

Modeling Software Evolution with Game Theory

Vibha Sazawal and Nikita Sudan

UM Institute for Advanced Computing Studies (UMIACS)
University of Maryland, College Park

16 May, 2009

Outline

- 1 Motivation
- 2 Introduction to Game Theory
- 3 Basic Software Evolution Game
- 4 Case Study: Sun's Support for World Calendars
- 5 Discussion and Related Work
- 6 Conclusion

The software process as an economic problem

- Investment of human and technical resources over time
- How should resources be allocated over time to maximize value?
- How should resource allocation change in response to new requirements or market trends?

Economic theories applied to software engineering

- Boehm's *Software Engineering Economics*
 - net present value
 - value of information
- Software quality attributes as options
- Sequencing of minimum marketable features
- and others...

Game Theory: a complementary economic approach

Games are a series of moves between players

- series of moves → sequence of process decisions
- players → a software engineering team, customers, and competitors
- A perfect match to the software engineering process?

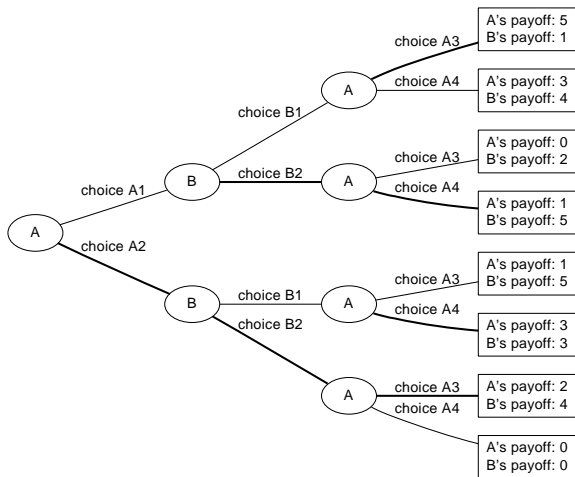
Our Approach: the Basic Software Evolution Game

- a simple two-player game that requires a minimum of economic theory to understand
- purpose: software engineering teams can use the game to predict possible outcomes and plan accordingly

Extensive-form sequential game

- A tree of decisions
- Nodes represent players, edges their possible decisions
- Leaves of tree store the payoff for each player
- Optimal path solved via backwards induction

Example of a game



Basic Software Evolution Game

- Two players: Software Engineering Team (SE Team) and Customer (World)
- First move: SE Team chooses among n initial designs
- Second move: World chooses among m change requests
- Third move: SE Team responds to request
- and so on...

Basic SE Game: $n=2, m=2$



Payoffs for SE Team and World

Variables

- π is payoff,
- U is utility
- P is payment
- C is cost
- *base* is the initial implementation
- Δ is a change (world-requested or requirements-preserving)

Payoffs for the SE Team and World (2)

SE Team payoff:

$$\pi_{SETeam} = U(P_{base}) + \sum_{i=1}^{tot\Delta} U(P_{\Delta_i}) - U(C_{base}) - \sum_{i=0}^{tot\Delta} U(C_{\Delta_i} | \Delta_1 \dots \Delta_{i-1}) \quad (1)$$

World payoff:

$$\pi_{World} = U(base) + \sum_{i=1}^{tot\Delta} U(\Delta_i | \Delta_1 \dots \Delta_{i-1}) - U(P_{base}) - \sum_{i=1}^{tot\Delta} U(P_{\Delta_i}) \quad (2)$$

How to estimate values for these variables?

- Multi-criteria utility estimation: a variety of strategies for finding points of indifference
- Cost estimation: COCOMO II or other techniques
- Pricing: compute estimates with the help of marketing

How to estimate values for these variables? (2)

Lightweight Approach

- Define utilities relationally
- Introduce constants $U_{World}(base)$ and $U_{SETeam}(P_{base})$
- Utilities associated with changes can be described as fractional amounts of the base utilities

Case Study: java.util.Calendar

- Java 1.1: support for the Gregorian calendar only
- Java 1.4: limited support for Thai Buddhist calendar
- Java 6 (2006): limited support for Japanese Imperial calendar

Many requests for alternate calendar support

- Bug ID 4609228, submitted December 2001
- Requests for Japanese, Arabic, Persian, and Hebrew calendars
- Bug comments point to difficulties in accommodating world calendars with current codebase

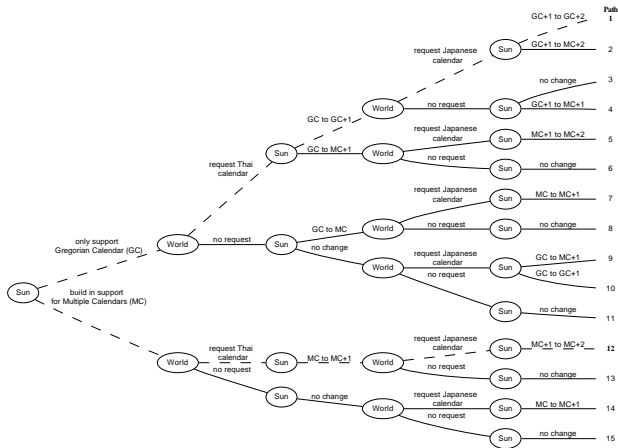
Competition: IBM's ICU4J Project

- 2005: open source project launched to add the Persian Calendar to ICU4J's classes
- As of now: ICU4J supports Chinese, Coptic, Hebrew, Indian, Islamic, Thai Buddhist, Japanese Imperial, and Taiwanese calendars
- Bug comments suggest IBM has gained goodwill for its support of alternate calendars
 - Business disadvantage for Sun?

