  – Study protocol guides data collection
    • Author’s observations might be biased
    • However *systematic* data collection ensures data (whether good or bad) will not be overlooked
• Study 1: *The UVA LifeFlow LVAD MBCS*
  – Development of safety-critical software

• Study 2: *Repair study*
  – Assess ABD repair mechanism via fault-injection

• Study 3: *Limits of software dependability*
  – What fundamental limits does argument reveal?

• Study 4: *Tokeneer*
  – Development of security-critical software
Feasibility Sub-claims

• Look for evidence of infeasibility:
  – Generally
  – *Choice order* might be infeasible
  – *Decision criteria* might be missing or superfluous
  – *Delaying decisions about how to complete repair* to later process synthesis might be infeasible
  – *ABD might be cost-prohibitive*:
    • Might take too long to assemble and assess options
    • Might take too long to create and maintain arguments

• Look for evidence that choices cannot be judged by impact on arguments alone
Case Study 1: The UVA LifeFlow LVAD

- Left Ventricular Assist Device
- Continuous-flow axial design
- Magnetic bearings
- Less blood damage than current models!
• Compute control updates in hard-real-time (5 kHz)
  – State-space control model, 16 states
• No more than $10^{-9}$ failures per hour of operation
Resulting Synthesized Process

• **Formal specification** in PVS
• Design a cyclic executive to manage the real-time tasks
• Design bearing control task routines
• **Implement** MBCS in **SPARK Ada** (2,510 lines)
• Implement bootstrap (106 assembly instructions)
• Use AdaCore's GNAT **Pro High-Integrity Edition** compiler
• **Formally verify the implementation**
  — Used Echo approach, PVS, and SPARK Tools
• Analyze Worst-Case Execution Time (**WCET**) and stack usage
• Requirements-based **functional testing** to Modified Condition / Decision Coverage (MC/DC) (not completed)
Prototype Implementation
The system adequately solves the problem it is intended to solve in the context it is intended to be operated in.

**G_Fitness**

**G_ArgByIndSubArgs1** Argument by independent sub-arguments

**ST_ArgByIntegrationTesting** Argument by appeal to integration testing

**ST_ArgBySatisfactionOfReqs** Argument by appeal to demonstrable satisfaction of requirements

**G_ArgByIndSubArgs2** Argument by independent sub-arguments

**G_RealTimeReqsSatisfied** The system satisfies its real-time requirements

**G_NonRealTimeReqsSatisfied** The system satisfies its non-real time requirements

**G_TestingShowsNon-TimingReqs Satisfied** Testing shows that the system satisfies its non-timing requirements

**G_CodeRefinesSpec** The code refines the non-real-time portions of the specification

**G_ReqSatisfied** The delivered system satisfies its requirements

**G_LLSpecRefinesFormalSpec** The low-level specification refines the formal portion of the specification

**G_TransformedCodeRefinesLLSpec** The transformed code refines the low-level specification

**G_TransformedCodeSemanticEquivalence** The transformed codeso is semantically equivalent to the code

**G_CodeRefinesSpec** The code refines the non-real-time portions of the specification

**G_ExecRefinesCode** The executable refines the code

**G_ReqSatisfied** The delivered system satisfies its requirements

**G_ArgByIndSubArgs2** Argument by independent sub-arguments

**ST_ArgByIntegrationTesting** Argument by appeal to integration testing

**ST_ArgBySatisfactionOfReqs** Argument by appeal to demonstrable satisfaction of requirements

**G_RealTimeReqsSatisfied** The system satisfies its real-time requirements

**G_NonRealTimeReqsSatisfied** The system satisfies its non-real time requirements

**G_TransformedCodeRefinesLLSpec** The transformed code refines the low-level specification

**G_TransformedCodeSemanticEquivalence** The transformed code is semantically equivalent to the code

**G_CodeRefinesSpec** The code refines the non-real-time portions of the specification

**G_ExecRefinesCode** The executable refines the code

**G_RealTimeReqsSatisfied** The system satisfies its real-time requirements

**G_NonRealTimeReqsSatisfied** The system satisfies its non-real time requirements

**G_ArgByIndSubArgs1** Argument by independent sub-arguments

**Context elaborations**

**Legend:**
- A goal (claim)
- An argument strategy
- A solution (evidence)
- An assumption
- Contextual information
- In the context of

