Predicting Value from Design

Mary Shaw

with Ashish Arora, Shawn Butler, Vahe Poladian, Chris Scaffidi

Carnegie Mellon University

http://www.cs.cmu.edu/~shaw/

Institute for Software Research, International

Dependability in the real world

- Dependability needs arise from user expectations
  - "Good enough" is often good enough
- Uncertainty is inevitable
  - Specifications won't be complete
  - Knowledge of operating environment is uncertain
- Costs matter, not just capabilities
  - Few clients can afford highest dependability at any cost
- Integration is a greater challenge than components
  - Especially if components come from many sources
  - Especially if system serves multiple objectives

Specifications: Conventional Doctrine

Component specification is

sufficient and complete	--	says everything you need to know or may rely on
static	--	written once and frozen
homogeneous	--	written in single notation

"Three prerequisites must be met for a component to be used in more than one system:

complete, opaque enclosure;
complete specification of its external interface;
and design consistency across all sites of reuse.

Without these, reuse will remain an empty promise."

Real *(Incomplete)* Architectural Specs

- Heterogeneous

Many kinds of information:	functional, structural, extra-functional, family properties
Many types of values:	integer, formula, narrative

- Intrinsically incomplete

Open-ended needs:	cannot anticipate all properties of interest
Cost of information:	impractical, even for common properties

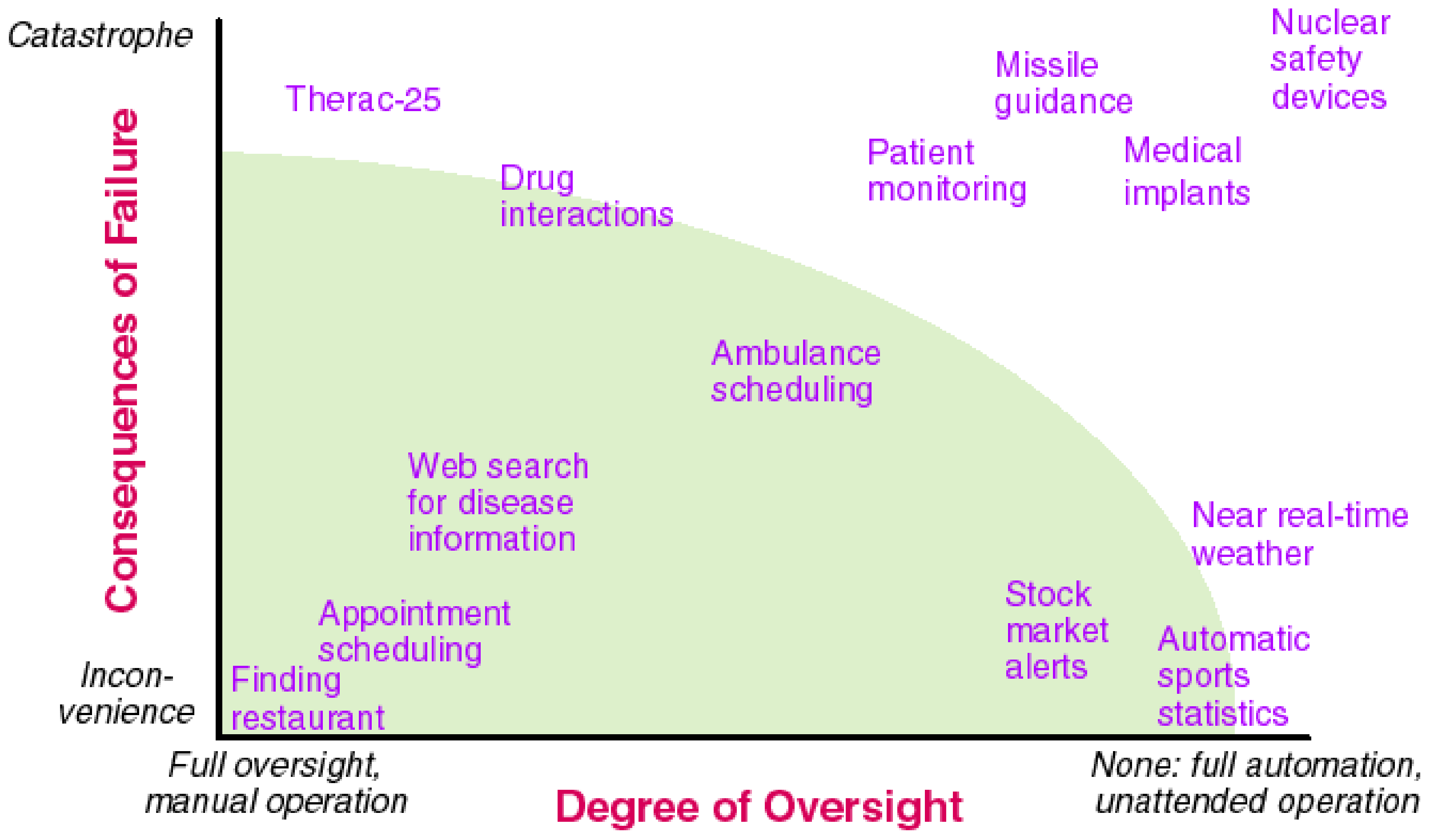
- Evolving

Partial information:	understanding commensurate with amount of information
New properties:	additional properties added as discovered

Sufficient Correctness

- Traditional model
  - Gold standard of program correctness is functional correctness
  - For systems, also need extrafunctional properties such as reliability, security, accuracy, usability
- In practice
  - Most software in everyday use has bugs …
    - … yet we get work done
  - It isn’t practical to get complete specifications
    - Too many properties people can depend on
    - Variable confidence in what we do know
  - We don’t really need “correctness”, but rather assurance that the software is good enough for its intended use

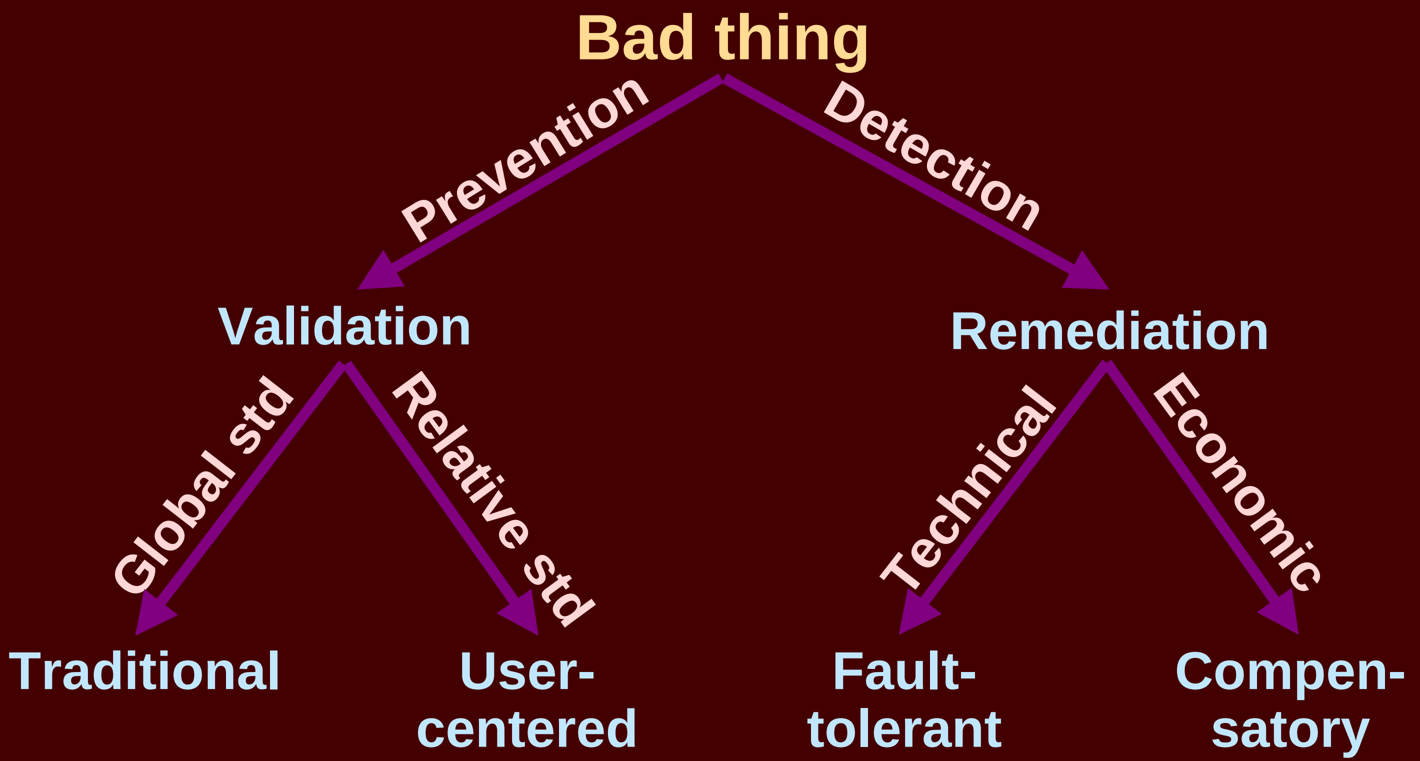
How much must you trust your software?



