

SOFTWARE PROJECT

LECTURE 3

WOUTER SWIERSTRA

LAST TIME

- ESTABLISHING A PRODUCT BACKLOG
 - PLANNING ITERATIONS

TODAY

- RISKS, CONSTRAINTS, AND QUALITY ATTRIBUTES
 - SOFTWARE ARCHITECTURE

YOUR CURRENT SITUATION...

- YOU HAVE A PRIORITIZED PRODUCT BACKLOG.
- YOU'VE HAD A PLANNING SESSION TO FILL YOUR FIRST SPRINT BACKLOG.
- YOU'RE AIMING TO DEVELOP A FIRST PROTOTYPE IN THE FIRST ITERATIONS.

WHAT CAN POSSIBLY GO WRONG?

A close-up, slightly blurred photograph of a glass bowl filled with various colored marshmallows. The colors include shades of purple, blue, green, and pink. The lighting is soft, creating a gentle glow around the marshmallows.

RISK

WHAT IS YOUR MARSHMALLOW?

RISKS

RISK IS DETERMINED BY PERCEIVED PROBABILITY AND PERCEIVED IMPACT.

IT IS IMPOSSIBLE TO ACCURATELY ESTIMATE EITHER FACTOR.
INSTEAD USE THIS TO DECIDE THE RISKS ON WHICH TO FOCUS.

- IT IS VERY LIKELY THAT YOU WILL GET YOUR ESTIMATES WRONG FOR THE FIRST SPRINT. THIS ISN'T SO BAD.

RISK ASSESSMENT FOR SOFTWARE PROJECTS

- > YOU HAVE LITTLE EXPERIENCE WORKING IN A TEAM:
 - > YOU ARE EXPOSED TO UNFAMILIAR TECHNOLOGY:
 - > YOU HAVE TO WORK IN AN UNFAMILIAR DOMAIN:
- > BESIDES THE PROJECT, YOU ALL HAVE OTHER COURSES, JOBS, ETC.

YOU **CANNOT** CHANGE THIS, BUT YOU **CAN** ACT ACCORDINGLY.

ABOUT RISK

AS COMPUTER SCIENTISTS, WE ARE TRAINED TO FOCUS ON
TECHNICAL RISK

- > HOW TO IMPLEMENT FEATURE X?
- > HOW TO INTERFACE WITH SYSTEM Y?
 - > HOW TO CALL LIBRARY Z?

SOMETIMES THE IMPORTANT RISK IS NOT IN THE TECHNICAL PART.

NON-ENGINEERING RISKS

- > 'THE LEAD DEVELOPER'S HOBBIES ARE BASE-JUMPING AND CAGE FIGHTING.'
- > 'SENIOR VP HATES OUR MANAGER.'
- > 'THE CUSTOMER DOESN'T KNOW WHAT HE WANTS.'
- > 'BOB AND ALICE REFUSE TO WORK TOGETHER.'

**BUT THAT
WON'T HAPPEN
TO US.**

RISKS: EXAMPLE

IN A RECENT PROJECT THERE WAS AN OUTSPOKEN STUDENT WHO STRONGLY PUSHED TECHNOLOGY X.

WHEN I ASKED THEM ABOUT THEIR CHOICE FOR X, THEY LIST THE 'STANDARD' ADVANTAGES OF USING X, BUT HADN'T THOUGHT ABOUT WHETHER IT WAS THE RIGHT TOOL FOR THIS PROJECT.

RISKS: EXAMPLE

IN A RECENT PROJECT THERE WAS AN OUTSPOKEN STUDENT WHO STRONGLY PUSHED TECHNOLOGY X.

WHEN I ASKED THEM ABOUT THEIR CHOICE FOR X, THEY LIST THE 'STANDARD' ADVANTAGES OF USING X, BUT HADN'T THOUGHT ABOUT WHETHER IT WAS THE RIGHT TOOL FOR THIS PROJECT.

THIS STUDENT ABANDONED THE SOFTWARE PROJECT HALFWAY THROUGH. THE REMAINING STUDENTS THREW AWAY THEIR CODE

THE BOTTOM LINE

YOU NEED TO IDENTIFY RISKS...

... AND TAKE DECISIONS TO MINIMIZE THEM.

... MANY OF THESE DECISIONS ARE ARCHITECTURAL.