**ST_ArgOverRefinement** Argument by showing that successive forms refine an original

**G_SpecRefinesRequirements** The specification refines the requirements

**G_CodeRefinesSpec** The code refines the non-real-time portions of the specification

**G_ExecRefinesCode** The executable refines the code

**G_ReqSatisfied** The delivered system satisfies its requirements

**G_ArgByIndSubArgs2** Argument by independent sub-arguments

**ST_ArgByIntegrationTesting** Argument by appeal to integration testing

**ST_ArgBySatisfactionOfReqs** Argument by appeal to demonstrable satisfaction of requirements

**G_RealTimeReqsSatisfied** The system satisfies its real-time requirements

**G_NonRealTimeReqsSatisfied** The system satisfies its non-real time requirements

**G_TestingShowsNon-TimingReqs Satisfied** Testing shows that the system satisfies its non-timing requirements

**G_CodeRefinesSpec** The code refines the non-real-time portions of the specification

**G_ExecRefinesCode** The executable refines the code

**G_ReqSatisfied** The delivered system satisfies its requirements

**G_ArgByIndSubArgs2** Argument by independent sub-arguments

**ST_ArgByIntegrationTesting** Argument by appeal to integration testing

**ST_ArgBySatisfactionOfReqs** Argument by appeal to demonstrable satisfaction of requirements

**G_RealTimeReqsSatisfied** The system satisfies its real-time requirements

**G_NonRealTimeReqsSatisfied** The system satisfies its non-real time requirements

**G_ReqSatisfied** The delivered system satisfies its requirements
Fitness argument

• 348 GSN elements
• Widest step: 5 child elements
• Longest path: 26 elements
• General form:
  – Integration testing \textit{and}
  – Appeal to satisfied requirements
    • Timing demonstrated by WCET analysis
    • Functionality demonstrated by testing \textit{and} formal proof
\section*{Success Argument}

\begin{itemize}
\item \textbf{C. System} \hspace{1cm} “System” is the magnetic bearing control software \\
\hspace{1cm} DC-001
\item \textbf{C. OperatingContext} \hspace{1cm} The system is a component of the LifeFlow LVAD First Prototype \\
\hspace{1cm} DC-001
\item \textbf{C. AcceptableCost} \hspace{1cm} Presently available resources and staff plus target hardware costs \\
\hspace{1cm} DC-001
\item \textbf{C. Requirements} \hspace{1cm} Requirements imposed by the LifeFlow LVAD First Prototype are recorded in /project_docs/requirements/tags/v.00.05.02_20090716/vad_cntrl_sw.reqs.pdf \\
\hspace{1cm} DC-001
\item \textbf{C. AcceptableTime} \hspace{1cm} LifeFlow LVAD First Prototype delivery date \\
\hspace{1cm} DC-001
\item \textbf{C. AssumedDepGoals} \hspace{1cm} The assumed dependability goals, as stated in /project_docs/assumed_dep_goals/tags/v00.01_20081212/assumed_dep_goals.pdf \\
\hspace{1cm} DC-010
\item \textbf{G. Success} \hspace{1cm} The development effort will lead to an adequate system in acceptable time and at acceptable cost \\
\hspace{1cm} C. DevelopmentSchedule \hspace{1cm} The development schedule as recorded in /project-docs/plan/trunk/vad_ctrl_sw.plan \\
\hspace{1cm} DC-003
\item \textbf{G. PlanDeliversOnTime} \hspace{1cm} The development effort will be completed on time \\
\hspace{1cm} DC-003
\item \textbf{G. SchedeuledDeliveryDateAccurate} \hspace{1cm} The schedule accurately predicts the delivery date \\
\hspace{1cm} DC-003
\item \textbf{G. SchedeuledDeliveryDateAcceptable} \hspace{1cm} The predicted delivery date is acceptable \\
\hspace{1cm} DC-003
\item \textbf{G. DevRisksMitigated} \hspace{1cm} All credible development risks have been adequately mitigated \\
\hspace{1cm} DC-002
\item \textbf{G. DevRisksEnumerated} \hspace{1cm} All credible development risks have been enumerated \\
\hspace{1cm} DC-002
\item \textbf{G. OverFlowRiskMitigated} \hspace{1cm} The risk that we will be unable to demonstrate freedom from overflow has been adequately mitigated \\
\hspace{1cm} DC-004
\item \textbf{G. HardwareIOReqsRiskMitigated} \hspace{1cm} The risk that the hardware I/O code given in the requirements is erroneous has been adequately mitigated \\
\hspace{1cm} DC-004
\item \textbf{G. MemoryInadequateRiskMitigated} \hspace{1cm} The risk that we will not be able to fit the code and data into the available memory has been adequately mitigated \\
\hspace{1cm} DC-004
\item \textbf{G. FlashProgrammingRiskMitigated} \hspace{1cm} The risk that we will not be able to program the MPC5554's flash memory with the tools we already have has been adequately mitigated \\
\hspace{1cm} DC-004
\item \textbf{G. ArgOverRisks} \hspace{1cm} Argument over all credible development risks \\
\hspace{1cm} DC-002
\item \textbf{G. ReqLateRiskMitigated} \hspace{1cm} The risk that late delivery of a pending portion of the requirements will preclude success has been adequately mitigated \\
\hspace{1cm} DC-011
\item \textbf{G. DepGoalsUnacheivableRiskMitigated} \hspace{1cm} The risk that the as-yet unknown dependability goals will be unachievable has been adequately mitigated \\
\hspace{1cm} DC-010
\item \textbf{G. TimingGoalsUnachievableRiskMitigated} \hspace{1cm} The risk that we will be unable to demonstrably meet the timing goals has been adequately mitigated \\
\hspace{1cm} DC-010
\item \textbf{G. ProgramInadequateAccuracyRiskMitigated} \hspace{1cm} The risk that we will not be able to demonstrate sufficient accuracy in the control calculations has been adequately mitigated \\
\hspace{1cm} DC-004
\item \textbf{G. PoorPlanningRiskMitigated} \hspace{1cm} The risk that following a poor development plan will waste effort has been adequately mitigated \\
\hspace{1cm} DC-002
\item \textbf{G. InadequateAccuracyRiskMitigated} \hspace{1cm} The risk that we will not be able to demonstrate sufficient accuracy in the control calculations has been adequately mitigated \\
\hspace{1cm} DC-004
\end{itemize}
Success Argument

