CeBASE

Evolving Knowledge About COTS

Victor Basili
University of Maryland
and
Fraunhofer Center - Maryland

COTS Issues

What are COTS?

What is the right COTS based development process or processes?

What techniques and technologies support CBS?

What are the CBS trade-offs?

What are the dependability, performance, … issues associated with COTS?

What is the state of our current knowledge with COTS and how has it been recorded, integrated for use?

How do we improve the COTS based development product and process?

How can empirical studies help?

Motivation for the Empirical Research

- Industry needs a basis for
  - choosing among life cycle models and development approaches
    - COTS/legacy/agent/portfolio-based systems
    - Rapid/evolutionary/spiral/adaptive development
    - Open-source; extreme programming; architecture-based development
  - tailoring them for specific needs
    - testing for detecting a specific defect class
    - designing for evolving a particular feature

Example of Using Empirical Results for development, research, education

Example: When are peer reviews more effective than functional testing?

- **Peer reviews** are more effective than functional testing for faults of **omission** and **incorrect specification** (Based upon studies at UMD and USC)

Implications for empirically based **software development process**:

- If , for a given project set, there is an expectation of a larger number of faults of omission or incorrect facts than use peer reviews.

Implications for **software engineering research**:

- How can peer reviews be improved with better techniques and technology to identify faults of omission and incorrect fact?

Implications for **software engineering education**:

- Teach how to experiment with and choose the appropriate analytic techniques

Empirical Goal for COTS Based Development

What and how can we
learn from our past experiences and use them
to better acquire the appropriate COTS products
and
integrate them to build a system that
satisfies our project requirements
given cost and schedule constraints?

Measurable COTS Goals

- Analyze the proposed ***COTS products***
  - for the purpose of evaluating
  - with respect to appropriateness for purpose
  - from the point of view of the project
  - in the context of the development organization.
- Analyze the ***COTS Based Development process***
  - for the purpose of improving its effectiveness
  - with respect to some attribute measure
  - from the point of view project goals
  - in the context of the development environment

What can empirical study do for the process developer?

- Help formulate measurable hypotheses and goals
  - What are the process goals and expectations?
  - How might they be measured?
- Help validate expectations/hypotheses about the processes
  - How would the process be characterized in the context of the project?
  - What early validations might be defined?
- Help provide feedback for improving the processes
  - What are the essential strengths and shortfalls?
- Help match the processes with the right projects

What can empirical study do for the project?

- Help identify what product properties are desired
  - Who are the stakeholders?
  - What are their needs? What are their product goals?
  - What are the project evaluation criteria?
  - What product models are available?
- Help identify what the project should choose to achieve those needs
  - What processes are available?
  - How are they characterized?
  - How might the project tailor and integrate various processes to achieve its needs?
- Help the project tailor and apply a process

What is required?

- Characterization of the process
- Characterization of the project
- Characterization of the project goals
- An empirical evaluation process
- An Experience Base

Knowledge needed about the process

- Characteristics/attributes
  - General:
    - Purpose (e.g., defect detection, design support …)
    - System (product & environment) where suitable (real-time systems, COTS based systems…)
    - Phase/activity when applied (design, testing …)
    - On what artifacts (design, class diagrams, code, test cases …)
    - By whom (developer, tester …)
    - Effort/cost of application (staff hours)
  - Product-related Attributes
    - Hypotheses about the technology capability with respect to dependability properties (reliability, availability, safety, …)
    - Evidence of past assessment

Example technology: glue code reading

- Characteristics/attributes
  - General:
    - Purpose: defect detection, specifically related to issues in integrating COTS with custom code
    - System: any COTS-based system
    - Phase: code, activity: inspections
    - On what artifacts: code and design
    - By whom: developer
    - Effort/cost of application: 4 hours/ 400LOC of glue code
  - Product-related Attributes
    - Hypotheses: Increase reliability and availability of the system, decreases the number of failures caused by COTS integration, decreases the expert reviewer effort required to find faults
    - Evidence of past assessment: In controlled experiments, the technology increased the defect detection by 30%

Knowledge needed about the project

- Characteristics/attributes
  - General:
    - System: Function, size, type (mission critical, real-time, embedded, web-based, …),
    - Development: Life-Cycle model (Waterfall, Spiral, …), Reuse (COTS based, …), Contract (in-house, out-sourced, …), language, …
    - Use Profile: number of users, sites, platforms, operational profile, lifetime, …
    - Organization Experience: with COTS products, ….
    - State: operational, under construction, at requirements, …
  - Product-related attributes
    - Availability requirements and issues, Reliability requirements and issues, Maintainability Requirements and Issues, …
    - Models used