Value-based approach to dependability

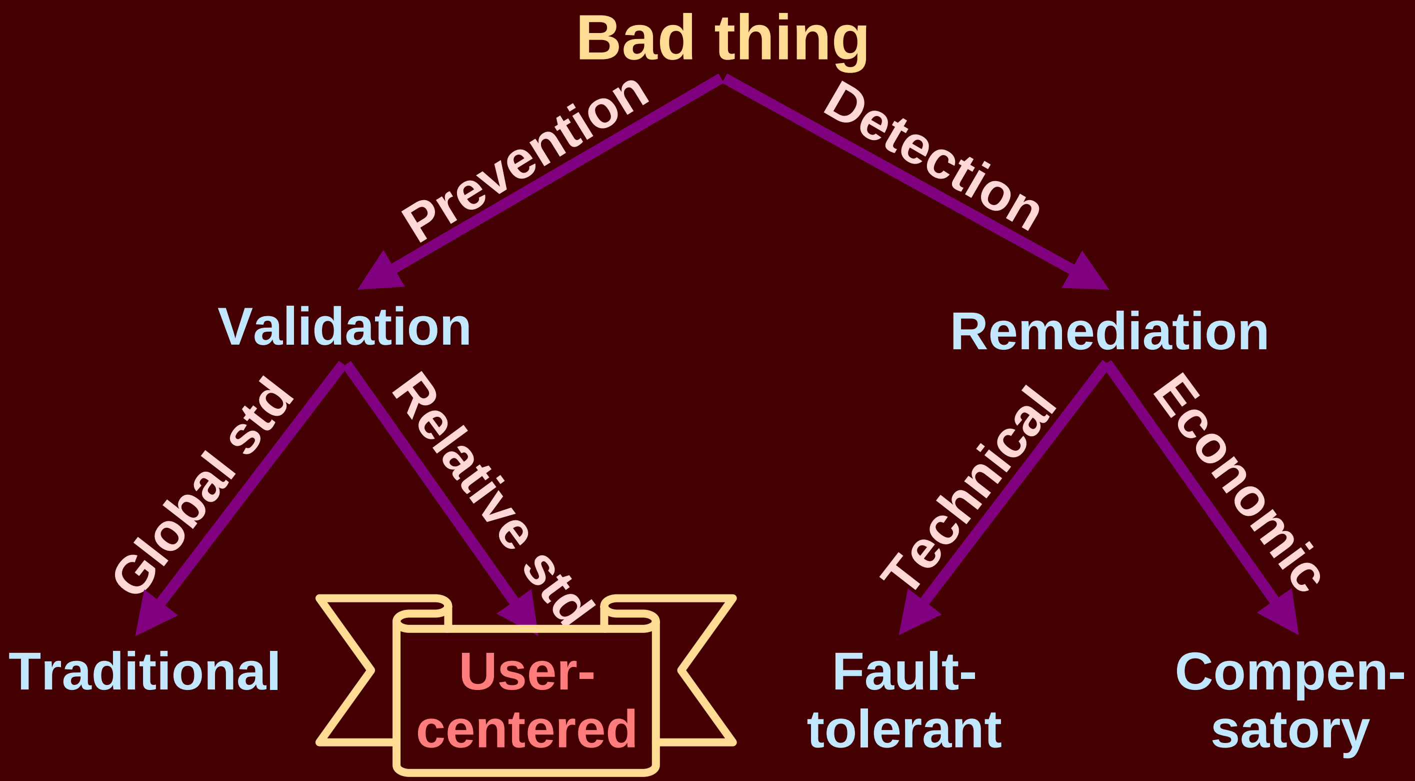
- Value is benefit net of cost
  - Value to a given client is benefit net of cost to ***that*** client
- Dependability involves uncertainty
  - … about properties of software components
  - … about interactions of components or systems
  - … about operating environment
  - … about consequences of failure
- Value of dependability involves prediction and risk management
- Benefits and costs are largely set early in design
  - But at that time benefits and costs can only be predicted

Ways to deal with undependability



- Traditional: prevent through careful development, analysis
- User centered: set criteria for proper operation to reflect user needs
- Fault tolerant: repair failures as they occur
- Compensatory: provide financial compensation

Ways to deal with failure



- Traditional: prevent through careful development, analysis
- User centered: set criteria for proper operation to reflect user needs
- Fault tolerant: repair failures as they occur
- Compensatory: provide financial compensation

Context-Sensitive Requirements

- Different users have …
  - …different tolerance for system error and failure
  - …different interests in results from a resource
  - …different tolerance and interests at different times
- Criteria for proper operation should reflect these differences
  - Requirements can't be tied solely to resource
  - Users need ways to express differences
- Need ***user-centered requirements*** as part of architectural design and resource integration

Security technology *portfolio selection*

- Different sites have different security issues
- Elicit concerns about threats and relative priorities with multi-attribute decision techniques
  - converts subjective comparisons to quantitative values
- Associate threat analysis with cost of successful attack and countermeasures available in the market
  - Consider cost-effectiveness and defense in depth
- Iterate, using sensitivity analysis and multiattribute techniques to refine recommendations
  - Get better understanding as well as recommendation
- Shawn Butler (CMU '03)

Anomaly Detection

- If you have specifications, you can detect violations
- Most everyday software does not have good specs
- Problem: how to discover "normal" behavior and capture this as predicates
  - Infer predicates from resource's history
    - Multiple statistical, data mining techniques
  - *Set-up:* elicit user expectations while tuning predicates
    - Using templates that show what techniques can express
  - *Operation:* apply inferred predicates to detect anomalies
- Inferred predicates serve as proxies for specs
- "Anomaly" is in the eye of the specific user
- Orna Raz (CMU '04)

Utility-based Adaptive Configuration

- Mobile systems are resource-limited
  - Processor power, bandwidth, battery life, storage capacity, media fidelity, user distraction, …
- Users require different capabilities at different times
  - Editing, email, viewing movies, mapping, …
  - Dynamic preferences for quantity and quality of service
- Abstract capabilities can be provided by different combinations of services
  - Specific editors, browsers, mailers, players, …
- Use utility theory and linear/integer programming to find best series of configurations of services
- Vahe Poladian (5th year PhD student)

Idea: *Multidimensional cost analysis*

- Types of cost
  - Dollars, computer resources, user distraction, staff time, reputation, schedule, lives lost
- Naïve view
  - Convert all costs to a single scale, e.g., dollars
- Problem
  - Cost dimensions have different properties
- Resolution
  - Carry cost vector as far into analysis as possible
  - Convert to single scale at the latest point possible

We need better ways to analyze a software design and predict the value its implementation will offer to a customer or to its producer

Engineering design

- Engineers . . .
  - iterate through design alternatives
  - reconcile client's constraints
  - consider cost & utility as well as capability
  - recognize that early decisions affect later costs

. . . but . . .