- 49 GSN elements
- Widest step: 10 child elements
- Longest path: 9 elements
- General form:
  - Appeal to planning *and*
  - Argument over enumerated development risks
- Evolution:
  - Starts small, grows early, becomes moot
## Case Study 1 Results

<table>
<thead>
<tr>
<th><strong>Feasibility?</strong></th>
<th>No difficulties observed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Are obligations appropriate choice drivers?</strong></td>
<td>28 choices:</td>
</tr>
<tr>
<td></td>
<td>• 19 followed direct reasoning</td>
</tr>
<tr>
<td></td>
<td>• 6 others addressed an obligation</td>
</tr>
<tr>
<td></td>
<td>• 2 addressed unnoted development risk</td>
</tr>
<tr>
<td></td>
<td>• 1 remaining case was an implicit choice</td>
</tr>
<tr>
<td><strong>Can options be judged by argument impact alone?</strong></td>
<td>We observed no value that could not be represented in the fitness or success argument</td>
</tr>
<tr>
<td><strong>Missing or superfluous decision criteria?</strong></td>
<td>We observed that impact on schedule was not covered in the choice criteria; it has been added</td>
</tr>
</tbody>
</table>
Case Study 4: Tokeneer

• National Security Agency challenge problem
• Secure enclave protection system
• Originally developed by Altran Praxis\(^1\) in 2003
  – Demonstrated use of SPARK Ada and Correctness-By-Construction to comply with Common Criteria
• Re-implemented in order to study ABD
  – Limited scope to a small system subset (1 scenario)
  – Assumed formal specification given
  – Assumed that the customer demands the use of C
  – Focus on breadth (demonstrate compliance with as many Common Criteria requirements as practicable)

1. Altran Praxis were known as Praxis High-Integrity Systems in 2003
The Tokeneer ID Station

- Secure Enclave
  - Tokeneer ID Station computer

- Administrator's console

- Door equipped with latch actuator and position sensor

- Token (smart card) reader, display, and fingerprint scanner
The Common Criteria

CC Part 1: Introduction and General Model

## The Synthesized Tokeneer Process

| Write Configuration Management Plan | Complies with applicable SARs  
29 pages, 29 configuration items |
|-------------------------------------|--------------------------------------------------------------------------------|
| Write Policies and Procedures       | Defect tracking policy and reporting system  
Life-cycle design document complying with applicable SARs |
| High-Level Design                   | Functional behavior formalized in \textit{Z}  
121 pages, 136 \textit{Z} schemas  
Internal consistency checked with \textit{fuzz} and inspections*  
Correspondence checked with:  
• Traceability matrix  
• Hand proofs (completed 1) |
| Low-Level Design                    | Functional behavior formalized in \textit{Z}  
133 pages, 130 \textit{Z} schemas  
Internal consistency checked with \textit{fuzz} and inspections*  
Correspondence to HLD checked as with HLD |

* Inspections were defined but informal reviews done instead due to staff availability
The Synthesized Tokeneer Process (2)

| Implementation | One scenario use case (user enters with valid token)  
|                | Used **MISRA** subset of C (compliance checked by tool)  
|                | 10 modules, **1,756 pLOC**, 1,155 ILOC |

| Annotations and Static Checking | • Documented pre- and post-conditions with assert (to the degree practical)  
|                                | • **Splint** annotations for pointer use, global variable use  
|                                | • All Splint checks pass |

| Unit Testing | Developed unit test suite for most complicated module  
|             | Achieved 100% statement, 100% branch coverage |

| Integration | Developed integration tests for one module pair  
|            | Covers the implemented use case scenario  
|            | Achieved 96.7% statement, 91.7% branch coverage (mod 1)  
|            | Achieved 90.5% statement, 65.0% branch coverage (mod 2) |
## Breakdown of Effort