Example project: EOSDIS

- Characteristics/attributes
  - General:
    - System: space data acquisition, distribution and archiving, 1.1 MLOC, distributed, real-time components, client server
    - Development: Incremental, COTS-intensive, Contracted, C++ and Java
    - Use Profile: hundreds of users, 4 data archiving sites, Unix servers, 24 / 7 / 365, lifetime: 1999 – 2015
    - Developer profile: current expertise in COTS use, …
    - State: operational, under maintenance and evolution
  - Product-related attributes
    - RMA requirements: operational availability: 0.96, MDT: four (4) hours or less,…
    - RMA issues: user needs and developer (contract) requirements different, COTS updates affect overall RMA, …
    - Models used: system models do not apply well to software, …

Knowledge needed about the product goal models

Characteristics/attributes:

- Property (e.g., reliability, availability, safety …)
- Object (design, code, …)
- Purpose (estimation, assessment)
- Dependability perspective (user, developer, tester …)
- Assessment technique (analytical, simulation, measurement)
- Measurement model description
  - Type, Inputs/Outputs, Approach, Assumptions
- Phase/activity where applicable (testing, coding …)
- System where applicable (telecommunications, safety critical, ..)
- Maturity
- Accuracy
- Cost
- Known limitations, Problems

Example Model: Reliability

- *Model* Reliability growth
- *Dependability attribute* Reliability
- *Purpose* Prediction
- *Object* Product
- *Dependability perspective* Tester
- *Assessment technique* Measurement-based; Controlled experimentation
- *Measurement model*
  - *Type* Quantitative
  - *Inputs* Usually the number of defects at the beginning of testing; the failure rate at the beginning of testing
  - *Outputs* Failure rate evolution in time
  - *Model description* Each model has a specific formula or set of formulae
- *Phase/activity where applied* System testing phase; applied on code/system
- *Maturity* These models have been applied since the '80s on many projects (e.g. Musa's model has been used in telecommunication)
- *Accuracy* Need to get more info
- *Cost* Need to get more info
- *Known limitations, Problems* They have assumptions that might not be true (e.g. all failures are similar, all failure occurrences are independent, etc.)

Example Outcome: OO Inspections

- A process for inspections of Object-Oriented designs was developed using multiple iterations through this method.
  - Early iterations concentrated on **feasibility**: Collected data concerning *effort required, results due to the process* in the context of *offline, toy systems.* Tested hypotheses about whether further effort spent would be justified.
  - Mid-process iterations concentrated on **usability**: Collected data concerning *usability problems, results due to individual steps* in the context of *small systems in actual development.* Tested hypotheses concerning the best ordering and streamlining of process steps to best match user expectations.
  - Most recent iterations concentrated on **effectiveness**: Collected data concerning *effectiveness compared to other inspection techniques previously used by developers* in the context of *real systems under development.* Tested hypotheses about whether the new techniques actually represented a usable improvement to practice.

Empirical activities needed to characterize and assess processes and products

- Formulating new and/or evolving existing hypotheses regarding software development decisions.
- Collecting empirical data and experiences.
- Recording influencing variables.
- Building models
  - Lessons learned
  - Heuristics/patterns
  - Decision support frameworks
  - Quantitative models and tools
- Integrating models into a framework.
- Testing Hypotheses by Application
- Building and Experience Base

CeBASE Empirical Research Activities

Empirical Data and Experiences Collection

The Center for empirically-Based Software Engineering (CeBASE) accumulates **empirical models** to provide validated guidelines for selecting techniques and models

CeBASE works on:

- Integrating existing data and models
- Initially focusing on new results in two high-leverage areas
  - **COTS Based Development**
  - **Defect Reduction**
  - **Agile Development**

The CeBASE strategy is to build an empirical **experience base** continuously evolving with empirical evidence to help us identify what affects cost, reliability, schedule,...

Building an Experience Base: Dust to Pearls Approach

Main principle: Capture the knowledge "dust" that developers use and exchange on a daily basis, and make it available throughout the community

- *Immediately* with minimal modification
- *Long-term* after analysis and synthesis.