JUST ENOUGH SOFTWARE ARCHITECTURE

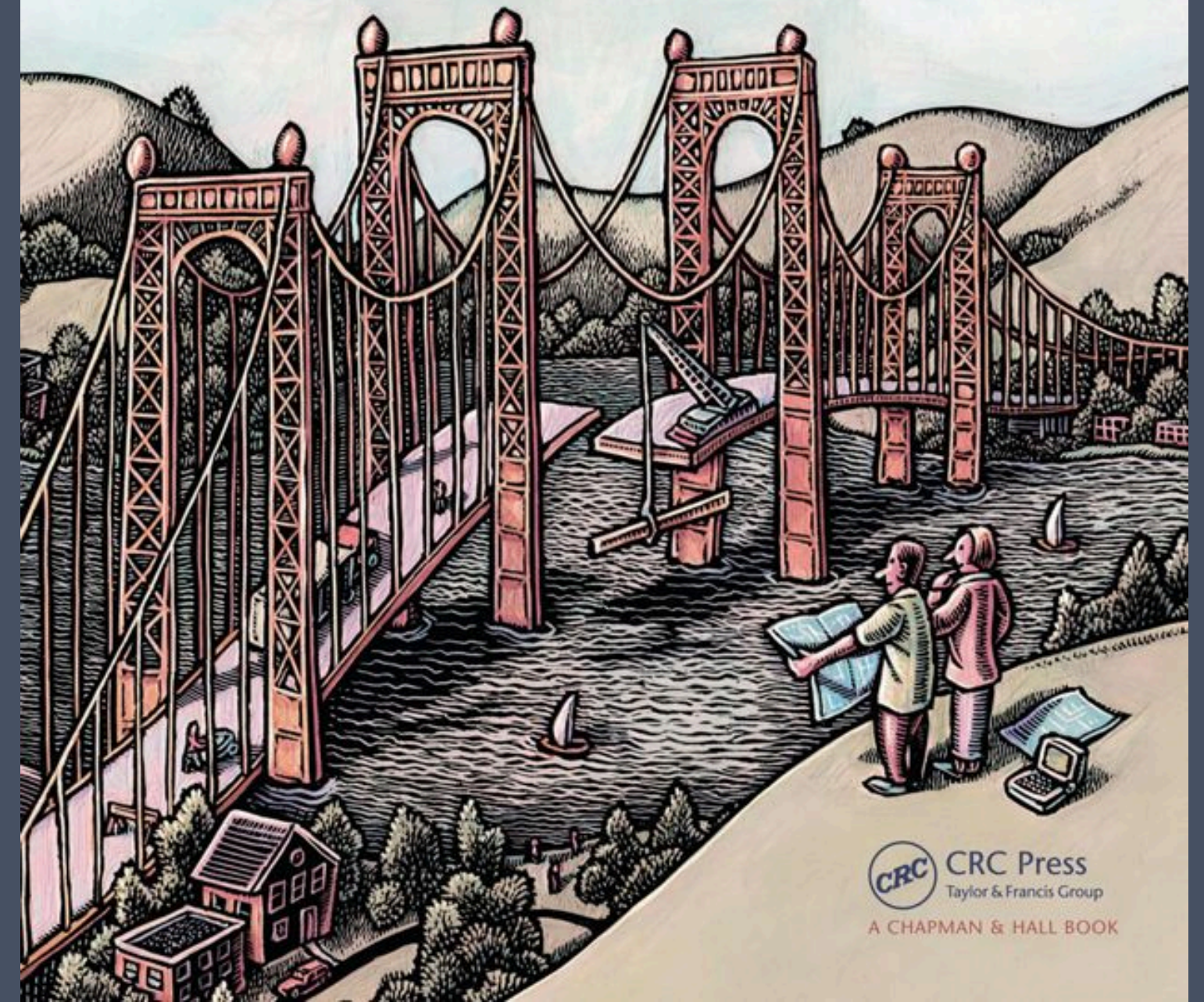
A RISK-DRIVEN APPROACH


JUST ENOUGH SOFTWARE ARCHITECTURE

A RISK-DRIVEN APPROACH

GEORGE FAIRBANKS

FOREWORD BY DAVID GARLAN



 **CRC Press**
Taylor & Francis Group
A CHAPMAN & HALL BOOK

WHAT IS SOFTWARE ARCHITECTURE?

THE HIGHEST-LEVEL BREAKDOWN OF A SYSTEM INTO ITS PARTS; THE DECISIONS THAT ARE HARD TO CHANGE; THERE ARE MULTIPLE ARCHITECTURES IN A SYSTEM; WHAT IS ARCHITECTURALLY SIGNIFICANT CAN CHANGE OVER A SYSTEM'S LIFETIME; AND, IN THE END, ARCHITECTURE BOILS DOWN TO WHATEVER THE IMPORTANT STUFF IS.