<table>
<thead>
<tr>
<th>Activity</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaborate CC portions of fitness argument</td>
<td>19.5h</td>
</tr>
<tr>
<td>Assemble, assess, and choose from options</td>
<td>9.2h</td>
</tr>
<tr>
<td>Modify the assurance arguments to reflect choices</td>
<td>13.1h</td>
</tr>
<tr>
<td>ABD repairs</td>
<td>3.6h</td>
</tr>
<tr>
<td>Argument review and risk assessment</td>
<td>8.0h</td>
</tr>
<tr>
<td>Develop life-cycle description, configuration management plan, and project standards documents</td>
<td>18.8h</td>
</tr>
<tr>
<td>Tool provisioning</td>
<td>5.0h</td>
</tr>
<tr>
<td>Develop high-level design</td>
<td>52.5h</td>
</tr>
<tr>
<td>Develop low-level design</td>
<td>29.0h</td>
</tr>
<tr>
<td>Implementation</td>
<td>17.8h</td>
</tr>
<tr>
<td>Unit testing (2 modules)</td>
<td>31.3h</td>
</tr>
<tr>
<td>Integration testing (2 modules)</td>
<td>4.3h</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>212.4h</strong></td>
</tr>
</tbody>
</table>
## Defects Found

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Defect</th>
<th>Found during</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>Wrong schema named in state diagram</td>
<td>High-Level Design</td>
</tr>
<tr>
<td>High-Level Design</td>
<td>Logic error x2 and misc. typos</td>
<td>Review</td>
</tr>
<tr>
<td></td>
<td>Logic error x2 and misc. typos</td>
<td>Low-Level Design</td>
</tr>
<tr>
<td></td>
<td>Typos in C-language interface specs</td>
<td>Implementation</td>
</tr>
<tr>
<td>Low-Level Design</td>
<td>Typos in C-language interface specs</td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>Referenced old HLD version</td>
<td>Implementation</td>
</tr>
<tr>
<td>Implementation</td>
<td>Post-condition assertion incorrect</td>
<td>Unit Testing</td>
</tr>
<tr>
<td></td>
<td>Logic error x2</td>
<td>Unit Testing</td>
</tr>
<tr>
<td>MISRA Checker</td>
<td>Flags all function names as using standard library prefixes</td>
<td>Implementation</td>
</tr>
</tbody>
</table>
Success and Fitness Arguments

• Fitness argument:
  – 611 elements
  – Compliance with SARs is 41% of argument
  – Compliance arguments frequently shallow
    • E.g. existence of test plan solved by test plan

• Success argument:
  – 86 elements
  – Arranged as argument over development risk
# Case Study 4 Results

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility?</td>
<td>No difficulties observed</td>
</tr>
<tr>
<td><strong>Obligations as choice drivers?</strong></td>
<td>40 choices:</td>
</tr>
<tr>
<td></td>
<td>• 39 followed direct reasoning</td>
</tr>
<tr>
<td></td>
<td>• 1 other addressed an obligation</td>
</tr>
<tr>
<td><strong>Argument impact as option assessment?</strong></td>
<td>We observed no value that could not be represented in the fitness or success argument</td>
</tr>
<tr>
<td>Decision criteria?</td>
<td>No evidence of missing or superfluous criteria</td>
</tr>
<tr>
<td><strong>ABD process synthesis time?</strong></td>
<td>9.2 hours spent making choices</td>
</tr>
<tr>
<td></td>
<td>Process nearly complete</td>
</tr>
<tr>
<td></td>
<td>If a full implementation took 1,500 hours...</td>
</tr>
<tr>
<td><strong>Argument creation time?</strong></td>
<td>32.6 hours spent writing argument</td>
</tr>
<tr>
<td></td>
<td>Very little tool assistance</td>
</tr>
<tr>
<td></td>
<td>Again, little more needed for full implementation</td>
</tr>
</tbody>
</table>
Observations

• Arguments reveal important subtleties
  – Suppose that you employ full formal verification
    • High confidence that the code refines the specification
    • Does the executable refine the code?
    • Does your testing confirm correct compilation?
• Perfect deductive arguments are not possible
  – Inductive leap required by imperfect test coverage
  – Human beings depended upon to follow protocols
  – Etc.

Conclusions

• Args facilitate evidence adequacy judgments
  – The limitations case study illustrated this
• Success args can record process rationales
  – Two examples are given in the my thesis
• ABD is feasible and yields evidence of fitness
  – Arguments included all important values
  – Choice order was feasible
  – Decision criteria (after amendment) were acceptable
  – Repair mechanism feasible
  – ABD cost overhead is not prohibitive

Questions?