Modelling java.util.Calendar evolution using the Basic SE Evolution Game

- Sun's initial design choice: Gregorian Calendar support (GC) or Multi-Calendar support (MC)
- In response to the World's requests
 - Sun can make incremental changes to GC, OR
 - Sun can restructure from GC to MC as it makes an incremental change
 - Once Sun has MC, additional calendars can be added at low cost

Sun java.util.Calendar Evolution as a Game



Payoffs for Sun Case Study

- Java is free but it has strategic value for Sun
- We model goodwill obtained from Calendar support
 - Sun obtains goodwill (G) from the Gregorian calendar and additional goodwill for other calendars
 - If IBM's ICU4J offers support first, Sun obtains less goodwill for providing support later
- Sun incurs costs (C) when it makes changes
- World obtains utility (U) from each additional calendar supported by Sun
 - If IBM's ICU4J offers support first, World's utility gained from Sun's support goes down

Examples of payoffs

	Sun chooses GC
Path 1	Sun: $(G_{base} + G_{thai} + \alpha G_{japanese})$ $-(C_{GC+1 GC} + C_{GC+2 GC+1} + C_{GC})$ World: $U_{base} + U_{thai} + \beta U_{japanese}$
Path 5	Sun: $(G_{base} + G_{thai} + G_{japanese})$ $-(C_{MC+1 GC} + C_{MC+2 MC+1} + C_{GC})$ World: $U_{base} + U_{thai} + U_{japanese}$
	Sun chooses MC
Path 12	Sun: $(G_{base} + G_{thai} + G_{japanese})$ $-(C_{MC+1 MC} + C_{MC+2 MC+1} + C_{MC})$ World: $U_{base} + U_{thai} + U_{japanese}$

Simplifying Assumptions

Assumption	Rationale
$C_{MC+1 MC} = 0$ $C_{MC+2 MC+1} = 0$	additions of new calendars are trivial when MC is the base design
$C_{MC} = 2C_{GC}$	design support for multiple calendars takes more time and expense than designing only for the Gregorian calendar
$\alpha = \frac{1}{8}, \beta = \frac{1}{4}$	$\beta > \alpha$ because legacy code using Sun's classes won't need to be converted to a competitor's class
$C_{GC+1 GC} = \frac{1}{8}C_{GC}$ $C_{GC+2 GC+1} = \frac{1}{8}C_{GC}$	incremental changes are expensive to add to GC
$C_{MC GC} = \frac{3}{2}C_{GC}$ $C_{MC+1 GC} = \frac{3}{2}C_{GC}$	expensive to restructure from GC to MC
$C_{MC+1 GC+1} = C_{MC GC} + \frac{1}{8}C_{GC}$ $C_{MC+2 GC+1} = C_{MC GC} + \frac{1}{8}C_{GC}$	cost of restructuring change is higher because of previous incremental change

Results: Heavily Dependent on Goodwill Estimation!

Goodwill obtained from		World's utility from		Solution
Thai Calendar	Japanese Calendar	Thai Calendar	Japanese Calendar	
$0.01 \times G_{base}$	$0.125 \times G_{base}$	$0.01 \times U_{base}$	$0.125 \times U_{base}$	Path 1: build GC, incr. extend
$1 \times G_{base}$	$1 \times G_{base}$	$0.01 \times U_{base}$	$0.125 \times U_{base}$	Path 12: build MC initially
$1 \times G_{base}$	$1 \times G_{base}$	$1 \times U_{base}$	$1 \times U_{base}$	Path 12: build MC initially

How well does the Basic Software Evolution Game scale up to more complex scenarios?

- As number of design choices, change requests, go up?
- As legacy projects accrue multiple versions?
- As there are multiple competitors?
- As we expand the game to include more of the software lifecycle?

Game theory can accommodate these changes, but the games become large and unwieldy.

Related Work: Theory W

Theory W

- emphasizes software processes in which all stakeholders come out a winner.

Difference between Theory W and classic game theory

- Theory W encourages negotiation to find a sequence of process steps where everyone wins
- In game theory, rational actors only look out for themselves

A game theoretic framework **can** be used to find a path with high payoffs for all, **if** stakeholders cooperate.

Conclusion

Design/maintenance decisions have business implications.

Lightweight game theory can help SE teams plan.

We presented the evolution of `java.util.Calendar` as an instantiation of our Basic Software Evolution Game.

The game expressively models

- What actually happened
- Circumstances when alternate design/maintenance decisions would be optimal

There is a lot of potential for game theoretic approaches to software process!

Acknowledgements

We gratefully thank

- Our anonymous reviewers
- Sun, for their public bug database