MARTIN FOWLER'S DEFINITION OF SOFTWARE ARCHITECTURE

THE FUNDAMENTAL ORGANIZATION OF A SYSTEM EMBODIED
IN ITS COMPONENTS, THEIR RELATIONSHIPS TO EACH OTHER
AND TO THE ENVIRONMENT AND THE PRINCIPLES GUIDING
ITS DESIGN AND EVOLUTION

SOFTWARE ARCHITECTURE (IEEE 1471)

THE SOFTWARE ARCHITECTURE OF A COMPUTING SYSTEM IS THE SET OF STRUCTURES NEEDED TO REASON ABOUT THE SYSTEM, WHICH COMPRISE SOFTWARE ELEMENTS, RELATIONS AMONG THEM, AND PROPERTIES OF BOTH

SOFTWARE ENGINEERING INSTITUTE (CLEMENTS ET AL., 2010)

EXAMPLE ARCHITECTURE: RACKSPACE

RACKSPACE IS A COMPANY THAT MANAGES HOSTED EMAIL SERVERS.

ENGINEERS MUST SEARCH LOG FILES TO DIAGNOSE PROBLEMS.

RACKSPACE DEVELOPED THREE DIFFERENT VERSIONS OF THIS TOOL.

RACKSPACE - I

A SHELL SCRIPT TO SSH TO EACH MACHINE, AND GREP THE MAIL LOG.

ENGINEERS CAN USE DIFFERENT GREPS TO PERFORM DIFFERENT QUERIES.

AS THE NUMBER OF SEARCHES INCREASED, OVERHEAD BECAME NOTICEABLE.

IT REQUIRES AN ENGINEER, RATHER THAN TECH SUPPORT TO

RACKSPACE - II

MOVE LOGS OFF EMAIL SERVERS AND MAKE IT SEARCHABLE BY TECH SUPPORT THROUGH A WEB INTERFACE.

ALL LOGS COLLECTED ON A CENTRAL MACHINE, AND LOADED INTO A DATABASE.

THE CENTRALIZED SERVER IS CONSTANTLY UNDER HEAVY LOAD, AS IT WAS CONSTANTLY ADDING NEW ENTRIES.

SLOWDOWN AND RANDOM FAILURES, RESULTING IN THE LOSS OF

RACKSPACE – III

LOG DATA SAVED INTO A DISTRIBUTED FILE SYSTEM, SPREAD OVER MANY MACHINES, INDEXED IN PARALLEL.

THREE COPIES OF EVERYTHING, SPREAD OVER DIFFERENT MACHINES.

CAPABLE OF HANDLING 140 GIGABYTES OF DATA GENERATED PER DAY, SIX TERABYTES OF DATA TOTAL.

QUERIES OVER WEBINTERFACE WERE FAST AND RELIABLE.



WHAT DO THESE
ARCHITECTURES
SHARE? WHAT
MAKES THEM

WHY SOFTWARE ARCHITECTURE?

SOFTWARE ARCHITECTURE IS THE SKELETON OF A SYSTEM

YOUR ARCHITECTURE DEFINES THE OVERALL STRUCTURE OF YOUR SYSTEM.

YOUR REQUIREMENTS AND CONSTRAINTS DETERMINE SUITABLE ARCHITECTURE.

RACKSPACE:

WHY SOFTWARE ARCHITECTURE?

ARCHITECTURE IS (MOSTLY) ORTHOGONAL TO FUNCTIONALITY.

THERE IS NO SINGLE 'BEST' ARCHITECTURE.

THERE ARE MANY WAYS TO DELIVER THE SAME FUNCTIONALITY.

DEPENDING ON YOUR NEEDS, YOU NEED TO CHOOSE THE BEST ARCHITECTURE FOR YOUR PROJECT.

WHY SOFTWARE ARCHITECTURE?

ARCHITECTURE CONSTRAINS SYSTEMS

YOU HAVE THE FREEDOM TO CHOOSE WHICH CONSTRAINTS ARE IMPORTANT.

- > EXCHANGING DATA FREELY? OR ONLY OVER SECURE CONNECTIONS WITH TRUSTED PARTIES?
- > SCALABLE? OR EASY TO SET UP AND MAINTAIN?

YOUR CHOICE OF ARCHITECTURE INFLUENCES WHAT A SYSTEM

CONSTRAINTS

WHERE (NON-FUNCTIONAL) REQUIREMENTS AND USER STORIES SPECIFY YOUR GOAL, **CONSTRAINTS** LIMIT THE SOLUTION SPACE.