The mechanisms used are based on

- Dust: The AnswerGarden Approach (Short-term needs based, Ad Hoc!, organic growth, fine-granular items)
- Pearls: The Experience Factory Approach (Long term needs, analysis&synthesis, feedback loops, separate groups)

Analyze and synthesize the dust and turn it into knowledge pearls

Building an Experience Base: Dust to Pearls (Light-Weight) Approach

How do you seed an empty Experience Base?

- Build a warehouse of knowledge
- Build the base and evolve in real-time (light-weight)

Light-weight approach based on short cycles in the QIP:

- Invest less now, harvest some now,
- Evaluate, improve
- Invest more later, harvest more later

An approach that

- Encourages experts to contribute to the Experience Base by quickly adding value and feeding back results
- Is light-weight enough not to add any significant work to already busy experts
- Allows the Experience Factory Group (CeBASE team) to analyze and evolve the content of the Experience Base over time

Building an Experience Base: Dust-to-Pearl Examples

In CeBASE we use the approach in several different ways:

Incident -> Captured COTS Lessons Learned -> set of COTS LLs -> Best Practices

Question -> Answer in E-mail -> FAQ -> set of FAQs -> Process Description

Defect -> Bug Report -> set of Bug Reports -> Problem areas -> Solutions -> Design Rules -> Development Practices

eWorkshop Chat statements -> Real time analysis -> Summary -> Best Practices, Lessons Learned (the following presentation)

Motivation for eWorkshops

- Goal is to learn from the experience of others in terms of problems and solutions and build an empirical **experience base** of continuously evolving empirical evidence we can elicit, share, and integrate knowledge and data from experts in the field

- Meetings among experts are a classical way of creating and disseminating knowledge. By analyzing these discussions, knowledge can be created and knowledge can be shared.
- But:
  - Experts are spread all over the world and it is hard to get people to travel to meet
  - Workshops are usually oral presentations and discussions that generally are not captured for further analysis
  - Certain personalities often dominate a discussion

Motivation for eWorkshops

- To overcome these problems, we designed the concept of the *eWorkshop*, using the facilities of the Internet

- eWorkshop:
  - an on-line meeting using a Web-based chat-application, that uses simple collaboration tools, minimizing potential technical problems and decreasing the time it takes to learn the tools
  - requires a defined process, a set of roles and a control room

CeBASE

The eWorkshop Process (Organizing Team View)

Organization of the workshop follows a protocol :

1. Choose a topic of discussion
2. Invite participants
3. Distribute Pre-meeting information sheet
4. Establish meeting codes – for meeting analysis
5. Publish synthesized info from pre-meeting sheets
6. Schedule pre-meeting training on tools
7. Set up control room
8. Conduct meeting
9. Post-meeting analysis and synthesis and storage
10. Dissemination of packaged knowledge

The eWorkshop Tools

- The main tool is the web-based chat-application, adapted from some open source software

- The **chat tool** allows participants to
  - be located remotely from the control room
  - create identities based upon their names
  - submit statements to a message board
  - carry on a discussion online by responding to statements on the message board following a set of conventions
  - vote and measure level of consensus

- All statements are automatically captured in real-time, allowing them to be analyzed in real-time and afterwards in more depth

COTS eWorkshops

- History
  - Started from Top-10 list of Basili&Boehm in *IEEE Computer*
  - Subjected heuristics to scrutiny by experts
  - Based on discussion, heuristics could be
    - Refined
    - Added
    - Restated
    - … or a meta-statement could be made about the state of the knowledge
- Ongoing goal:
  - Revised / edited list of heuristics…
  - … that summarizes the state of the knowledge and indicates the level of confidence
  - … which can be compared to other data and continually expanded

COTS LL Repository Populating

- Initially, CeBASE collects LLs from
  - literature
  - eWorkshops
  - SEI collection of LLs
- Once repository is in use, Users
  - submit their LLs
  - submit questions; they receive answers from experts that will be captured and transformed in LLs
- Content evolution and refinement - based on:
  - New LLs
  - Users' feedback
  - Users' questions and experts' answers
  - Metrics collection and analysis
  - Content analysis/knowledge discovery