Engineering design

- Engineers . . .
  - iterate through design alternatives
  - reconcile client's constraints
  - consider cost & utility as well as capability
  - recognize that early decisions affect later costs

. . . but . . .

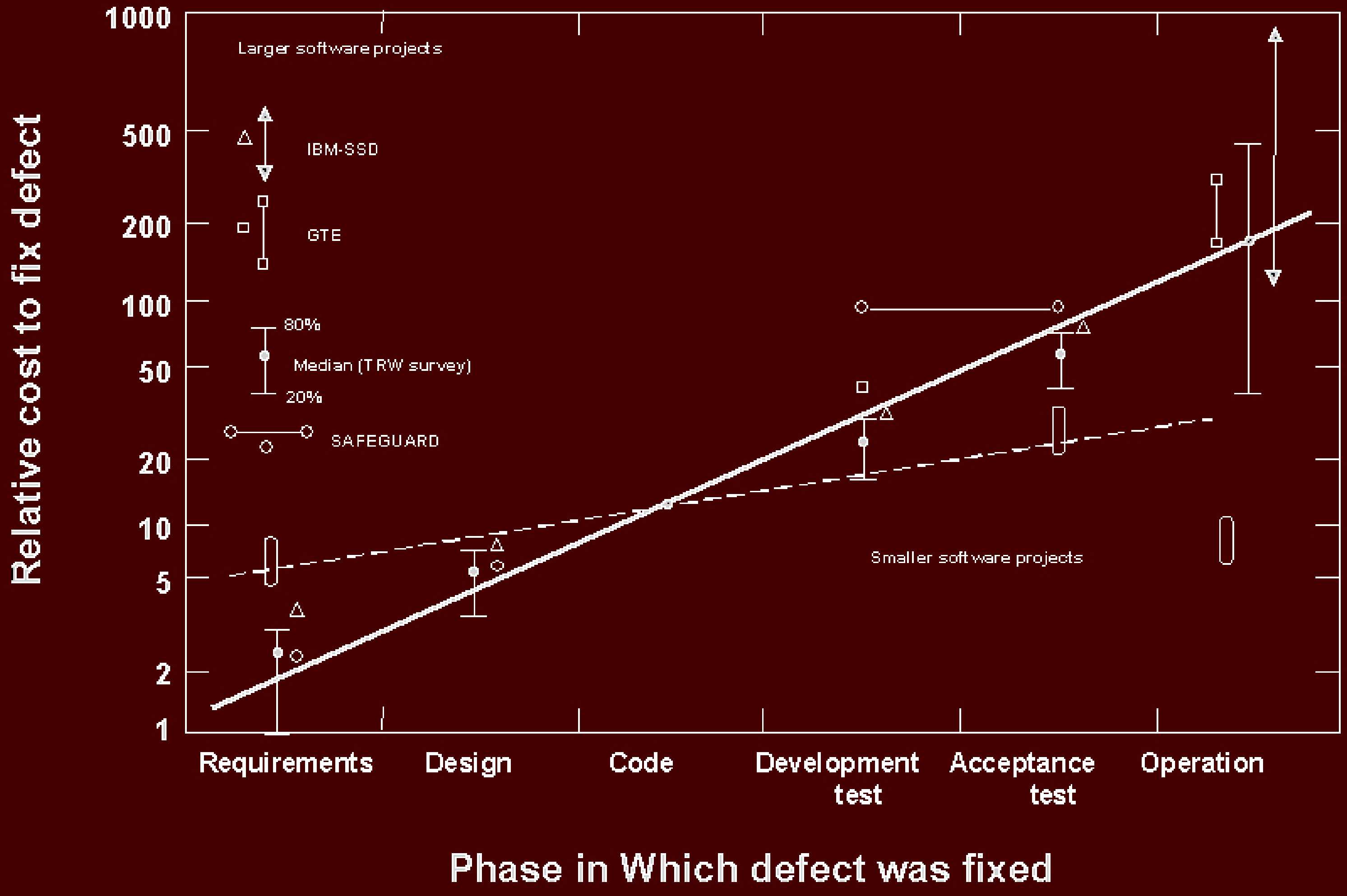
- Software engineers . . .
  - lack adequate techniques for early analysis of design
  - design for component spec rather than client expectation
  - rarely include cost as 1st-class design consideration

Why does early design evaluation matter?

- Cost of repair
  - Fixing problems after delivery often costs 100x more than fixing them in requirements and design
  - Up to half of effort goes to avoidable rework
    - "avoidable rework" is effort spent fixing problems that could have been avoided or fixed earlier with less effort
  - Early reviews can catch most of the errors

-- Boehm/Basili, IEEE Computer, 2001

Cost of delaying risk management



-- Barry Boehm

Why does early design evaluation matter?

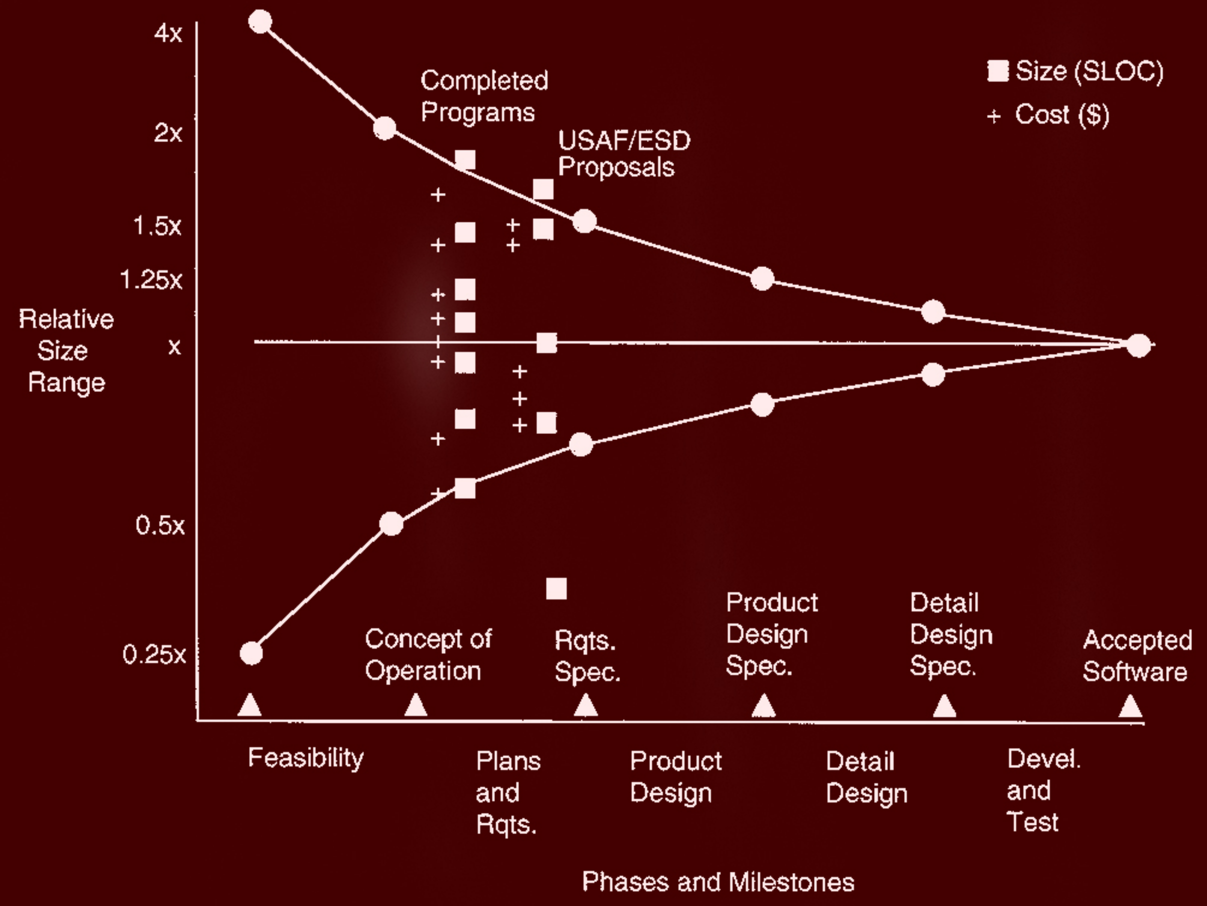
- Cost of repair
  - Fixing problems after delivery often costs 100x more than fixing them in requirements and design
  - Up to half of effort goes to avoidable rework
    - "avoidable rework" is effort spent fixing problems that could have been avoided or fixed earlier with less effort
  - Early reviews can catch most of the errors

. . . but . . .