- **TECHNICAL: EXISTING SYSTEMS, LANGUAGES USED, PLATFORMS, ETC.**
- **ORGANIZATIONAL: CLIENT AVAILABILITY, EXISTING BUSINESS PROCESSES, ...**

A HARD TRUTH

YOU ARE NOT A SOFTWARE ARCHITECT.

AFTER THIS COURSE, YOU WILL STILL NOT BE SOFTWARE
ARCHITECT.

AFTER THE COURSE ON SOFTWARE ARCHITECTURE, YOU WILL
STILL NOT BE A SOFTWARE ARCHITECT.

COME BACK IN TWENTY YEARS.

'REAL' SOFTWARE ARCHITECTURE

MANY SOFTWARE ARCHITECTURE DOCUMENTS ARE GIVE AN EXTENSIVE OVERVIEW OF MANY DIFFERENT VIEWS OF A SYSTEM:

- > PHYSICAL MACHINES RUNNING SOFTWARE COMPONENTS:
 - > DECOMPOSITION INTO COMPONENTS AND CLASSES:
 - > DYNAMIC BEHAVIOUR OF PROCESSES:
 - > USER'S PERSPECTIVE.

THESE DOCUMENTS ARE LENGTHY AND COMPLICATED AND

THE BOTTOM LINE

YOUR PROJECT IS 'SMALL ENOUGH' TO DO WITHOUT A THOROUGHLY THOUGHT OUT, FULLY-DOCUMENTED ARCHITECTURE.

ARCHITECTURE DOCUMENTS ARE TOO BIG A HAMMER FOR OUR PROJECTS.

WHAT I WOULD ENCOURAGE YOU TO DO IS:

1. IDENTIFY RISKS, CONSTRAINTS, AND QUALITY ATTRIBUTES.

RISK SCENARIO

ONE SOFTWARE PROJECT HAD TO DEVELOP SOFTWARE TO HELP PLAN WHERE TO BUILD WIND TURBINE PARKS.

THEY WERE RESPONSIBLE FOR THE VISUALIZATION: THE ACTUAL CALCULATION AND DATA MANAGEMENT WOULD BE DONE BY A THIRD PARTY.

IT WAS NOT READY YET... INTEGRATION WAS GOING TO BE A HUGE RISK.

STRATEGIES FOR DEALING WITH RISK

- AVOIDANCE – TAKE PRECAUTIONS: FAVORING STABLE TECHNOLOGY, WORKING WITH MOCKS AND STUBS, ETC.
- MINIMIZATION – SHARE CODE OWNERSHIP TO AVOID STAGNATION WHEN ONE PERSON IS ILL: CREATING PROTOTYPES TO DEMO TO THE CUSTOMER.
- CONTINGENCY PLANS – WE ACCEPT THIS RISK, BUT PROVIDE AN ALTERNATIVE: HOW WILL WE REDUCE THE SCOPE OF THE

RISK IDENTIFICATION EXERCISE

YOU HAVE YOUR MOSCOW-ED USER STORIES.

EVERY TEAM MEMBER GETS 1,2,3 PLANNING POKER CARDS.

EVERYONE ASSIGNS THESE CARDS TO THE TOP STORIES IN THE
PRODUCT BACKLOG.

RECORD THIS RISK-VALUE ASSESSMENT.

RISK IDENTIFICATION EXERCISE

DISCUSS THE RESULTS OF IDENTIFYING THE HIGH-RISK STORIES:

- HOW WILL YOU COPE WITH THIS RISK?
- WHAT NEW STORIES WILL YOU INTRODUCE?
- HOW DOES THIS INSIGHT EFFECT THE PRIORITIES IN YOUR BACKLOG?



WHERE TO START?

START WITH HIGH-VALUE STORIES.

BUT SHOULD YOU FOCUS ON HIGH-RISK OR LOW-RISK STORIES?



WHERE TO START?

START WITH HIGH-VALUE STORIES.

BUT SHOULD YOU FOCUS ON HIGH-RISK OR LOW-RISK STORIES?

HIGH-RISK STORIES HAVE PRIORITY.

QUALITY ATTRIBUTES

THERE ARE HUGE CATALOGS OF SOFTWARE QUALITY ATTRIBUTES, DESCRIBING THE NON-FUNCTIONAL PROPERTIES OF SOFTWARE SYSTEMS.

THESE ARE TYPICALLY NOT DOCUMENTED IN USER STORIES.
BUT YOUR SYSTEM DOES HAVE NON-FUNCTIONAL REQUIREMENTS.
THESE NON-FUNCTIONALS ARE NEED TO BE BAKED INTO YOUR

EXAMPLE: SECURITY

SUPPOSE YOUR SYSTEM MUST HANDLE DATA THAT YOU MUST KEEP PRIVATE.