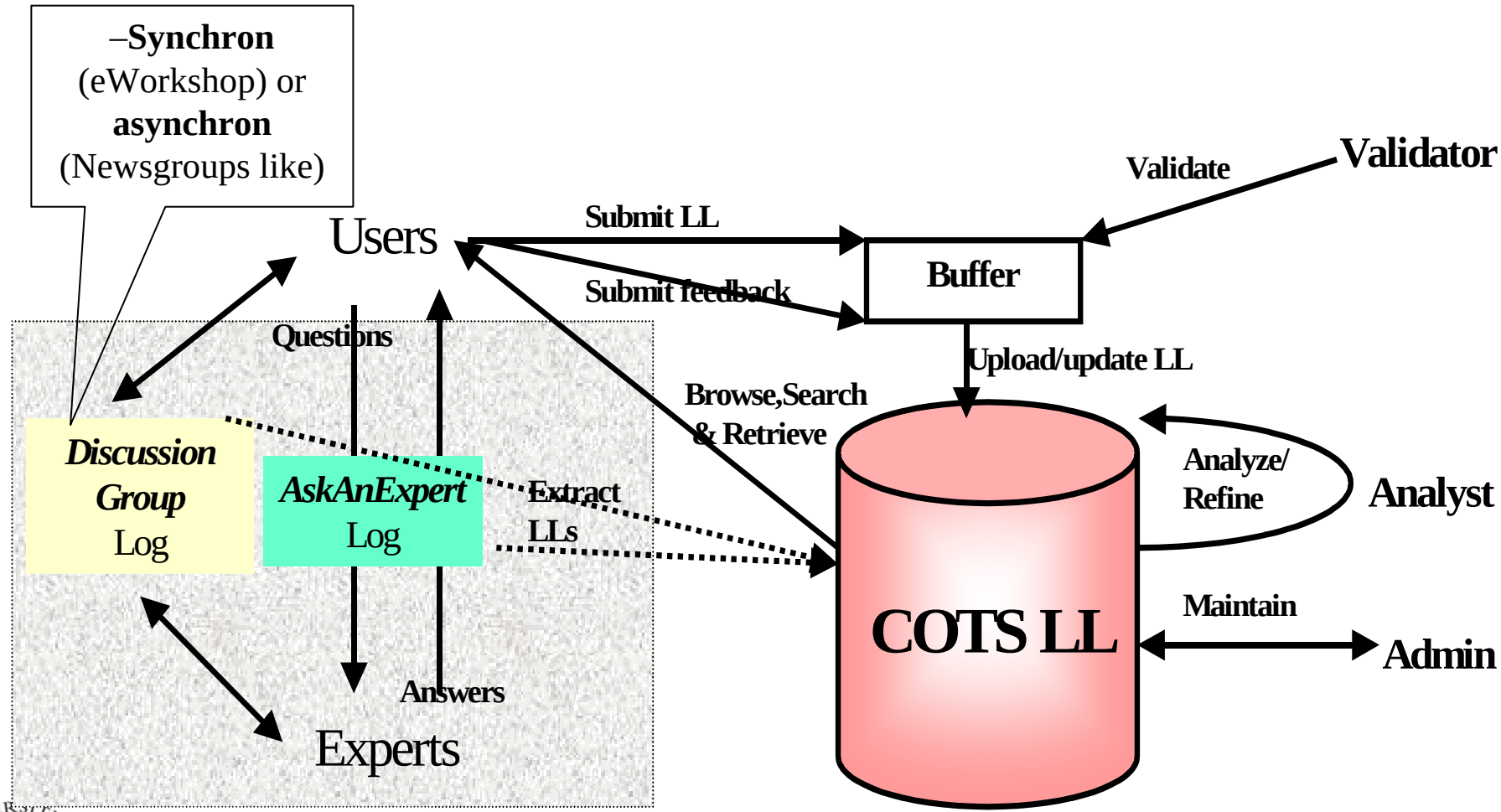
COTS LL Example (1/2)

- *Type*: Good practice
- *Statement*: For large, multi vendor solicitations: hold pre-award hearings so that each vendor will have an opportunity to ask questions and all vendors will hear the same response
- *Issue/Risk factor*: Vendor protest situation
- *Recommended action*: Hold pre-award hearings so that each vendor will have an opportunity to ask questions and all vendors will hear the same response
- *Comments*: With RFP on street without a pre-award hearing, one vendor submitted 5 pages of technical question irrelevant to the solicitation. When we refused to answer all questions and explained irrelevance, we learned that the vendor intended to protest award if not awarded contract. To avoid protest, we pulled RFP.