- Confidence in estimates is lowest early in a project

-- Boehm/Basili, IEEE Computer, 2001

Confidence in estimates



Software costing and sizing accuracy vs phase

-- Boehm, COCOMO II, 2000

Why does early design evaluation matter?

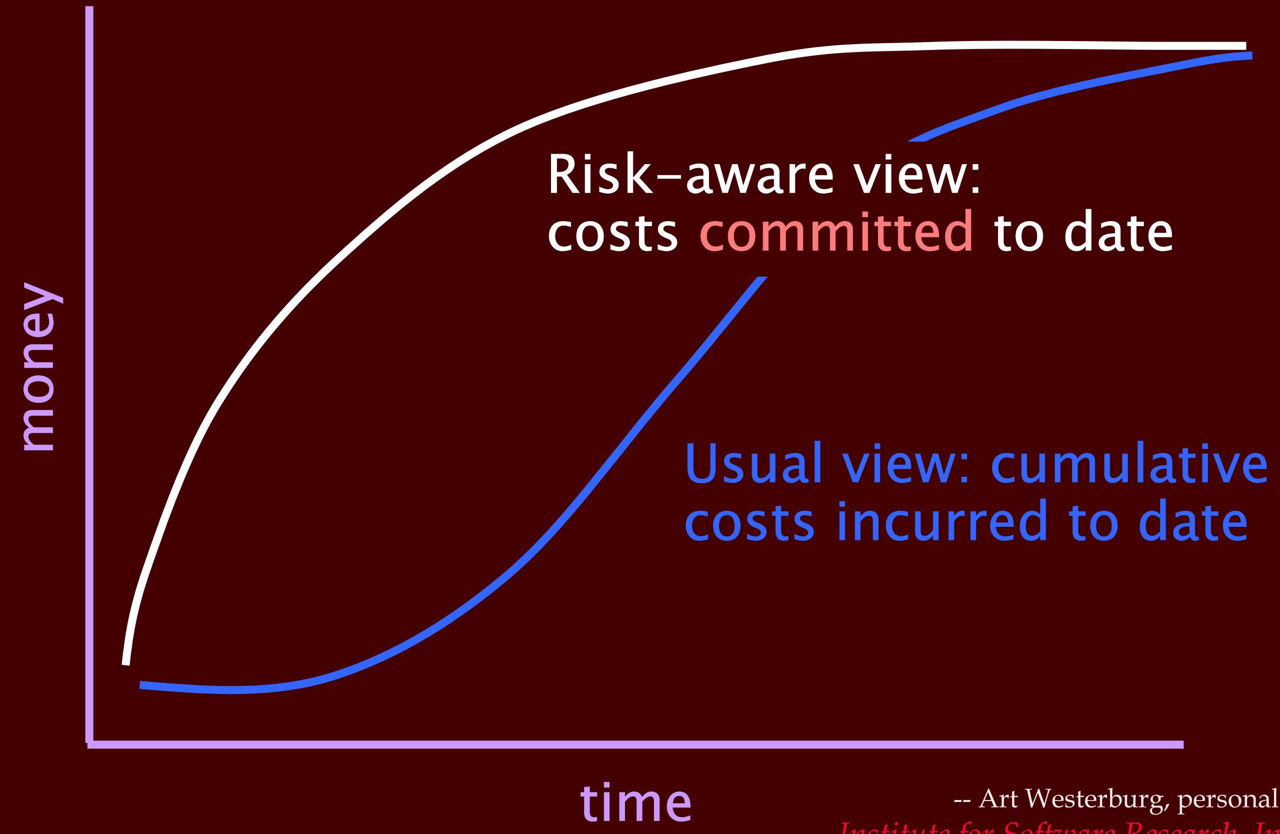
- Cost of repair
  - Fixing problems after delivery often costs 100x more than fixing them in requirements and design
  - Up to half of effort goes to avoidable rework
    - "avoidable rework" is effort spent fixing problems that could have been avoided or fixed earlier with less effort
  - Early reviews can catch most of the errors

. . . but . . .

- Confidence in estimates is lowest early in a project
- Early decisions commit most of the resources

Costs, commitment, and uncertainty

- Engineering involves deciding how to make irreversible commitments in the face of uncertainty



-- Art Westerburg, personal communication

Current software design evaluation

- Relatively little attention to early design evaluation
- Software-centric evaluations
- Minor role for costs other than development
- Sparse, scattered, inconsistent evaluation methods

Current software design evaluation

- Relatively little attention to early design evaluation
  - Leverage lower cost of change during design
- Software-centric evaluations
  - Consider user-specific preferences, or perceived value
- Minor role for costs other than development
  - Expand role for larger-scale economic issues
- Sparse, scattered, inconsistent evaluation methods
  - Find ways to use models together

What needs to be done?

- Make early predictive design evaluation viable
  - Identify existing techniques that apply early
  - Explain them in a consistent way
  - Determine how to compose them
  - Develop new techniques
- Provide a unifying model
  - Be explicit about interfaces
  - Be clear about method and confidence
- Support it with tools

Early, code-free, design evaluation

- Target of evaluation
  - very high level design, before "software design" methods start elaborating the box and line diagrams
  - evaluation that weighs costs as well as capabilities
  - evaluation that recognizes user needs and preferences
  - evaluation that does not depend on ~~access to code~~
- Long-term objective: framework to unify models
  - general, to handle models for various specific attributes
  - open-ended, esp. with respect to the aspects considered
  - flexible, handling various levels of detail and precision

Economists' view of value

- A firm's goal is typically to maximize total revenue minus cost of the inputs, represented by

$$\max [(B(z) - C(y))] \text{ such that } F(y,z) \leq 0$$

- Here
  - In vector z, z_j represents quantity of product j sold
  - B(z) is the total revenue from selling those products
  - In vector y, y_i represents quantity of input i consumed
  - C(y) is the total cost of those inputs
  - F(y, z) is a vector, as well, so $F(y, z) \leq 0$ represents a list of equations representing constraints on the problem

Model for predictive analysis of design

$U(d, \theta) = B(x,\theta) - C(d,x,m)$ for $\{ x : F(d,x,m) \}$, where $x = P(d,m)$

Value U of design d to a client with preferences θ is benefit B net of cost C provided the desired result x is achievable and attributes x of implementation are predicted by P

Let	d be a design	in some appropriate notation
	x be in A^n	an open-ended vector of capabilities
	v be in V^n	a multidimensional value space
	m be in some notation	a development method
	θ express user pref	a multidimensional utility space
	B express benefits	predicted value v of x to user with pref θ
	C express costs	cost v of getting x from d with method m
	F checks feasibility	whether d with x can be achieved with m
	P predicts capabilities	attributes x that m will deliver for d

Basic value proposition

$U = B - C$

Following economics, value is benefit net of cost

Adopting a software tool will cost \$X, and it will save you \$Y, right away, on your current project.

$$U = \$Y - \$X$$

Value based on product attributes

$$U(\boxed{d}) = B(\boxed{x}) - C(\boxed{x})$$

The value of a design is the benefit, net of cost, of the implementation as represented by its capabilities.