THERE ARE SEVERAL ARCHITECTURAL TACTICS TO HELP ACHIEVE THIS:

- AUTHENTICATION – NEGATIVE IMPACT ON USABILITY
- ENCRYPTION – NEGATIVE IMPACT ON PERFORMANCE (AND

EXAMPLE: AVAILABILITY

SIMILARLY, IF YOUR SYSTEM **MUST** BE AVAILABLE AT ALL TIMES:

- FAULT DETECTION: PING, HEARTBEAT, ETC.
- FAULT RECOVERY: VOTING/POLLING, ACTIVE REDUNDANCY (HOT RESTART), PASSIVE REDUNDANCY (WARM RESTART), SPARE SERVERS, ETC.
- FAULT PREVENTION: CHECKSUMS, ERROR DETECTION, ETC.

QUALITY ATTRIBUTES

THERE IS ALWAYS THE TEMPTATION TO READ THROUGH A LIST OF QUALITY ATTRIBUTES AND SELECT AN ARBITRARY NUMBER OF PROPERTIES YOU CONSIDER TO BE IMPORTANT.

DON'T TREAT QUALITY ATTRIBUTES LIKE YOUR CHRISTMAS WISHLIST.

IF SOMEONE IS HOLDING A GUN TO YOUR HEAD, WHICH IS MORE IMPORTANT:



WHY DO PROJECTS
FAIL?
WHAT ARE YOU
GOING TO DO

META-RETROSPECTIVE

YOU NOW KNOW ALL YOU NEED TO KNOW TO SELF-ORGANIZE
YOUR TEAM AND PLAN YOUR SOFTWARE PROJECT.

THE SCRUM/AGILE APPROACH IS PROBABLY THE MOST POPULAR
SOFTWARE DEVELOPMENT METHODOLOGY AT THE MOMENT.

DOES EVERYONE AGREE THAT THEY WORK?

Bertrand Meyer



Agile!

The Good, the Hype and the Ugly

 Springer

BERTRAND MEYER

AGILE! THE GOOD, THE HYPE AND THE UGLY

E-BOOK AVAILABLE IN THE LIBRARY

MEYER: THE BAD AND THE UGLY

- REJECTION OF ANY UPFRONT TASKS
- USER STORIES OF ONLY REQUIREMENTS – RESULTING SOFTWARE MAY BE HARD TO ADAPT AND YOU MIGHT MISS OTHER IMPORTANT FEATURES.
- FEATURE BASED DEVELOPMENT IGNORES FOUNDATION WORK

MEYER: THE HYPED

- THERE IS NO CREDIBLE EVIDENCE THAT PAIR PROGRAMMING WORKS
 - FEW TEAMS ARE EXPERIENCED ENOUGH TO BE SELF-ORGANIZING
- PLANNING POKER MAY DROWN THE VOICE OF THE EXPERTS
 - CROSS-FUNCTIONAL TEAMS IGNORE INDIVIDUALS

MEYER: THE GOOD & BRILLIANT

- SHORT DAILY MEETINGS
- REFACTORING IS IMPORTANT (BUT CANNOT REPLACE DESIGN)
 - CONTINUOUS INTEGRATION AND REGRESSION TESTING
 - SHORT-TIME BOXED ITERATIONS
 - REFINING WORKING SOFTWARE

ERIK MEIJER
SOFTWARE IS
EATING THE



DO YOU AGREE
WITH ERIK?



VIDEO AFTER
THE PROJECT.
DO YOU STILL

HOMework - I

- IDENTIFY THE HIGH-RISK HIGH-VALUE STORIES.
- WHAT NON-ENGINEERING RISKS CAN YOU IDENTIFY? HOW WILL YOU HANDLE THEM? FILL YOUR PRODUCT BACKLOG ACCORDINGLY.
- WHAT CONSTRAINTS CAN YOU IDENTIFY?

HOMework - II

- LOOK AT THE LISTS OF QUALITY ATTRIBUTES:
 - WHICH ARE CRUCIAL FOR YOUR PROJECT?
 - HOW BIG IS THE ASSOCIATED RISK?
- WHAT DECISIONS NEED TO BE MADE TO HANDLE THIS RISK?
- WHICH ARCHITECTURAL PATTERNS OR TACTICS CAN HELP?
- WHICH OTHER QUALITY ATTRIBUTES MAY BE NEGATIVELY

**NEXT TIME:
YOUR
PRESENTATIONS!**