COTS LL Example (2/2)

- *Aspect*: Managerial
- *Object*: Vendor
- *Life-cycle Phase*: Solicitation, acquisition
- *Recommended audience*: Program manager
- *Type of system*: ERP
- *Type of company*: Unknown
- *Number of COTS per project:*1
- *Type of COTS*: Unknown
- *Type of data*: Qualitative

COTS LL Repository
Process



COTS LL Repository
End-user Capabilities

- Search/retrieve LLs by keyword (text) or combination of attributes
- Browse repository
- Submit new LLs
- Send feedback
  - About a specific lesson
  - About the repository

Evolving COTS Hypotheses

Hypothesis Formation: Top 10 Hypotheses

- H1: *99% of all executing instructions come from COTS products.*
- H6: *Although glue-code development usually accounts for less than half the total CBS software development effort, the effort per line of glue code averages about three times the effort per line of developed-applications code*
- H8: *CBS assessment and tailoring efforts vary significantly by COTS product classes – operating systems, database management systems, user interface, device driver and so forth.*
- Hypothesis 9: *Personnel capability and experience remain the dominant factors influencing CBS-development productivity.*
- H10: *CBS is currently a high-risk activity, with effort and schedule overruns exceeding non-CBS software overruns. Yet many systems have used COTS successfully for cost reduction and early delivery.*

Evolving COTS Hypotheses
Some COTS Projects at NASA/GSFC

We conducted some structured interviews at NASA/GSFC

Characterized the 6 projects around several nominal and ordinal variables

- Functionality: Class of function
- Criticality: effect of defects on human or economic loss.
- Size: effort spent on the project
- Schedule: constraint conditions
- Architecture: Amount of COTS to be integrated
- Functionality: reused from COTS. % of product covered by COTS
- Result: was experience of using COTS a positive or a negative?

Projects ranged from small to large in size, using one or more COTS, most with an aggressive schedule, one with a normal schedule, all projects have a low or medium criticality

  - clearest limitation of sample is that there are no projects with high or very high criticality..

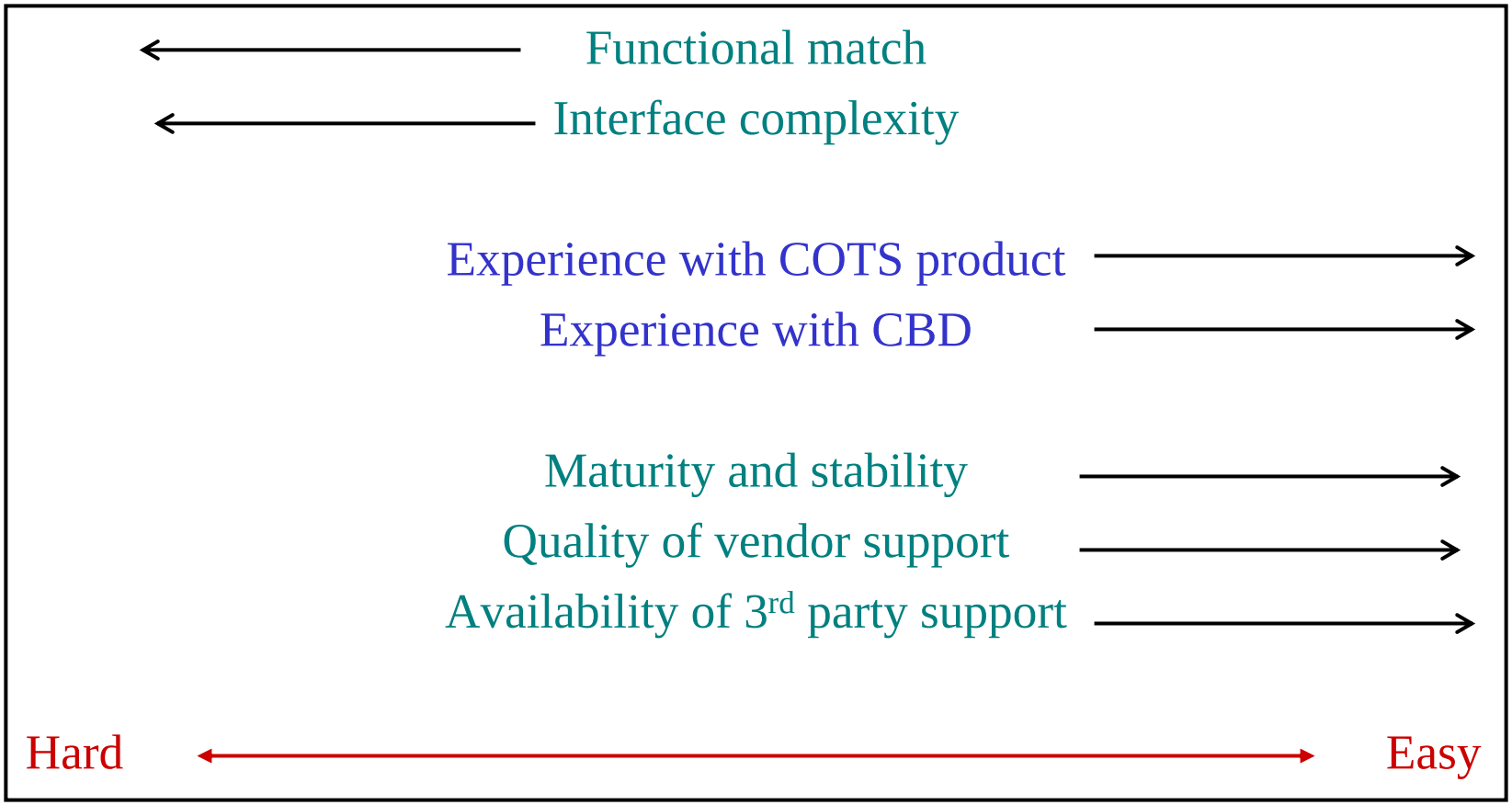
Evolving COTS Hypotheses
How are COTS defined?