Let	***d*** be a design	in some appropriate notation
	x be in R^n	an open-ended vector of capabilities
	v be in R	value in dollars
	B express benefits	predicted value ***v*** of ***x*** to user
	C express costs	cost ***v*** of getting or using ***x***

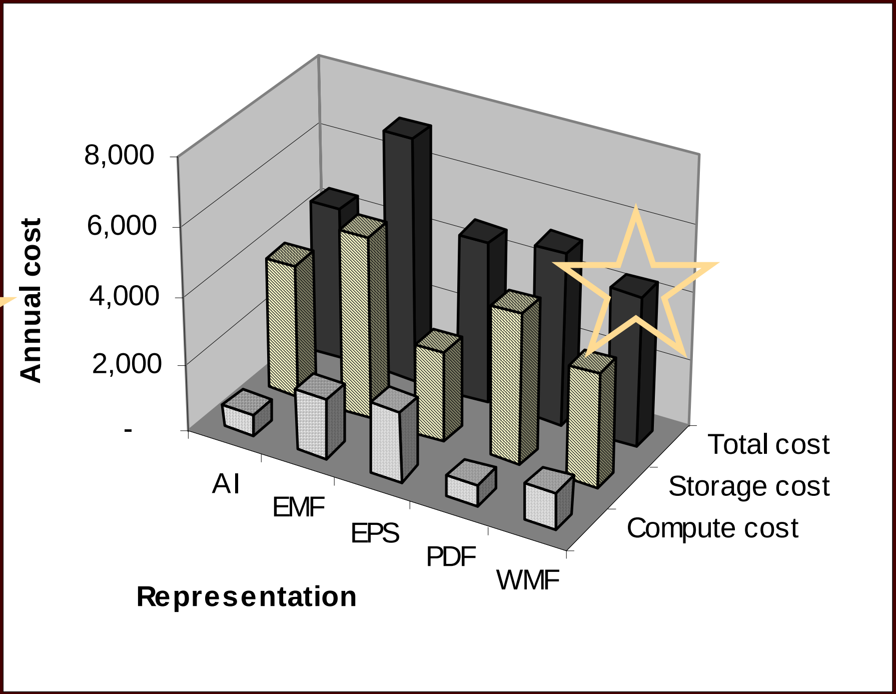
Ex 2: Choosing a representation

- You store maps to view and edit in drawing package
- Only 1 of every 50 reads leads to a write
- Cost: $10K per sec read/write, $0.1/KB storage
- You get data for your typical data sets:

File type	Seconds to open (read)	Seconds to write (save or export)	File size (KB)
AI	6	93	6243
EMF	9	88	17908
EPS	5	17	20909
PDF	7	95	6243
WMF	5	86	11038

Best representation *for this application*

Design *d* <format>	Attributes *x* <time,size>	Cost *v* <total $>
AI	<393,6243>	$4554
EMF	<538, 17908>	17908
EPS	<267, 20909>	4761
PDF	<445, 6243>	5074
WMF	<336, 11038>	4464



Ex 3: Determining value of features

- For spreadsheets,
  - Adherence to dominant standard ➔ 46% higher price
  - 1% increase in installed base ➔ 0.75% increase in price
  - Quality-adjusted prices over 5 years declined 16%/year
- Hedonic model a good predictor
  - Hedonic model estimates value of product aspects to consumer's utility or pleasure; it assumes price is a function of product features

Econometric analysis of spreadsheet market, 1987-92

--Brynjolfsson/Kemerer, Network Externalities in Microcomputer Software, Mgt Sci, 1996

Predicting attributes from design

$U(d) = B(x) - C(x)$ $\boxed{\text{where } x = P(d)}$

We often need to predict the implementation properties *x* before the code is written

Let	*d* be a design	in some appropriate notation
	x be in R^n	an open-ended vector of capabilities
	v be in R	value in dollars
	B express benefits	predicted value *v* of *x* to user
	C express costs	cost *v* of getting *x* from *d*
	P predicts capability	capabilities *x* of implementation of *d*

Ex 4: Predicting size from function points

COCOMO Early Design

- Examine design to count function points

	Complexity Levels		
Type	Low	Average	High
Internal logical files	7	10	15
External interface files	5	7	10
. . . etc . . .	. . .	. . .	. . .

- Choose programming language
- Use pre-calibrated table to estimate code size

Language	Ada 95	C++	Java	PERL	VB 5.0
LOC per Fcn Pt	49	55	53	27	34

-- Boehm, COCOMO II, 2000

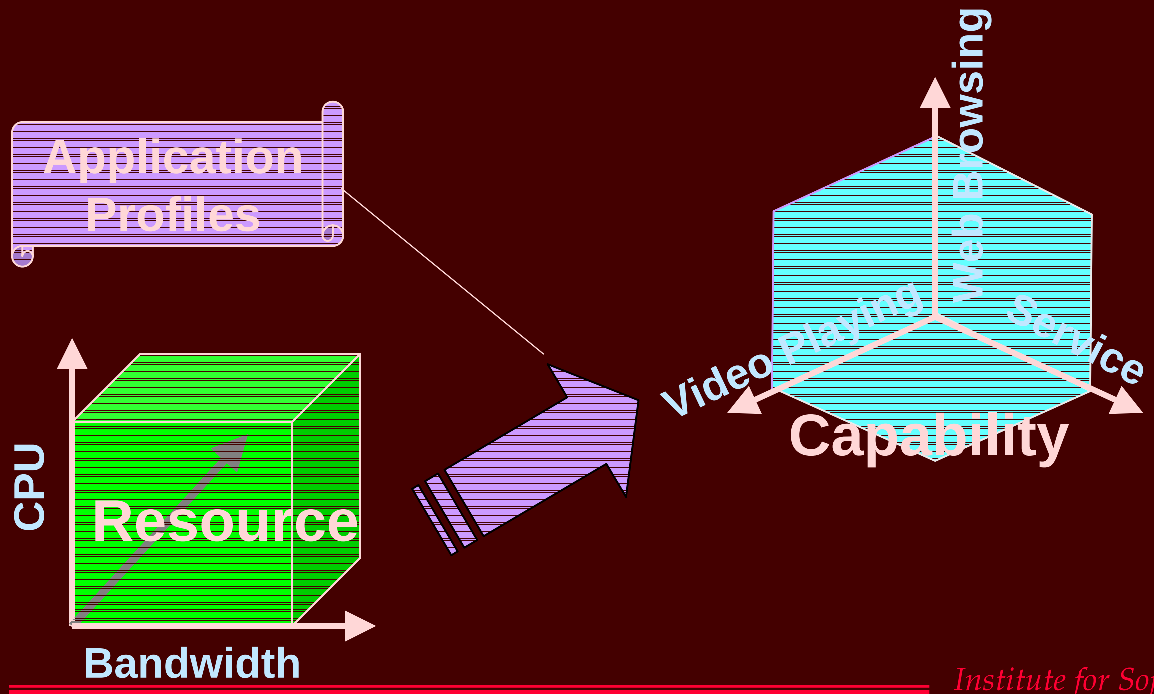
Ex 5: Predicting mobile performance

Given a configuration of applications to support a user task, what will its resource requirements be?