- Product where the buyer has no access to the source code; the vendor controls its development; the product has a non-trivial installed base
  - Did not consider products (compilers, CASE tools) used to produce the final deliverable, but not integrated with it.
- COTS includes products such as operating systems, databases, and network protocols, i.e., well known products, used for years, there is experience, consulting and training about them easily available, and in most cases a second source vendor or product is available
  - These COTS not likely to raise problems, the problems they raise have already been solved or well understood how to solve them
  - Project managers tend not mention these COTS because they are *business as usual*, and concentrate their attention on COTS that are new for their team and therefore likely to raise problems and risks

Evolving COTS Hypotheses
How are COTS defined?

- Managers tend to mention:
  - Potential problems
    - integrability, possibility of obtaining the desired functionality, dependability of the product and the vendor
  - New process activities required
    - selection, integration, vendor interaction
- Project managers associate COTS with these problems and issues
  - COTS are COTS products *likely to raise risks*.
  - These risks depend on several properties, both of the organization using the COTS, and of the COTS product itself .
- We try to distinguish hard COTS from easy COTS

Evolving COTS Hypotheses
Distinguish between easy and hard COTS



Evolving the Top Ten COTS Hypotheses

Hypothesis 1: *99% of all executing instructions come from COTS products*

The 99% represents predominantly COTS offering the basic layer of services, such as operating system, device drivers, networks. We argue that this basic layer of services is offered by easy COTS Assuming the 80-20 rule applies, we could propose a sub-hypothesis:

Hypothesis 1.1: *80% of all executing instructions come from easy COTS and generate 20% of the problems. 20% of all executing instructions come from hard COTS and generate 80% of the problems.*

Evolving the Top Ten COTS Hypotheses

Hypothesis 6: *Although glue-code development usually accounts for less than half the total CBS software development effort, the effort per line of glue code averages about three times the effort per line of developed-applications code.*

It is reasonable that, due to familiarity with easy COTS, the distribution of effort varies with easy or hard COTS, since easy COTS have less impact on effort. We refine the hypothesis

Hypothesis 6.1: *When easy COTS are involved, glue-code development accounts for the minority of the total CBS development effort.*

Hypothesis 6.2: *When hard COTS are involved, glue-code development accounts for around half the total CBS development effort, and the effort per line of glue code averages about three times the effort per line of developed-applications code.*

Evolving the Top Ten COTS Hypotheses

Hypothesis 8: *CBS assessment and tailoring efforts vary significantly by COTS product classes – operating systems, database management systems, user interface, device driver and so forth.*

Tailoring and assessment efforts depend on how well members of the project are familiar with them, i.e. whether the COTS are easy or hard for them. Based on what we have seen here, we propose a related hypothesis

Hypothesis 8a: *CBS assessment and tailoring efforts vary significantly by COTS class -- easy and hard.*

Evolving the Top Ten COTS Hypotheses

Hypothesis 9: *Personnel capability and experience remain the dominant factors influencing CBS-development productivity*.