Design ***d*** is "configuration" expressed as
{<application, (QoS settings>}

```
{ <Windows Media Player,
          (24 fps, 300x200, high quality audio) >
  <MS Word,
          ( ) >,
  <Firefox,
          (5 s, text) >
}
```

Resource use of configuration



Ex 5: Predicting mobile performance

Empirical profiling yields resource usage

Implementation attributes maintain distinctions among resource consumers:

{<application, (QoS settings), resource usage>}

```
{ <Windows Media Player,
        (24 fps, 300x200, high quality audio),
        (25%, 256 Kpbs, 30 MB)>,
  <MS Word,
        ( ),
        (2%, 0 Kpbs, 28 MB>,
  <Firefox,
        (5 s, text),
        (8%, 56 Kpbs, 10 MB)>
}
```

Time $\neq$ Money

$U(d\quad) = B(x\quad) - C(d,x,m)$ where $x = P(d\quad)$

Capabilities x and values v are multidimensional; they may be measured on different scales

Let	d be a design	in some appropriate notation
	x be in A^n	open-ended vector of arbitrary attributes
	v be in V^n	open-ended vector of arbitrary attributes

B express benefits	predicted value v of x to user
C express costs	cost v of getting x from d with method m
P predicts capability	capabilities x that m will deliver for d

Multidimensional Cost Analysis

- Different factors in a problem are appropriately measured in different ways
  - Dollars, computer resources, user distraction, staff time, reputation, schedule, lives lost
- It's tempting to convert everything to dollars, but this can lead to …
  - Loss of information related to different properties
  - Errors by converting nominal, ordinal, or interval scales to a ratio scale
  - Loss of flexibility by early choice of conversion
  - Confusion of precision with accuracy
- Many analysis techniques require a single cost unit, but you should delay conversion as long as possible

Properties of Resources

- **Perishable**: lost if not used

Perishable	bandwidth
Nonperishable	disk space

- **Fungible**: convertible to other resources

Complete	common currency
Partial	bandwidth vs CPU (compression)
None	calendar time vs staff months

- **Rival**: use by one person precludes use by another

Rival	money, labor, bandwidth
Nonrival	information goods

- **Measurement scale**: appropriate scale & operations
  - Nominal, ordinal, interval, ratio

Ex 6: Algorithmic Complexity

- Analysis of algorithms tells you how running time will scale with problem size
  - A sort algorithm might be O(n log n)
  - Scalability is not a scalar attribute!!
- In this case
  - ***d***, the design, is the pseudo-code of the sort algorithm
  - ***x***, the capabilities, is O(n log n) scalability
  - ***v***, the value space, includes a scalability dimension
  - ***m***, the development method, is a programming technique
  - ***P*** predicts competent implementation ➔ expected runtime
  - ***C*** is the cost (e.g., performance) of O(n log n) execution time

Considering development method

$U(d) = B(x) - C(x)$ where $x = P(d,m)$

We don't have the code during early design, so we have to predict the implementation properties ***x*** assuming ***d*** is implemented by method ***m***

Let	***d*** be a design	in some appropriate notation
	x be in R^n	an open-ended vector of capabilities
	v be in V^n	a multidimensional value space
	m be in some notation	a development method
	B express benefits	predicted value ***v*** of ***x*** to user
	C express costs	cost ***v*** of getting ***x*** from ***d*** with method ***m***
	P predicts capability	capabilities ***x*** that ***m*** will deliver for ***d***

Ex 6a: Algorithmic Complexity, again

- Analysis of algorithms tells you how running time will scale with problem size
  - A sort algorithm might be O(n log n)
- In this case
  - ***d,*** the design, is the pseudo-code of the sort algorithm
  - ***x,*** the capabilities, is O(n log n) scalability
  - ***v,*** the value space, includes a scalability dimension
  - ***m,*** the development method, is a programming technique
  - ***P*** predicts competent implementation ➔ expected runtime
  - ***C*** is the cost (e.g., performance) of O(n log n) execution time
- Implementation must be competent, not just correct
  - I once saw an O(n^3) implementation in a class assignment!

Ex 7: COCOMO II Early Design Model

- COCOMO predicts effort (PM) & schedule (TDEV)
  - $PM = A (Size)^E \Pi_i EM_i$ where $E = B + 0.01\Sigma_j SF_j$
    - A, B are calibrated to 161 projects in the database
    - EM_i and SF_j characterize project and developers
    - TDEV is similar
- But it depends on Size, and LOC aren't known early
  - Count unadjusted function points (UFP) in requirements
  - Use COCOMO II's conversion table (previous example!!)
  - Size = KSLOC(programming language, UFP)

Ex 7: Predicting development effort

$C(d,x,m)$

$= C(\text{Size}, x, \langle A, B, Em_j, SF_k \rangle) = \langle PM \rangle$

$= \langle A \times \text{Size}^E \prod_i EM_i, \rangle$ where $E = B + 0.01\Sigma_j SF_j$

$= \langle A \times \text{KSLOC}(pl, \text{UFP}(d))^E \prod_i EM_i \rangle$

With nominal values for A, B, SF_j, EM_j

$= \langle 2.94 \times \text{KSLOC}(pl, \text{UFP}(d))^{1.0997} \rangle$

For 100KSLOC system,

$= \langle 465.3153 \text{ person-months} \rangle$

Client-focused Value

$U(d,\boxed{\theta}) = B(x,\boxed{\theta}) - C(d,x,m)$ where $x = P(d,m)$

Most significantly, value can only be reckoned relative to the needs and preferences (utilities) of a stakeholder – in this case, the client or user

Let	d be a design	in some appropriate notation
	x be in R^n	an open-ended vector of capabilities
	v be in V^n	a multidimensional value space
	m be in some notation	a development method
	θ express user pref	a multidimensional utility space
	B express benefits	predicted value v of x to user with pref θ
	C express costs	cost v of getting x from d with method m
	P predicts capability	capabilities x that m will deliver for d

Ex 8: Mobile configuration utility

$U(d, \theta) = B(x,\theta) - C(d,x,m)$ where $x = P(d,m)$

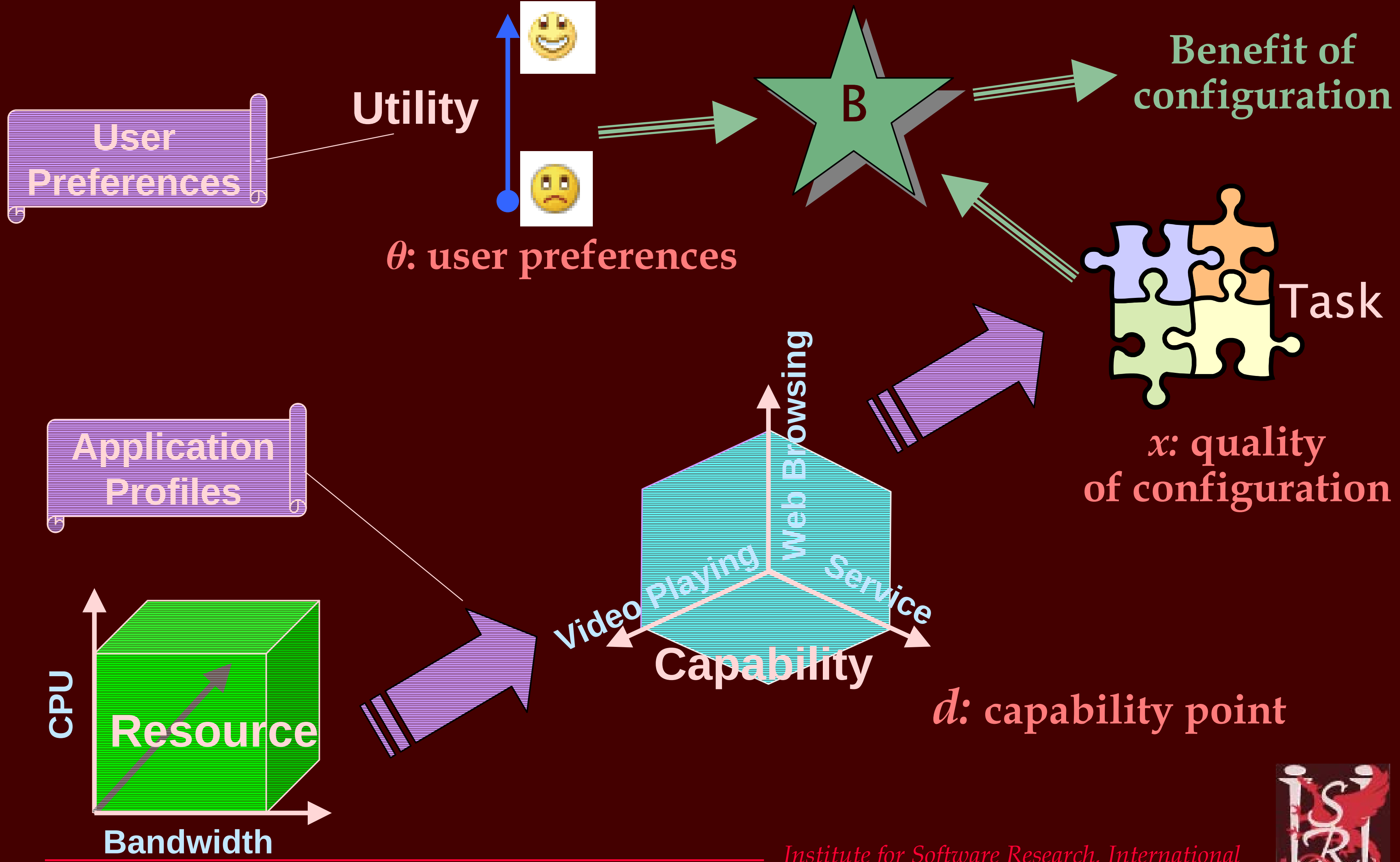
We previously saw prediction of x from d

- x is qualities of delivered service (e.g. video fidelity)
- d is application configuration (player + editor)
- v is <user utility, seq of configurations, resource use>
- Objective is a sequence of configurations d with the that best satisfies each user's personal preferences θ

Video player	Windows media	1.0
	RealPlayer	0.8
Frame rate	10 fps	0.1
	18 fps	0.5
	24 fps	1.0

. . . etc . . .

Ex 8: Mobile configuration utility



Institute for Software Research, International

Ex 8: Mobile configuration utility

For the configuration design point

{ <Windows Media Player,
(24 fps, 300x200, high quality audio),
(25%, 256 Kpbs, 30 MB)>,

… etc …}

The utility is weighted by attribute

<player, frame rate, frame size, audio> ~~ <.5, 1.0, .5, 1.0>

Then the player component of the utility is

$.5 * \theta(\text{Media Player}) + 1.0 * \theta(\text{24 fps}) + .5 * \theta(\text{300x200}) + 1.0 * \theta(\text{high})$

$= .5 + 1.0 + .5 + 1.0$

$= 3.0$

Uncertainty in values

$U(d, \theta) = B(x,\theta) - C(d,x,m)$ where $x = P(d,m)$

Capabilities x and values **of B, C** may be contingent and uncertain, so the value space may express uncertainty such as ranges, probabilities, future values

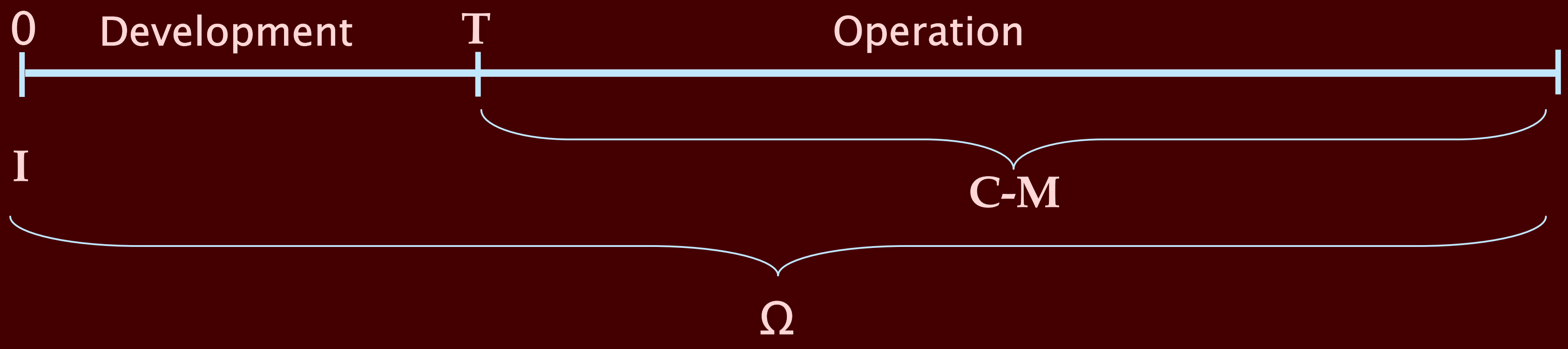
Let d be a design	in some appropriate notation
x be in R^n	an open-ended vector of capabilities
v be in V^n	a multidimensional value space
m be in some notation	a development method
θ express user pref	a multidimensional utility space
B express benefits	predicted value v of x to user with pref θ
C express costs	cost v of getting x from d with method m
P predicts capability	capabilities x that m will deliver for d

Ex 9: Present Value Analysis

- Purchase or license a component?
  - Benefit $60K/year, realized at end of year
  - License cost $50K/year, due at beginning of year
  - Purchase cost $120K, at beginning
  - Interest rate 5%/year

		0.05	interest rate									
					<<<<< Present Values >>>>			**<<<< cumulative values >>>**			**<<Val=(ben-cost)>>**	
End yr	Purchase	License	Benefit	1/(1+I)^N	Purchase	License	Benefit	Purchase	License	Benefit	Val | purc	Val | lic
0	120	50		1.00	120.00	50.00	-	120.00	50.00	-		
1		50	60	0.95	-	47.62	57.14	120.00	97.62	57.14	(62.86)	7.14
2		50	60	0.91	-	45.35	54.42	120.00	142.97	111.56	(8.44)	13.95
3		50	60	0.86	-	43.19	51.83	120.00	186.16	163.39	43.39	20.42
4			60	0.82	-	-	49.36	120.00	186.16	212.76	92.76	26.59
sum	120	200	240		120.00	186.16	212.76					

Economic Value in a SW Project



- Note the times at which variables are evaluated
  - Development cost (I) is PV at time 0 of development cost
  - Asset value (C) and Operation cost (M) are PV at time T
- Risk (d) is used as discount rate to move C&M to 0
- Flexibility value (Ω) measures value of strategic flexibility

$$NPV = (C-M)/(1+d)^T - I + \Omega$$

--Erdogmus, Comparative evaluation of development strategies with NPV, EDSER-1, 1999

Usage scenarios

- Evaluating a given design, comparing products
  - Most of the previous examples explore this scenario
- Composing evaluation functions
  - COCOMO Early Design composes code size estimate with the effort and schedule estimators
- Optimizing among design alternatives
  - We show dynamic reconfiguration for mobile devices
- Deciding what design information to capture
  - Look at the design representations used the the predictors that may be appropriate
- Exploring tradeoff spaces
  - We now show how to use COCOMO in this way

Ex 10: Tradeoffs in development costs

- Most of EM_i and SF_j describe development method, but four describe characteristics of the product
  - SCHED (required development schedule constraint)
  - RCPX (required reliability and complexity)
  - RUSE (required reusability)
  - PDIF (platform difficulty)
- We can restate the Early Design estimators to retain these as parameters
  - For simplicity, use only RCPX, SCHED

COCOMO II, Product Factors Isolated

$U(d\quad) = \qquad - C(d,x,m)$ where $x = P(d,m)$

- x = <RCPX, SCHED>, x_i in {XL,VL,L,N,H,VH,XH}
- d is Size = KSLOC(prog lang, UFP(rqts))
- v is value space <PM,TDEV,RCPX, SCHED>
- m is encoded in the adaptive factors