Personnel capability and experience with repect to the particular COTS seems to be important most important. Projects suggest a refined sub-hypothesis of Hypothesis 9:

Hypothesis 9.1: *Personnel experience with the COTS products used is dominant factor influencing CBS-development productivity.*

Evolving the Top Ten COTS Hypotheses

Hypothesis 10: *CBS is currently a high-risk activity, with effort and schedule overruns exceeding non-CBS software overruns. Yet many systems have used COTS successfully for cost reduction and early delivery*

In all cases the option of not using COTS would have meant a much longer schedule. Thus, we refine the hypothesis into two sub-hypotheses:

Hypothesis 10.1: *For hard COTS, CBS is a high-risk activity, with effort and schedule overruns exceeding non-CBS software overruns*

Hypothesis 10.2*: For easy COTS, CBS can be used successfully for cost reduction and early delivery*.

CeBASE Empirical Research Activities

Hypothesis Formation

- Large number of possible quality goals and measures
- Requires a set of clear, testable hypotheses to focus empirical work
- Example: Evolving the CeBASE proposed heuristics

CeBASE Empirical Research Activities

Empirical Data and Experiences Collection

- Ultimate source of data is the execution of software projects
- Research process grounded in the observed behavior of software processes in execution, not theoretical claims about properties
- Data can be gathered in a number of ways (e.g. controlled experiments in realistic settings, case studies on industrial projects) and can produce many different types of measures (e.g. effort spent on the process, effectiveness of the process, reliability of the system that results)
- Data and experiences can be stored in “experience bases” so they can be used for future projects
- More than a database, an experience base contains quantitative data, lessons learned or qualitative experiences that are harder to capture and categorize

CeBASE Empirical Research Activities

Recording Influencing Variables

- Data in an experience base requires the appropriate “meta-data” to be carried along to describes the context from which it was collected
- Differences in the results of the technology assessment may be potentially traced back to important differences in the context in which it was applied
- For example, in CeBASE lessons learned characterizes CBD, via a set of context variables - the situation in which each lesson was observed
- Users of the database must be able to understand whether their environment is similar to the one in which the observation was made

CeBASE Empirical Research Activities

Building Models

- From multiple experience bases, lessons learned are analyzed and abstracted into models to provide meaningful information about the results that can be expected from applying the technology
- Abstraction requires multiple projects' data so that (1) there is some confidence the results are not accidental, relating just to the specific circumstances of a particular project; and (2) differences in results can be analyzed across different types of projects, looking for patterns
- Type of model created to represent the knowledge must be appropriate to the type of decision it would support and level of rigor of the underlying data
- Different types of models, at various levels of rigor are next given listed in order of increasing rigor

CeBASE Empirical Research Activities

Building Models

- Individual **lessons learned** describe phenomena regarding a process' use on a project, e.g., a particular process was hard to apply
- **Heuristics/patterns** abstract common features of a technology's use across multiple environments
- **Decision support frameworks** integrate multiple heuristics related to a common area, with guidance as to what development tasks they relate to and under what circumstances they should be applied
- **Quantitative models and tools** give estimates for various facets of project behavior based on a characterization of the project environment and the types of technologies used (COCOTS)

CeBASE Empirical Research Activities

Integrating Models into a Framework

- Since various projects will have different goals and constraints, models need to be packaged with some idea of *when* to apply each of them and and *how to resolve contradictions* among them, for example, when organizational and project goals may conflict
- A decision support framework is needed that allows the integration of multiple models, identifying potential conflicts, and permitting trade-offs of various characteristics, e.g., the ability to trade-off performance for cost, or ease of application for effect