  $<A, B, Em_{j \text{ not RCPX, SCHED}}, SF_k>$
- COCOMO (P) then predicts the cost element of v

  $PM = A\ (\text{Size})^E\ \Pi_{i \text{ not RCPX, SCHED}}\ EM_i \times EM_{RCPX} \times EM_{SCHED}$

  where $E = B + 0.01\Sigma_j\ SF_j$

Cost of Achieving Given RCPX, SCHED

$C(d,x,m)$

$= C(d, \langle \text{RCPX, SCHED} \rangle, \langle A, B, Em_j, SF_k \rangle)$

$= \langle \text{PM,TDEV,RCPX, SCHED} \rangle$

$= \langle A \times \text{Size}^E \, \Pi_{i \text{ not RCPX, SCHED}} \, EM_i \times EM_{RCPX} \times EM_{SCHED},$
$\text{TDEV,RCPX, SCHED} \rangle$

where $E = B + 0.01\Sigma_j \, SF_j$

$= \langle A \times \text{KSLOC(pl, UFP(}d\text{))}^E \, \Pi_{i \text{ not RCPX, SCHED}} \, EM_i \times$
$EM_{RCPX} \times EM_{SCHED}, \text{TDEV, RCPX, SCHED} \rangle$

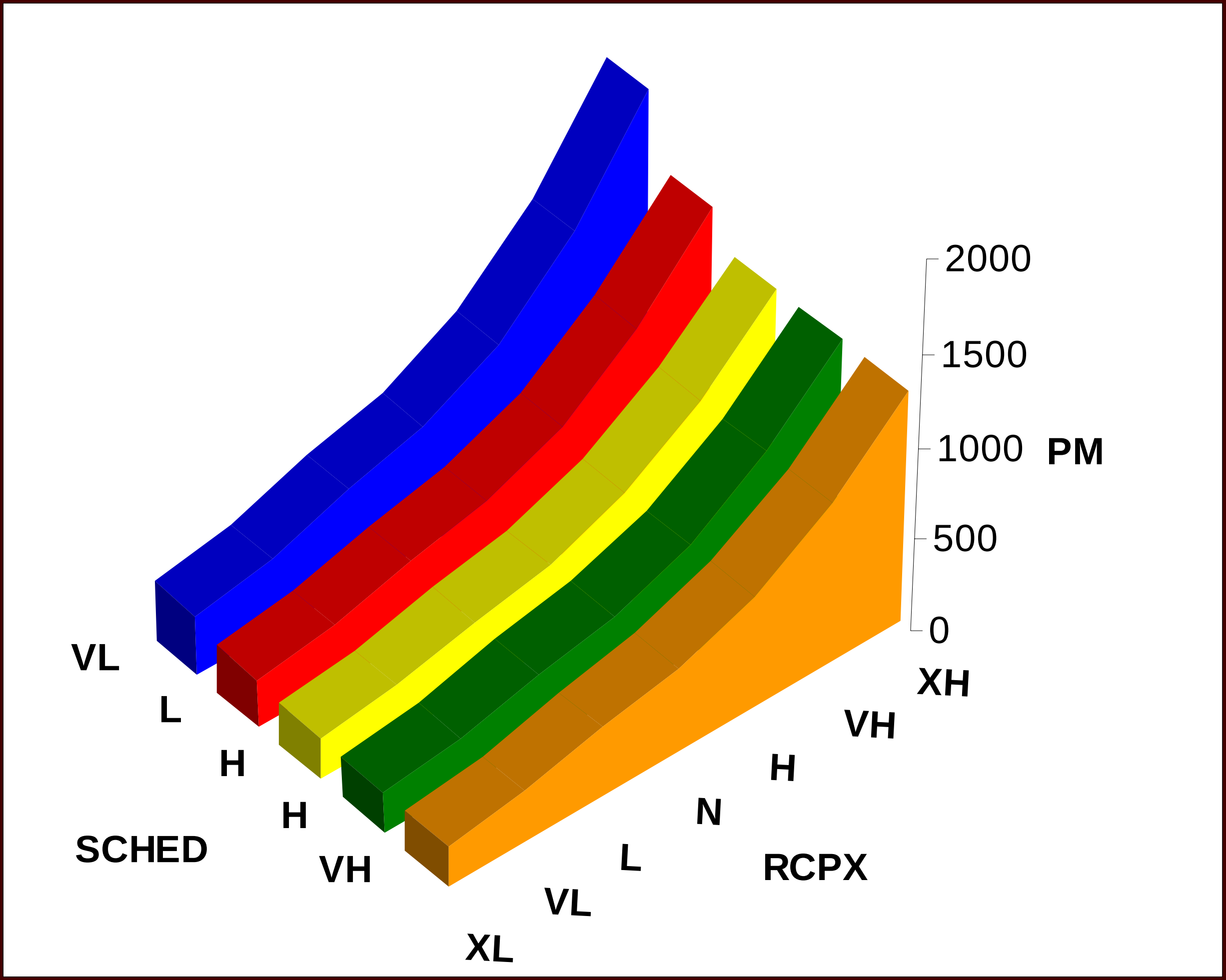
With nominal values for A, B, SF_j, all EM_j but RCPX, SCHED

$= \langle 2.94 \times \text{KSLOC(pl, UFP(}d\text{))}^{1.0997} \times EM_{RCPX} \times EM_{SCHED},$
$\text{TDEV,RCPX, SCHED} \rangle$

For 100KSLOC system,

$= \langle 465.3153 \times EM_{RCPX} \times EM_{SCHED}, \text{TDEV,RCPX, SCHED} \rangle$

Effort to Achieve Given RCPX, SCHED



Review: Examples

- Toy examples
  - 1. Value as simple benefit minus cost
  - 2. Selection of representation for a task
  - 9. Present value analysis for buy vs license decision
- Real models
  - 3. Feature value from econometric analysis of spreadsheets
  - 6. Performance prediction based on algorithmic complexity
  - 7. Schedule and effort from COCOMO II
    - 4. KSLOC prediction from requirements via function points
  - 10. RCPX & SCHED tradeoffs from COCOMO II
- Current and recent research
  - Multidimensional costs
  - 5, 8. User-oriented configuration of mobile devices

Other examples

- Security Attribute Evaluation Method (SAEM, Butler)
  - Elicit client's threat, asset protection priorities (θ)
  - Evaluate per-threat countermeasure effectiveness ($\boldsymbol{x} = \boldsymbol{P}(\boldsymbol{d},\boldsymbol{m})$) of candidate security technology to add ($\boldsymbol{d}$)
  - Weight countermeasures by priorities ($\boldsymbol{B}(\boldsymbol{x},\theta)$)
- Cognitive modeling for UIs (Keystroke, GOMS)
  - Design UI and select common tasks
  - Use cognitive model to predict task times ($\boldsymbol{x} = \boldsymbol{P}(\boldsymbol{d},\boldsymbol{m})$)
- Real options to evaluate delayed decision
  - Additional cost now to preserve flexibility
  - Cost to exercise flexibility later
    - $\boldsymbol{C}(\boldsymbol{d},\boldsymbol{x},\boldsymbol{m})$ expresses implementation and design cost now
    - $\boldsymbol{B}(\boldsymbol{x},\theta)$ expresses option value for exercising flexibility later

FAQ

Is it sound?	No, it’s light!
Is the model correct?	Maybe not, it’s a first cut
Is it complete?	No, it’s opportunistic
Is it universal?	No, it takes user view of value
Does it work?	Maybe. We’ll see
So, is it useful?	We already think so
What does it not do?	Things that need code

We need better ways to analyze a software design and predict the value its implementation will offer to a customer or to its producer

Many techniques provide early, but selective, evaluation

They are not organized to use systematically

Economic view offers promise for unification