→ We are integrating various COTS based development processes into MBASE

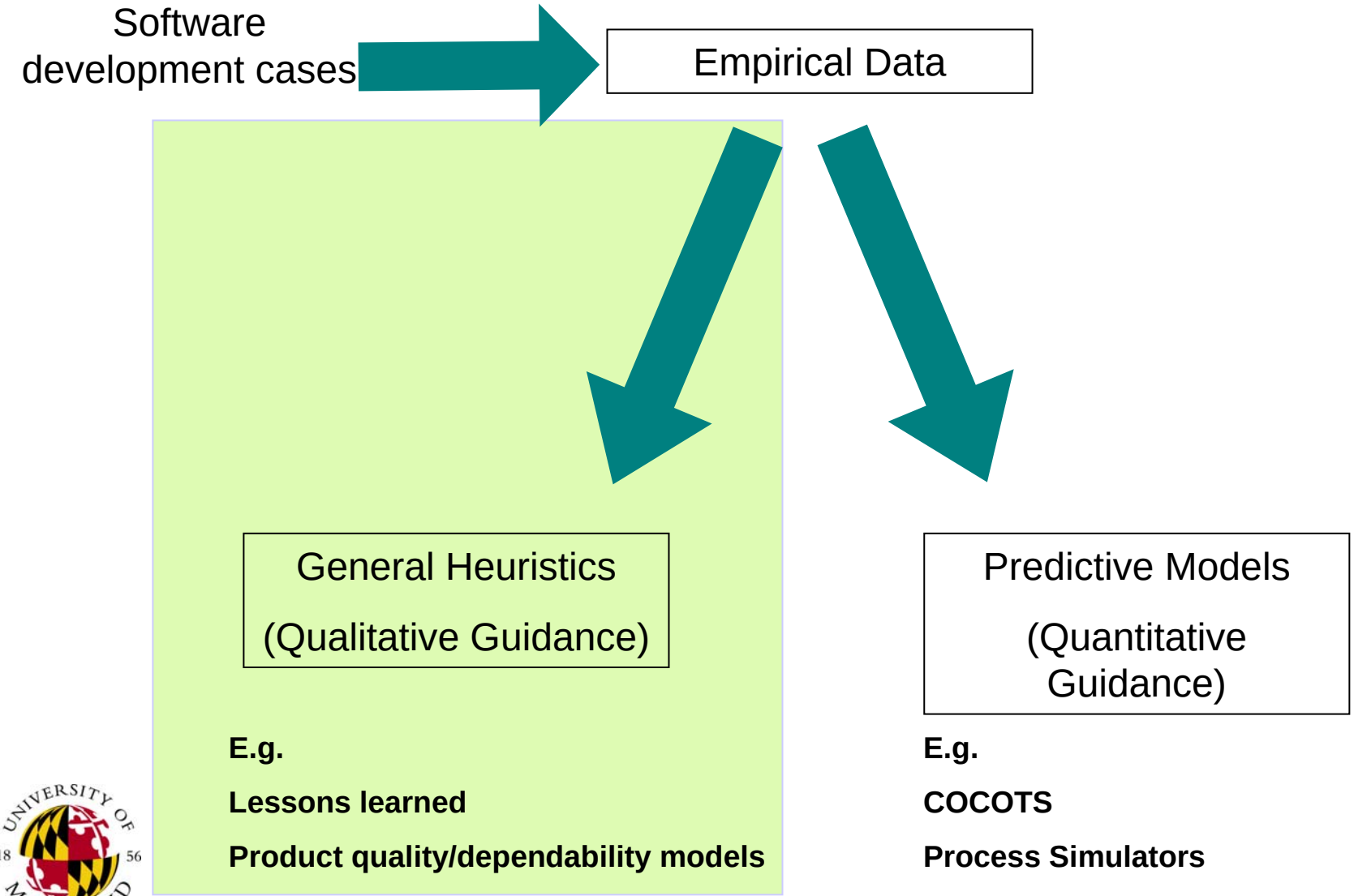
CeBASE Empirical Research Activities

Testing Hypotheses by Application

- Abstraction and model-building results are evaluated, either through off-line controlled experiments, application in new projects, or comparison against additional incoming project experiences to test whether they hold in the new situation
- Results consistent with the models, increase the level of confidence in those models increases; results inconsistent, explanatory environmental variables are sought and experimental methods re-examined

→ We are continually looking for opportunities to test our existing hypotheses

CeBASE EB framework



Conclusion

Empirical Methods can help the

- COTS Process Developer
  - Feedback on how well the process works, enabling its improvement
- COTS-Based Development Projects
  - Experience base of processes packaged with their strengths and weaknesses for achieving project goals, and support for evaluating and applying them in practice
- Software Engineering Community
  - Experience base of assessed, packaged COTS processes
  - A solid basis for empirical software engineering research and development

CeBASE

Contributors to the work in this presentation

Barry Boehm – University of Southern California
Mikael Lindvall - Fraunhofer Center
Maurizio Morisio Politecnico di Torino
Ioana Rus – Fraunhofer Center
Carolyn Seaman – UMBC & Fraunhofer Center
Forrest Shull – Fraunhofer Center

Fraunhofer USA

Center for Experimental Software Engineering, Maryland